Defining Efficiency

- Classical definition of efficiency:
  An algorithm is efficient iff it runs in polynomial time on a serial computer.

- Runtimes of “efficient” algorithms:
  \[ O(n) \quad O(n \log n) \quad O(n^3 \log^2 n) \quad O(n^{1.00000000000}) \]

- Runtimes of “inefficient” algorithms:
  \[ O(2^n) \quad O(n!) \quad O(1.00000001^n) \]

Tractable and Intractable Problems

- A problem is called **tractable** iff there is an efficient (i.e. polynomial-time) algorithm that solves it.

- A problem is called **intractable** iff there is no efficient algorithm that solves it.

- **Intractable problems are common.** We need to discuss how to approach them when you come across them in practice.
**Easy problems**

- Given composite \( N \) and \( x \) in \( \mathbb{Z}_N \) find \( x^{-1} \) in \( \mathbb{Z}_N \)

- Given prime \( p \) and polynomial \( f(x) \) in \( \mathbb{Z}_p[x] \)
  find \( x \) in \( \mathbb{Z}_p \) s.t. \( f(x) = 0 \) in \( \mathbb{Z}_p \) (if one exists)
  
  Running time is linear in \( \deg (f) \).

... but many problems are difficult

**Intractable problems with primes**

Fix a prime \( p > 2 \) and \( g \) in \( \mathbb{Z}_p^* \) of order \( q \).

Consider the function:

\[ x \mapsto g^x \text{ in } \mathbb{Z}_p \]

Now, consider the inverse function (discrete logarithm):

\[ \text{Dlog}_g (g^x) = x \]

where \( x \) in \( \{0, \ldots, q-2\} \)

Example:

<table>
<thead>
<tr>
<th>( g \times 1 )</th>
<th>( 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{Dlog}_2 (\cdot)</td>
<td>( 0, 1, 8, 2, 4, 9, 7, 3, 6, 5 )</td>
</tr>
</tbody>
</table>

**DLOG: more generally**

Let \( G \) be a finite cyclic group and \( g \) a generator of \( G \)

\[ G = \{ 1, g, g^2, g^3, \ldots, g^{q-1} \} \quad (q \text{ is called the order of } G) \]

**Def:** We say that DLOG is hard in \( G \) if for all efficient alg. \( A \):

\[ \Pr_{g \leftarrow G, x \leftarrow Z_q} \left[ A( G, q, g, g^x) = x \right] < \text{negligible} \]

Example candidates:

1. \( \mathbb{Z}_p^* \) for large \( p \),
2. Elliptic curve groups mod \( p \)

**An application: collision resistance**

Choose a group \( G \) where Dlog is hard (e.g. \( \mathbb{Z}_p^* \) for large \( p \))

Let \( q = |G| \) be a prime. Choose generators \( g, h \) of \( G \)

For \( x, y \in \{1, \ldots, q\} \) define \( H(x, y) = g^x \cdot h^y \) in \( G \)

**Lemma:** finding collision for \( H(\cdot, \cdot) \) is as hard as computing \( \text{Dlog}_g (h) \)

Proof: Suppose we are given a collision \( H(x_0, y_0) = H(x_1, y_1) \)
then \( g^{x_0 \cdot h^{y_0}} = g^{x_1 \cdot h^{y_1}} \)

Hence \( g^{x_0 \cdot h^{y_0}} = g^{x_1 \cdot h^{y_1}} \Rightarrow g^{x_0 \cdot h^{y_1} \cdot h^{-y_0}} \Rightarrow h = g^{x_0 \cdot h^{-y_0} \cdot h^{y_1}} \)
Intractable problems with composites

Consider the set of integers: \( \mathbb{Z}_2(n) := \{ N = p \cdot q \mid p, q \text{ are } n\text{-bit primes} \} \)

**Problem 1:** Factor a random \( N \) in \( \mathbb{Z}_2(n) \) (e.g. for \( n=1024 \))

**Problem 2:** Given a polynomial \( f(x) \) where \( \deg(f) > 1 \) and a random \( N \) in \( \mathbb{Z}_2(n) \), find \( x \) in \( \mathbb{Z}_N \) s.t. \( f(x) = 0 \) in \( \mathbb{Z}_N \)

The factoring problem

Gauss (1805): "The problem of distinguishing prime numbers from composite numbers and of resolving the latter into their prime factors is known to be one of the most important and useful in arithmetic."

Best known alg. (NFS): run time \( \exp(O(\sqrt[3]{n}) \) for \( n \)-bit integer

Current world record: RSA-768 (232 digits)
- Work: two years on hundreds of machines
- Factoring a 1024-bit integer: about 1000 times harder
  \( \Rightarrow \) likely possible this decade

Public key encryption

Bob: generates \( (PK, SK) \) and gives \( PK \) to Alice

Public key encryption

Bob: generates \( (PK, SK) \) and gives \( PK \) to Alice
Applications

Session setup (for now, only eavesdropping security)

Alice

\[ \text{Generate } (pk, sk) \]

\[ x \]

pk

\[ E(pk, x) \]

Bob

choose random \( x \) (e.g. 48 bytes)

Non-interactive applications: (e.g. Email)

- Bob sends email to Alice encrypted using \( pk_{\text{Alice}} \)
- Note: Bob needs \( pk_{\text{Alice}} \) (public key management)

Public key encryption

**Def:** a public-key encryption system is a triple of algs. \((G, E, D)\)

- \(G():\) randomized alg. outputs a key pair \((pk, sk)\)
- \(E(pk, m):\) randomized alg. that takes \( m \in M \) and outputs \( c \in C \)
- \(D(sk, c):\) det. alg. that takes \( c \in C \) and outputs \( m \in M \) or \( \perp \)

Consistency: \( \forall (pk, sk) \) output by \( G : \)

\[ \forall m \in M : D(sk, E(pk, m) ) = m \]

Public Key Encryption

from trapdoor permutations

Constructions from Trapdoor Permutation

**Def:** a trapdoor func. \( X \rightarrow Y \) is a triple of efficient algs. \((G, F, F^{-1})\)

- \(G():\) randomized alg. outputs a key pair \((pk, sk)\)
- \(F(pk,:):\) det. alg. that defines a function \( X \rightarrow Y \)
- \(F^{-1}(sk,:):\) defines a function \( Y \rightarrow X \) that inverts \( F(pk,:) \)

More precisely: \( \forall (pk, sk) \) output by \( G \)

\[ \forall x \in X : F^{-1}(sk, F(pk, x) ) = x \]
Public-key encryption from TDFs

- $(G, F, F^{-1})$: secure TDF $X \rightarrow Y$
- $(E_s, D_s)$: symmetric auth. encryption defined over $(K, M, C)$
- $H: X \rightarrow K$ a hash function

We construct a pub-key enc. system $(G, E, D)$:

**Key generation $G$:** same as $G$ for TDF

**$E(pk, m)$:**

$x \leftarrow X$, $y \leftarrow F(pk, x)$
$k \leftarrow H(x)$, $c \leftarrow E_s(k, m)$
output $(y, c)$

**$D(sk, (y, c))$:**

$x \leftarrow F^{-1}(sk, y)$,
$k \leftarrow H(x)$, $m \leftarrow D_s(k, c)$
output $m$

Public Key Encryption from RSA trapdoor permutations

Let $N = p \cdot q$ where $p, q$ are prime

$Z_N = \{0, 1, 2, ..., N-1\}$; $(Z_N)^* = \{\text{invertible elements in } Z_N\}$

**Facts:**

- $x \in Z_N$ is invertible $\iff \gcd(x, N) = 1$
- Number of elements in $(Z_N)^*$ is $\varphi(N) = (p-1)(q-1) = N-p-q+1$

**Euler's thm:**

$\forall x \in (Z_N)^*: x^{\varphi(N)} = 1$

Review: arithmetic mod composites

- $Z_N = \{0, 1, 2, ..., N-1\}$; $(Z_N)^* = \{\text{invertible elements in } Z_N\}$
The RSA trapdoor permutation


Very widely used:
- SSL/TLS: certificates and key-exchange
- Secure e-mail and file systems
- ... many others

G(): choose random primes p,q \approx 1024 bits. Set N=pq.

choose integers e, d s.t. e\cdot d = 1 (mod \varphi(N))

output \( pk = (N, e), \ sk = (N, d) \)

\[ F(pk, x) : \mathbb{Z}_N^* \rightarrow \mathbb{Z}_N^* ; \quad RSA(x) = x^e \quad (in \ Z_n) \]

\[ F^{-1}(sk, y) = y^d ; \quad y^d = RSA(x)^d = x^{ed} = x^{kp(N)^{k+1}} = \left(x^{\varphi(N)}\right)^k \cdot x = x \]

The RSA assumption

RSA assumption: RSA is one-way permutation

For all efficient algs. A:

\[ \Pr\left[ A(N,e,y) = y^{\frac{x}{e}} \right] < \text{negligible} \]

where p,q \in \mathbb{Z}_n. n-bit primes, N \neq pq, y^{\varphi(Z_n)}

RSA pub-key encryption (ISO std)

\{E_s, D_s\}: symmetric enc. scheme providing auth. encryption.

H: Z_n \rightarrow K where K is key space of (E_s, D_s)

- G(): generate RSA params: pk = (N, e), sk = (N, d)
- E(pk, m):
  1. choose random x in Z_n
  2. y \leftarrow RSA(x) = x^e , k \leftarrow H(x)
  3. output (y, E_s(k, m))
- D(sk, (y, c)):
  output D_s(H(RSA^{-1}(y)), c)

COMP 4420 - Kan Yang
Textbook RSA is insecure

Textbook RSA encryption:
- public key: \((N,e)\)
- secret key: \((N,d)\)

Encrypt: \(c \leftarrow m^e \mod N\)
Decrypt: \(c^d \mod N \rightarrow m\)

Insecure cryptosystem !!
- is not semantically secure and many attacks exist

⇒ The RSA trapdoor permutation is not an encryption scheme !

Different attack on textbook RSA

Suppose \(k\) is 64 bits: \(k \in \{0,\ldots,2^{64}\}\). Eve sees: \(c = k^e \mod N\)

If \(k = k_1 \cdot k_2\) where \(k_1, k_2 < 2^{34}\) (prob. \(\approx 20\%\)) then \(c/k_1^e = k_2^e \mod N\)

Step 1: build table: \(c/1^e, c/2^e, \ldots, c/2^{34}e\) . time: \(2^{34}\)
Step 2: for \(k_2 = 0,\ldots, 2^{34}\) test if \(k_2^e\) is in table. time: \(2^{34}\)
Output matching \((k_1, k_2)\).
Total attack time: \(\approx 2^{40} << 2^{64}\)

RSA encryption in practice

Never use textbook RSA.

RSA in practice (since ISO standard is not often used):

Main questions:
- How should the preprocessing be done?
- Can we argue about security of resulting system?

Public-Key Cryptography Standards (PKCS1 v1.5)

PKCS1 mode 2: {(encryption)

\[
\begin{array}{c|c|c|c|c}
\hline
& \text{random pad} & \text{FF} & \text{msg} \\
\hline
\text{msg} & & & \\
\hline
\end{array}
\]

RSA modulus size (e.g. 2048 bits)

- Resulting value is RSA encrypted
- Widely deployed, e.g. in HTTPS
Attack on PKCS1 v1.5 (Bleichenbacher 1998)

PKCS1 used in HTTPS:

Is this PKCS1?

Web Server

\[ C \]

\[ \rightarrow \]

\[ C = \text{Ciphertext} \]

\[ \rightarrow \]

\[ \text{Attacker} \]

\[ \text{yes: continue} \]

\[ \text{no: error} \]

\[ \Rightarrow \text{attacker can test if 16 MSBs of plaintext } \neq \text{'}02\text{' }\]

Chosen-ciphertext attack: to decrypt a given ciphertext \( C \) do:

- Choose \( r \in Z_n \). Compute \( c' \leftarrow (r \cdot \text{PKCS1}(m))^e \)
- Send \( c' \) to web server and use response

Baby Bleichenbacher

\[ \text{Attacker} \]

\[ c \]

\[ \text{yes: continue} \]

\[ \text{no: error} \]

Suppose \( N \) is \( N = 2^n \) (an invalid RSA modulus). Then:

- Sending \( c \) reveals \( \text{msb}(x) \)
- Sending \( 2^e \cdot c = (2x)^e \) in \( Z_n \) reveals \( \text{msb}(2x \text{ mod } N) = \text{msb}_2(x) \)
- Sending \( 4^e \cdot c = (4x)^e \) in \( Z_n \) reveals \( \text{msb}(4x \text{ mod } N) = \text{msb}_3(x) \)
- … and so on to reveal all of \( x \)

HTTPS Defense (RFC 5246)

Attacks discovered by Bleichenbacher and Klima et al. … can be avoided by treating incorrectly formatted message blocks … in a manner indistinguishable from correctly formatted RSA blocks. In other words:

1. Generate a string \( R \) of 46 random bytes
2. Decrypt the message to recover the plaintext \( M \)
3. If the PKCS#1 padding is not correct \( \text{pre_master_secret} = R \)

PKCS1 v2.0: OAEP

New preprocessing function: OAEP [BR94]

Thm [FOPS'01]: RSA is a trap-door permutation \( \Rightarrow \)
RSA-OAEP is CCA secure when \( H, G \) are random oracles

in practice: use SHA-256 for \( H \) and \( G \)

\[ \text{msg} \]

\[ \text{check pad on decryption. reject CT if invalid.} \]

\[ \text{plaintext to encrypt with RSA} \]

\( \in \{0,1\}^{n-1} \)
**OAEP Improvements**

**OAEP** [Shoup’01]

∀ trap-door permutation $F$

$F$-OAEP+ is CCA secure when $H, G, W$ are random oracles.

During decryption validate $W(m, r)$ field.

**SAEP** [B’01]

RSA ($e=3$) is a trap-door perm $\Rightarrow$

RSA-SAEP+ is CCA secure when $H, W$ are random oracle.

---

**RSA With Low public exponent**

To speed up RSA encryption use a small $e$: $c = m^e \pmod{N}$

- Minimum value: $e=3$ (gcd($e, \phi(N)$) = 1)
- Recommended value: $e=65537=2^{16}+1$

Encryption: 17 multiplications

**Asymmetry of RSA**: fast enc. / slow dec.

- ElGamal: approx. same time for both.

---

**How would you decrypt an SAEP ciphertext $ct$?**

- $(x, r) \leftarrow RSA^{-1}(sk, ct)$, $(m, w) \leftarrow x \oplus H(r)$, output $m$ if $w = W(m, r)$
- $(x, r) \leftarrow RSA^{-1}(sk, ct)$, $(m, w) \leftarrow r \oplus H(x)$, output $m$ if $w = W(m, r)$
- $(x, r) \leftarrow RSA^{-1}(sk, ct)$, $(m, w) \leftarrow x \oplus H(r)$, output $m$ if $r = W(m, x)$

---

**Key lengths**

Security of public key system should be comparable to security of symmetric cipher:

<table>
<thead>
<tr>
<th>Cipher key-size</th>
<th>RSA Modulus size</th>
</tr>
</thead>
<tbody>
<tr>
<td>80 bits</td>
<td>1024 bits</td>
</tr>
<tr>
<td>128 bits</td>
<td>3072 bits</td>
</tr>
<tr>
<td>256 bits (AES)</td>
<td><strong>15360</strong> bits</td>
</tr>
</tbody>
</table>
Implementation attacks

**Timing attack**: [Kocher et al. 1997], [BB’04]
The time it takes to compute $c^d \pmod{N}$ can expose $d$

**Power attack**: [Kocher et al. 1999]
The power consumption of a smartcard while it is computing $c^d \pmod{N}$ can expose $d$.

**Faults attack**: [BDL’97]
A computer error during $c^d \pmod{N}$ can expose $d$.
A common defense: check output. 10% slowdown.