DD2448 Foundations of Cryptography
Lecture 9

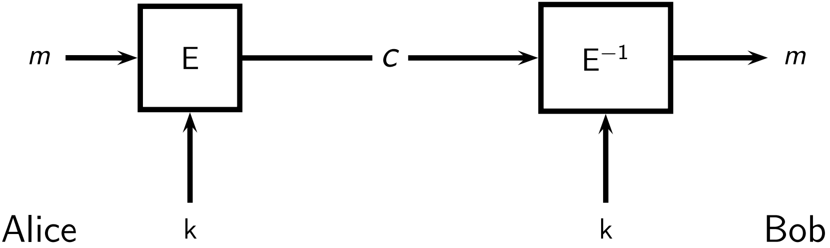
Douglas Wikström
KTH Royal Institute of Technology
dog@kth.se

April 7, 2020

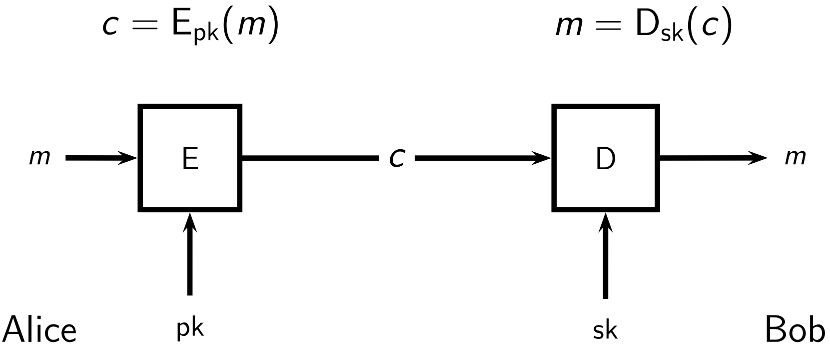
Public-Key Cryptography

Cipher (Symmetric Cryptosystem)

$$c = \mathsf{E}_{\mathsf{k}}(m) \qquad\qquad m = \mathsf{E}_{\mathsf{k}}^{-1}(c)$$



Public-Key Cryptosystem

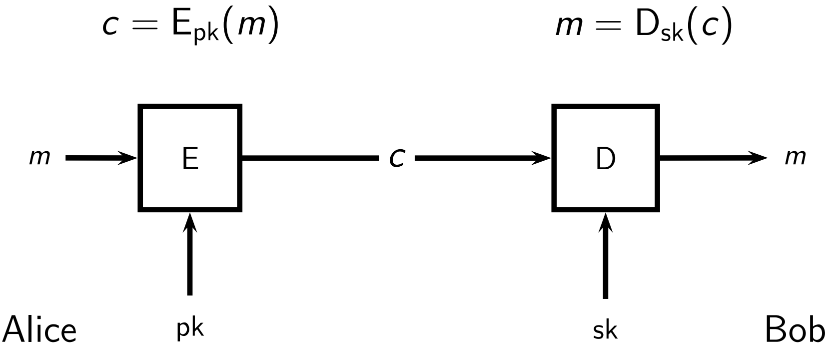


History of Public-Key Cryptography

Public-key cryptography was discovered:

- By Ellis, Cocks, and Williamson at the Government Communications Headquarters (GCHQ) in the UK in the early 1970s (not public until 1997).
- Independently by Merkle in 1974 (Merkle's puzzles).
- Independently in its discrete-logarithm based form by Diffie and Hellman in 1977, and instantiated in 1978 (key-exchange).
- Independently in its factoring-based form by Rivest, Shamir and Adleman in 1977.

Public-Key Cryptosystem



Public-Key Cryptography

Definition. A public-key cryptosystem is a tuple $(\mathsf{Gen}, \mathsf{E}, \mathsf{D})$ where,

- Gen is a **probabilistic key generation algorithm** that outputs key pairs $(\mathsf{pk}, \mathsf{sk})$,
- E is a (possibly probabilistic) **encryption algorithm** that given a public key pk and a message m in the plaintext space $\mathcal{M}_{\mathsf{pk}}$ outputs a ciphertext c, and
- D is a **decryption algorithm** that given a secret key sk and a ciphertext c outputs a plaintext m,

such that $\mathsf{D}_{\mathsf{sk}}(\mathsf{E}_{\mathsf{pk}}(m)) = m$ for every $(\mathsf{pk}, \mathsf{sk})$ and $m \in \mathcal{M}_{\mathsf{pk}}$.

RSA

The RSA Cryptosystem (1/2)

Key Generation.

- Choose $n/2$-bit primes p and q randomly and define $N = pq$.
- Choose e in $\mathbb{Z}^*_{\phi(N)}$ and compute $d = e^{-1} \bmod \phi(N)$.
- Output the key pair $((N, e), (p, q, d))$, where (N, e) is the public key and (p, q, d) is the secret key.

The RSA Cryptosystem (2/2)

Encryption. Encrypt a plaintext $m \in \mathbb{Z}_N^*$ by computing

$$c = m^e \bmod N \ .$$

Decryption. Decrypt a ciphertext c by computing

$$m = c^d \bmod N \ .$$

Why Does It Work?

$$(m^e \bmod N)^d \bmod N = m^{ed} \bmod N$$

Why Does It Work?

$$\begin{aligned}(m^e \bmod N)^d \bmod N &= m^{ed} \bmod N \\ &= m^{1+t\phi(N)} \bmod N\end{aligned}$$

Why Does It Work?

$$
\begin{aligned}
(m^e \bmod N)^d \bmod N &= m^{ed} \bmod N \\
&= m^{1+t\phi(N)} \bmod N \\
&= m^1 \cdot \left(m^{\phi(N)}\right)^t \bmod N
\end{aligned}
$$

Why Does It Work?

$$
\begin{aligned}
(m^e \bmod N)^d \bmod N &= m^{ed} \bmod N \\
&= m^{1+t\phi(N)} \bmod N \\
&= m^1 \cdot \left(m^{\phi(N)}\right)^t \bmod N \\
&= m \cdot 1^t \bmod N
\end{aligned}
$$

Why Does It Work?

$$
\begin{aligned}
(m^e \bmod N)^d \bmod N &= m^{ed} \bmod N \\
&= m^{1+t\phi(N)} \bmod N \\
&= m^1 \cdot \left(m^{\phi(N)}\right)^t \bmod N \\
&= m \cdot 1^t \bmod N \\
&= m \bmod N
\end{aligned}
$$

Implementing RSA

- Modular arithmetic.
- Greatest common divisor.
- Primality test.

Modular Arithmetic (1/3)

Basic operations on $O(n)$-bit integers using “text book” implementations.

Operation	Running time
Addition	$O(n)$
Subtraction	$O(n)$
Multiplication	$O(n^2)$
Modular reduction	$O(n^2)$
Greatest common divisor	$O(n^2)$

Modular Arithmetic (1/3)

Basic operations on $O(n)$-bit integers using "text book" implementations.

Operation	Running time
Addition	$O(n)$
Subtraction	$O(n)$
Multiplication	$O(n^2)$
Modular reduction	$O(n^2)$
Greatest common divisor	$O(n^2)$

Optimal algorithms for multiplication and modular reduction are much faster.

Modular Arithmetic (1/3)

Basic operations on $O(n)$-bit integers using "text book" implementations.

Operation	Running time
Addition	$O(n)$
Subtraction	$O(n)$
Multiplication	$O(n^2)$
Modular reduction	$O(n^2)$
Greatest common divisor	$O(n^2)$

Optimal algorithms for multiplication and modular reduction are much faster.

What about modular exponentiation?

Modular Arithmetic (2/3)

Square-and-Multiply.

$SquareAndMultiply(x, e, N)$

```
z ← 1
i =index of most significant one
while i ≥ 0
      do
          z ← z · z mod N
          if e_i = 1
             then z ← z · x mod N
          i ← i − 1
return z
```

Although basically is the same, the most efficient algorithms for exponentiation are faster.

Computing $g^{x_1}, \ldots, g^{x_k}$ can be done much faster!

Computing $\prod_{i \in [k]} g^{x_i}$ can be done much faster!

Computing $g_1^x, \ldots, g_k^x$ can be done somewhat faster!

Modular Arithmetic (3/3)

Although basically is the same, the most efficient algorithms for exponentiation are faster.

Computing $g^{x_1}, \ldots, g^{x_k}$ can be done much faster!

Computing $\prod_{i \in [k]} g^{x_i}$ can be done much faster!

Computing $g_1^x, \ldots, g_k^x$ can be done somewhat faster!

What about side-channel attacks?

Prime Number Theorem

The primes are relatively dense.

Prime Number Theorem

The primes are relatively dense.

Theorem. Let $\pi(m)$ denote the number of primes $0 < p \leq m$. Then

$$\lim_{m \to \infty} \frac{\pi(m)}{\frac{m}{\ln m}} = 1 \ .$$

Prime Number Theorem

The primes are relatively dense.

Theorem. Let $\pi(m)$ denote the number of primes $0 < p \leq m$. Then

$$\lim_{m\to\infty} \frac{\pi(m)}{\frac{m}{\ln m}} = 1 \ .$$

To generate a random prime, we repeatedly pick a random integer m and check if it is prime. It should be prime with probability close to $1/\ln m$ in a sufficently large interval.

Legendre Symbol (1/2)

Definition. Given an odd integer $b \geq 3$, an integer a is called a **quadratic residue** modulo b if there exists an integer x such that $a = x^2 \bmod b$.

Definition. The **Legendre Symbol** of an integer a modulo an **odd prime** p is defined by

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{if } a = 0 \\ 1 & \text{if } a \text{ is a quadratic residue modulo } p \\ -1 & \text{if } a \text{ is a quadratic non-residue modulo } p \end{cases} .$$

Legendre Symbol (2/2)

Theorem. If p is an odd prime, then

$$\left(\frac{a}{p}\right) = a^{(p-1)/2} \bmod p \ .$$

Legendre Symbol (2/2)

Theorem. If p is an odd prime, then

$$\left(\frac{a}{p}\right) = a^{(p-1)/2} \bmod p \enspace .$$

Proof.

- If $a = y^2 \bmod p$, then $a^{(p-1)/2} = y^{p-1} = 1 \bmod p$.

Legendre Symbol (2/2)

Theorem. If p is an odd prime, then

$$\left(\frac{a}{p}\right) = a^{(p-1)/2} \bmod p \ .$$

Proof.

- If $a = y^2 \bmod p$, then $a^{(p-1)/2} = y^{p-1} = 1 \bmod p$.
- If $a^{(p-1)/2} = 1 \bmod p$ and b generates $\mathbb{Z}_p^*$, then $a^{(p-1)/2} = b^{x(p-1)/2} = 1 \bmod p$ for some x. Since b is a generator, $(p-1) \mid x(p-1)/2$ and x must be even.

Theorem. If p is an odd prime, then

$$\left(\frac{a}{p}\right) = a^{(p-1)/2} \bmod p \ .$$

Proof.

- If $a = y^2 \bmod p$, then $a^{(p-1)/2} = y^{p-1} = 1 \bmod p$.
- If $a^{(p-1)/2} = 1 \bmod p$ and b generates $\mathbb{Z}_p^*$, then $a^{(p-1)/2} = b^{x(p-1)/2} = 1 \bmod p$ for some x. Since b is a generator, $(p-1) \mid x(p-1)/2$ and x must be even.
- If a is a non-residue, then $a^{(p-1)/2} \neq 1 \bmod p$, but $\left(a^{(p-1)/2}\right)^2 = 1 \bmod p$, so $a^{(p-1)/2} = -1 \bmod p$.

Jacobi Symbol (my academic great^9 grand father)

Definition. The **Jacobi Symbol** of an integer a modulo an odd integer $b = \prod_i p_i^{e_i}$, with p_i prime, is defined by

$$\left(\frac{a}{b}\right) = \prod_i \left(\frac{a}{p_i}\right)^{e_i} \quad .$$

Note that we can have $\left(\frac{a}{b}\right) = 1$ even when a is a non-residue modulo b.