Encryption III

601.442/642 Modern Cryptography

26th February 2026

Announcement

- Midterm grades released.

Announcement

- Midterm grades released.
- Homework 4 due **today**.

Announcement

- Midterm grades released.
- Homework 4 due **today**.
- Homework 5 will be out today and due next Thursday (5th March).

Recap: History of Public-Key Encryption

The main question with **secret-key encryption**: how to agree on a secret key?

Recap: History of Public-Key Encryption

The main question with **secret-key encryption**: how to agree on a secret key?

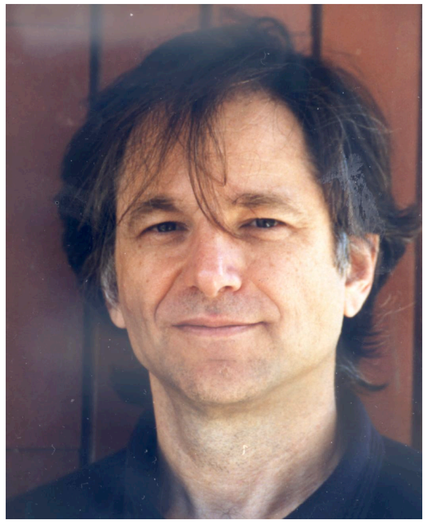
Recap: History of Public-Key Encryption

The main question with **secret-key encryption**: how to agree on a secret key?

- 1976: Whitfield Diffie and Martin Hellman proposed a key-exchange protocol

Recap: History of Public-Key Encryption

- 1976: Whitfield Diffie and Martin Hellman proposed a key-exchange protocol.
- 1977: Ronald Rivest, Adi Shamir, and Leonard Adleman proposed a public-key encryption scheme.
  - They then founded a company called RSA Security.



Recap: History of Public-Key Encryption

- 1976: Whitfield Diffie and Martin Hellman proposed a key-exchange protocol.
- 1977: Ronald Rivest, Adi Shamir, and Leonard Adleman proposed a public-key encryption scheme.
  - They then founded a company called RSA Security.
- 1982: Shafi Goldwasser and Silvio Micali defined and constructed computationally secure PKE.
  - Definition, construction and security proof.

Recap: History of Public-Key Encryption

- 1976: Whitfield Diffie and Martin Hellman proposed a key-exchange protocol.
- 1977: Ronald Rivest, Adi Shamir, and Leonard Adleman proposed a public-key encryption scheme.
  - They then founded a company called RSA Security.
- 1982: Shafi Goldwasser and Silvio Micali defined and constructed computationally secure PKE.
  - Definition, construction and security proof.
- 1985: Taher ElGamal proposed a PKE based on Diffie-Hellman key exchange.

Recap: History of Public-Key Encryption

- 1976: Whitfield Diffie and Martin Hellman proposed a key-exchange protocol.
- 1977: Ronald Rivest, Adi Shamir, and Leonard Adleman proposed a public-key encryption scheme.
  - They then founded a company called RSA Security.
- 1982: Shafi Goldwasser and Silvio Micali defined and constructed computationally secure PKE.
  - Definition, construction and security proof.
- 1985: Taher ElGamal proposed a PKE based on Diffie-Hellman key exchange.
- 1997: GCHQ revealed that it had independently developed public-key cryptography between 1969-1974.
  - Public-key encryption, RSA and DH-like key exchange discovered by James Ellis, Clifford Cocks, and Malcolm Williamson.

Recap: ElGamal Encryption

ElGamal Encryption

- KeyGen(1^λ):
  - $(\mathbb{G}, g, p) \leftarrow \mathsf{GrpGen}(1^\lambda)$
  - $\mathsf{sk} \xleftarrow{\$} \{0,\ldots,p-1\}$
  - $\mathsf{pk} = g^{\mathsf{sk}}$
- Enc(pk, m):
  - $r \xleftarrow{\$} \{0,1\}^\lambda$
  - $\mathsf{ct} := (m \cdot \mathsf{pk}^r,\ g^r)$
- Dec($\mathsf{sk}, (c_1, c_2)$):
  - $m := c_2^{-\mathsf{sk}} \cdot c_1$

Recap: ElGamal Encryption

ElGamal Encryption

- KeyGen(1^λ):
  - $(\mathbb{G}, g, p) \leftarrow \mathsf{GrpGen}(1^\lambda)$
  - $\mathsf{sk} \xleftarrow{\$} \{0,\ldots,p-1\}$
  - $\mathsf{pk} = g^{\mathsf{sk}}$
- Enc(pk, m):
  - $r \xleftarrow{\$} \{0,1\}^\lambda$
  - $\mathsf{ct} := (m \cdot \mathsf{pk}^r,\ g^r)$
- Dec($\mathsf{sk}, (c_1, c_2)$):
  - $m := c_2^{-\mathsf{sk}} \cdot c_1$

Intuition:

Recap: ElGamal Encryption

ElGamal Encryption

- KeyGen(1^λ):
  - $(\mathbb{G}, g, p) \leftarrow \text{GrpGen}(1^\lambda)$
  - $\text{sk} \xleftarrow{\$} \{0,\ldots,p-1\}$
  - $\text{pk} = g^{\text{sk}}$
- Enc(pk, m):
  - $r \xleftarrow{\$} \{0,1\}^\lambda$
  - $\text{ct} := (m \cdot \text{pk}^r, g^r)$
- Dec(sk, (c_1, c_2)):
  - $m := c_2^{-\text{sk}} \cdot c_1$

Intuition:

Encryption: Given $\text{pk} = g^{\text{sk}}$ and r, easy to compute $g^{\text{sk} \cdot r}$

Recap: ElGamal Encryption

ElGamal Encryption

- KeyGen(1^λ):
  - $(\mathbb{G}, g, p) \leftarrow \text{GrpGen}(1^\lambda)$
  - $\text{sk} \xleftarrow{\$} \{0,\ldots,p-1\}$
  - $\text{pk} = g^{\text{sk}}$
- Enc(pk, m):
  - $r \xleftarrow{\$} \{0,1\}^\lambda$
  - $\text{ct} := (m \cdot \text{pk}^r, g^r)$
- Dec(sk, (c_1, c_2)):
  - $m := c_2^{-\text{sk}} \cdot c_1$

Intuition:

Encryption: Given $\text{pk} = g^{\text{sk}}$ and r, easy to compute $g^{\text{sk} \cdot r}$

Decryption: Given $c_2 = g^r$ and sk, easy to compute $g^{r \cdot -\text{sk}}$

Recap: ElGamal Encryption

ElGamal Encryption

- KeyGen(1^λ):
  - $(\mathbb{G}, g, p) \leftarrow \text{GrpGen}(1^\lambda)$
  - $\text{sk} \xleftarrow{\$} \{0,\ldots,p-1\}$
  - $\text{pk} = g^{\text{sk}}$
- Enc(pk, m):
  - $r \xleftarrow{\$} \{0,1\}^\lambda$
  - $\text{ct} := (m \cdot \text{pk}^r, g^r)$
- Dec(sk, (c_1, c_2)):
  - $m := c_2^{-\text{sk}} \cdot c_1$

Intuition:

Encryption: Given $\text{pk} = g^{\text{sk}}$ and r, easy to compute $g^{\text{sk} \cdot r}$

Decryption: Given $c_2 = g^r$ and sk, easy to compute $g^{r \cdot -\text{sk}}$

Exponentiation in the group is efficient.

Recap: ElGamal Encryption

ElGamal Encryption

- KeyGen(1^λ):
  - $(\mathbb{G}, g, p) \leftarrow \mathsf{GrpGen}(1^\lambda)$
  - $\mathsf{sk} \xleftarrow{\$} \{0,\ldots,p-1\}$
  - $\mathsf{pk} = g^{\mathsf{sk}}$
- Enc(pk, m):
  - $r \xleftarrow{\$} \{0,1\}^\lambda$
  - $\mathsf{ct} := (m \cdot \mathsf{pk}^r, g^r)$
- Dec($\mathsf{sk}, (c_1, c_2)$):
  - $m := c_2^{-\mathsf{sk}} \cdot c_1$

Intuition:

Encryption: Given $\mathsf{pk} = g^{\mathsf{sk}}$ and r, easy to compute $g^{\mathsf{sk} \cdot r}$

Decryption: Given $c_2 = g^r$ and sk, easy to compute $g^{r \cdot -\mathsf{sk}}$

Security: Given g^{sk} and g^r, hard to compute $g^{r \cdot -\mathsf{sk}}$

Exponentiation in the group is efficient.

Recap: ElGamal Encryption

ElGamal Encryption

- KeyGen(1^λ):
  - $(\mathbb{G}, g, p) \leftarrow \mathsf{GrpGen}(1^\lambda)$
  - $\mathsf{sk} \xleftarrow{\$} \{0,\ldots,p-1\}$
  - $\mathsf{pk} = g^{\mathsf{sk}}$
- Enc(pk, m):
  - $r \xleftarrow{\$} \{0,1\}^\lambda$
  - $\mathsf{ct} := (m \cdot \mathsf{pk}^r, g^r)$
- Dec($\mathsf{sk}, (c_1, c_2)$):
  - $m := c_2^{-\mathsf{sk}} \cdot c_1$

Intuition:

Encryption: Given $\mathsf{pk} = g^{\mathsf{sk}}$ and r, easy to compute $g^{\mathsf{sk} \cdot r}$

Decryption: Given $c_2 = g^r$ and sk, easy to compute $g^{r \cdot -\mathsf{sk}}$

Exponentiation in the group is efficient.

Security: Given g^{sk} and g^r, hard to compute $g^{r \cdot -\mathsf{sk}}$

Intuitively, security holds due to hardness of discrete log.
Formally, we require the stronger DDH assumption.

Recap: ElGamal Encryption

ElGamal Encryption

- KeyGen(1^λ):
  - $(\mathbb{G}, g, p) \leftarrow \mathsf{GrpGen}(1^\lambda)$
  - $\mathsf{sk} \xleftarrow{\$} \{0,\ldots,p-1\}$
  - $\mathsf{pk} = g^{\mathsf{sk}}$
- Enc(pk, m):
  - $r \xleftarrow{\$} \{0,1\}^\lambda$
  - $\mathsf{ct} := (m \cdot \mathsf{pk}^r, g^r)$
- Dec($\mathsf{sk}, (c_1, c_2)$):
  - $m := c_2^{-\mathsf{sk}} \cdot c_1$

Intuition:

Encryption: Given $\mathsf{pk} = g^{\mathsf{sk}}$ and r, easy to compute $g^{\mathsf{sk} \cdot r}$

Decryption: Given $c_2 = g^r$ and sk, easy to compute $g^{r \cdot -\mathsf{sk}}$

Exponentiation in the group is efficient.

Security: Given g^{sk} and g^r, hard to compute $g^{r \cdot -\mathsf{sk}}$

Public-key encryption relies on computational problems that are **easy to compute** but **hard to reverse without** secret information.

RSA Public-key Encryption

- (Toy) RSA Assumption

RSA Public-key Encryption

- (Toy) RSA Assumption
  - Let p and q be λ-bit distinct primes and let $N = pq$.

RSA Public-key Encryption

- (Toy) RSA Assumption
  - Let p and q be λ-bit distinct primes and let $N = pq$.
  - Let $x \xleftarrow{\$} \{1, \ldots, N-1\}$.

RSA Public-key Encryption

- (Toy) RSA Assumption
  - Let p and q be λ-bit distinct primes and let $N = pq$.
  - Let $x \xleftarrow{\$} \{1, \ldots, N-1\}$.
  - For any efficient adversary A,

$$\Pr[A(N, x^3 \bmod N) = x] \leq \mathsf{negl}(\lambda).$$

  - In other words, A can't compute the cube-root of y modulo N, where $y = x^3$.

RSA Public-key Encryption

- (Toy) RSA Assumption
  - Let p and q be λ-bit distinct primes and let $N = pq$.
  - Let $x \xleftarrow{\$} \{1, \ldots, N-1\}$.
  - For any efficient adversary A,

$$\Pr[A(N, x^3 \bmod N) = x] \leq \mathsf{negl}(\lambda).$$

  - In other words, A can't compute the cube-root of y modulo N, where $y = x^3$.
- Seems suitable for public-key encryption: easy to compute but hard to invert without secret information.

RSA Public-key Encryption

- (Toy) RSA Assumption
  - Let p and q be λ-bit distinct primes and let $N = pq$.
  - Let $x \xleftarrow{\$} \{1, \ldots, N-1\}$.
  - For any efficient adversary A,

$$\Pr[A(N, x^3 \bmod N) = x] \leq \mathsf{negl}(\lambda).$$

  - In other words, A can't compute the cube-root of y modulo N, where $y = x^3$.
- Seems suitable for public-key encryption: easy to compute but hard to invert without secret information.
  - **Easy to compute:** Exponentiation modulo N takes $\mathsf{poly}(\log N) = \mathsf{poly}(\lambda)$ time.

RSA Public-key Encryption

- (Toy) RSA Assumption
  - Let p and q be λ-bit distinct primes and let $N = pq$.
  - Let $x \xleftarrow{\$} \{1, \ldots, N-1\}$.
  - For any efficient adversary A,

$$\Pr[A(N, x^3 \bmod N) = x] \leq \mathsf{negl}(\lambda).$$

  - In other words, A can't compute the cube-root of y modulo N, where $y = x^3$.
- Seems suitable for public-key encryption: easy to compute but hard to invert without secret information.
  - **Easy to compute:** Exponentiation modulo N takes $\mathsf{poly}(\log N) = \mathsf{poly}(\lambda)$ time.
  - **Hard to Invert:** Computing x is hard from the RSA assumption.

RSA Public-key Encryption

- (Toy) RSA Assumption
  - Let p and q be λ-bit distinct primes and let $N = pq$.
  - Let $x \xleftarrow{\$} \{1, \ldots, N-1\}$.
  - For any efficient adversary A,

$$\Pr[A(N, x^3 \bmod N) = x] \leq \mathsf{negl}(\lambda).$$

  - In other words, A can't compute the cube-root of y modulo N, where $y = x^3$.
- Seems suitable for public-key encryption: easy to compute but hard to invert without secret information.
  - **Easy to compute:** Exponentiation modulo N takes $\mathsf{poly}(\log N) = \mathsf{poly}(\lambda)$ time.
  - **Hard to Invert:** Computing x is hard from the RSA assumption.
    - Why do we believe the assumption is true?

RSA Public-key Encryption

- (Toy) RSA Assumption
  - Let p and q be λ-bit distinct primes and let $N = pq$.
  - Let $x \xleftarrow{\$} \{1, \ldots, N-1\}$.
  - For any efficient adversary A,

$$\Pr[A(N, x^3 \bmod N) = x] \leq \mathsf{negl}(\lambda).$$

  - In other words, A can't compute the cube-root of y modulo N, where $y = x^3$.
- Seems suitable for public-key encryption: easy to compute but hard to invert without secret information.
  - **Easy to compute:** Exponentiation modulo N takes $\mathsf{poly}(\log N) = \mathsf{poly}(\lambda)$ time.
  - **Hard to Invert:** Computing x is hard from the RSA assumption.
    - Why do we believe the assumption is true? TL;DR: Factoring is hard!

RSA Public-key Encryption

- (Toy) RSA Assumption
  - Let p and q be λ-bit distinct primes and let $N = pq$.
  - Let $x \xleftarrow{\$} \{1, \ldots, N-1\}$.
  - For any efficient adversary A,

$$\Pr[A(N, x^3 \bmod N) = x] \leq \mathsf{negl}(\lambda).$$

  - In other words, A can't compute the cube-root of y modulo N, where $y = x^3$.
- Seems suitable for public-key encryption: easy to compute but hard to invert without secret information.
  - **Easy to compute:** Exponentiation modulo N takes $\mathsf{poly}(\log N) = \mathsf{poly}(\lambda)$ time.
  - **Hard to Invert:** Computing x is hard from the RSA assumption.
    - Why do we believe the assumption is true? TL;DR: Factoring is hard!
  - How to invert with secret information i.e., (p, q)?

RSA Public-key Encryption

- (Toy) RSA Assumption
  - Let p and q be λ-bit distinct primes and let $N = pq$.
  - Let $x \xleftarrow{\$} \{1, \ldots, N-1\}$.

    This is not accurate, we will fix it later.
  - For any efficient adversary A,

$$\Pr[A(N, x^3 \bmod N) = x] \leq \mathsf{negl}(\lambda).$$

  - In other words, A can't compute the cube-root of y modulo N, where $y = x^3$.
- Seems suitable for public-key encryption: easy to compute but hard to invert without secret information.
  - **Easy to compute:** Exponentiation modulo N takes $\mathsf{poly}(\log N) = \mathsf{poly}(\lambda)$ time.
  - **Hard to Invert:** Computing x is hard from the RSA assumption.
    - Why do we believe the assumption is true? TL;DR: Factoring is hard!
  - How to invert with secret information i.e., (p, q)?

Arithmetic over $\mathbb{Z}_N$

- $\mathbb{Z}_N$: Addition and multiplication over $\mathbb{Z}$, reduced modulo N.

Arithmetic over $\mathbb{Z}_N$

- $\mathbb{Z}_N$: Addition and multiplication over $\mathbb{Z}$, reduced modulo N.
- $\mathbb{Z}_N$ is a **commutative ring**

Arithmetic over $\mathbb{Z}_N$

- $\mathbb{Z}_N$: Addition and multiplication over $\mathbb{Z}$, reduced modulo N.
- $\mathbb{Z}_N$ is a **commutative ring**
  - Addition and multiplication are commutative and associative.

Arithmetic over $\mathbb{Z}_N$

- $\mathbb{Z}_N$: Addition and multiplication over $\mathbb{Z}$, reduced modulo N.
- $\mathbb{Z}_N$ is a **commutative ring**
  - Addition and multiplication are commutative and associative.
  - Multiplication distributes over addition.

Arithmetic over $\mathbb{Z}_N$

- $\mathbb{Z}_N$: Addition and multiplication over $\mathbb{Z}$, reduced modulo N.
- $\mathbb{Z}_N$ is a **commutative ring**
  - Addition and multiplication are commutative and associative.
  - Multiplication distributes over addition.
  - Additive identity is 0. Every element $a \in \mathbb{Z}_N$ has additive inverse $-a \in \mathbb{Z}_N$ such that $a + (-a) \equiv 0 \pmod{N}$.

Arithmetic over $\mathbb{Z}_N$

- $\mathbb{Z}_N$: Addition and multiplication over $\mathbb{Z}$, reduced modulo N.
- $\mathbb{Z}_N$ is a **commutative ring**
  - Addition and multiplication are commutative and associative.
  - Multiplication distributes over addition.
  - Additive identity is 0. Every element $a \in \mathbb{Z}_N$ has additive inverse $-a \in \mathbb{Z}_N$ such that $a + (-a) \equiv 0 \pmod{N}$.
  - Multiplicative identity is 1.

Arithmetic over $\mathbb{Z}_N$

- $\mathbb{Z}_N$: Addition and multiplication over $\mathbb{Z}$, reduced modulo N.
- $\mathbb{Z}_N$ is a **commutative ring**
  - Addition and multiplication are commutative and associative.
  - Multiplication distributes over addition.
  - Additive identity is 0. Every element $a \in \mathbb{Z}_N$ has additive inverse $-a \in \mathbb{Z}_N$ such that $a + (-a) \equiv 0 \pmod{N}$.
  - Multiplicative identity is 1.
- Computation over $\mathbb{Z}_N$ is **efficient**: Addition, subtraction, multiplication (and even exponentiation) takes $\mathsf{poly}(\log N) = \mathsf{poly}(\lambda)$ time.

Solving Equations over $\mathbb{Z}_N$

Goal: Understand hardness of computing cube-roots modulo N.

How hard is it to solve equations in $\mathbb{Z}_N$?

Solving Equations over $\mathbb{Z}_N$: Linear Equations

- Given $a, b,$ solve for x such that

$$a \cdot x \equiv b \pmod{N}.$$

Solving Equations over $\mathbb{Z}_N$: Linear Equations

- Given a, b, solve for x such that

$$a \cdot x \equiv b \pmod{N}.$$

- **Fact:** Solution exists *if and only if* $\gcd(a, N)$ divides b.

Solving Equations over $\mathbb{Z}_N$: Linear Equations

- Given a, b, solve for x such that

$$a \cdot x \equiv b \pmod{N}.$$

- **Fact:** Solution exists *if and only if* $\gcd(a, N)$ divides b.
- **Multiplicative inverse:** When $\gcd(a, N) = 1$, then there exists a solution $x = a^{-1}$ such that

$$a \cdot a^{-1} \equiv 1 \pmod{N}.$$

  - a^{-1} is called the multiplicative inverse of a (and vice versa).

Solving Equations over $\mathbb{Z}_N$: Linear Equations

- Given a, b, solve for x such that

$$a \cdot x \equiv b \pmod{N}.$$

- **Fact:** Solution exists *if and only if* $\gcd(a, N)$ divides b.
- **Multiplicative inverse:** When $\gcd(a, N) = 1$, then there exists a solution $x = a^{-1}$ such that

$$a \cdot a^{-1} \equiv 1 \pmod{N}.$$

  - a^{-1} is called the multiplicative inverse of a (and vice versa).
  - The Extended Euclidean algorithm computes a^{-1} in $\text{poly}(\lambda)$ time.

Solving Equations over $\mathbb{Z}_N$: Linear Equations

- Given a, b, solve for x such that

$$a \cdot x \equiv b \pmod{N}.$$

- **Fact:** Solution exists *if and only if* $\gcd(a, N)$ divides b.
- **Multiplicative inverse:** When $\gcd(a, N) = 1$, then there exists a solution $x = a^{-1}$ such that

$$a \cdot a^{-1} \equiv 1 \pmod{N}.$$

  - a^{-1} is called the multiplicative inverse of a (and vice versa).
  - The Extended Euclidean algorithm computes a^{-1} in $\text{poly}(\lambda)$ time.
- $\mathbb{Z}_N^\times = \{a \in \mathbb{Z}_N : \gcd(a, N) = 1\}$ forms a **group** under multiplication modulo N.
  - Identity is 1.
  - Each element has (multiplicative) inverse.

Solving Equations over $\mathbb{Z}_N$: Quadratic Equations

- Given $a, b, c,$ solve for x such that

$$ax^2 + bx \equiv c \pmod{N}.$$

Solving Equations over $\mathbb{Z}_N$: Quadratic Equations

- Given $a, b, c,$ solve for x such that

$$ax^2 + bx \equiv c \pmod{N}.$$

- **Simplification:** Given a' solve for x such that

$$x^2 \equiv a' \pmod{N}.$$

Square Root modulo N

- Given a solve for x such that

$$x^2 \equiv a \pmod{N}.$$

Square Root modulo N

- Given a solve for x such that

$$x^2 \equiv a \pmod{N}.$$

- **Case 1:** Easy if N is prime.

Square Root modulo N

- Given a solve for x such that

$$x^2 \equiv a \pmod{N}.$$

- **Case 1:** Easy if N is prime.
  - Tonelli-Shanks algorithm can compute square-root modulo a prime N in $\mathsf{poly}(\log N)$ time.

Square Root modulo N

- Given a solve for x such that

$$x^2 \equiv a \pmod{N}.$$

- **Case 1:** Easy if N is prime.

Square Root modulo N

- Given a solve for x such that

$$x^2 \equiv a \pmod{N}.$$

- **Case 1:** Easy if N is prime.
- **Case 2:** Easy when factoring N is easy.

Square Root modulo N

- Given a solve for x such that

$$x^2 \equiv a \pmod{N}.$$

- **Case 1:** Easy if N is prime.
- **Case 2:** Easy when factoring N is easy.
  - Example: Let $N = pq$, where p and q are distinct primes.

Square Root modulo N

- Given a solve for x such that

$$x^2 \equiv a \pmod{N}.$$

- **Case 1:** Easy if N is prime.
- **Case 2:** Easy when factoring N is easy.
  - Example: Let $N = pq$, where p and q are distinct primes.
    - **Chinese Remainder Theorem:** The map $x \mapsto (x \bmod p, x \bmod q)$ is an efficiently invertible bijection from $\mathbb{Z}_N \to \mathbb{Z}_p \times \mathbb{Z}_q$.

Square Root modulo N

- Given a solve for x such that

$$x^2 \equiv a \pmod{N}.$$

- **Case 1:** Easy if N is prime.
- **Case 2:** Easy when factoring N is easy.
  - Example: Let $N = pq$, where p and q are distinct primes.
    - **Chinese Remainder Theorem:** The map $x \mapsto (x \bmod p, x \bmod q)$ is an efficiently invertible bijection from $\mathbb{Z}_N \to \mathbb{Z}_p \times \mathbb{Z}_q$.
    - Find solutions for $x^2 \equiv a \pmod{p}$ and $x^2 \equiv a \pmod{q}$ and then compute solutions modulo N using CRT.

Square Root modulo N

- Given a solve for x such that

$$x^2 \equiv a \pmod{N}.$$

- **Case 1:** Easy if N is prime.
- **Case 2:** Easy when factoring N is easy.
  - Example: Let $N = pq$, where p and q are distinct primes.
    - **Chinese Remainder Theorem:** The map $x \mapsto (x \bmod p, x \bmod q)$ is an efficiently invertible bijection from $\mathbb{Z}_N \to \mathbb{Z}_p \times \mathbb{Z}_q$.
    - Find solutions for $x^2 \equiv a \pmod{p}$ and $x^2 \equiv a \pmod{q}$ and then compute solutions modulo N using CRT.
    - Up to 4 solutions modulo N.

Square Root modulo N

- Given a solve for x such that

$$x^2 \equiv a \pmod{N}.$$

- **Case 1:** Easy if N is prime.
- **Case 2:** Easy when factoring N is easy.

Square Root modulo N

- Given a solve for x such that

$$x^2 \equiv a \pmod{N}.$$

- **Case 1:** Easy if N is prime.
- **Case 2:** Easy when factoring N is easy.
- **Case 3:** Hard when factoring N is hard.

Square Root modulo N

- Given a solve for x such that

$$x^2 \equiv a \pmod{N}.$$

- **Case 1:** Easy if N is prime.
- **Case 2:** Easy when factoring N is easy.
- **Case 3:** Hard when factoring N is hard.
  - When N is a product of two distinct primes, finding square-roots is as hard as factoring!

Square Root modulo N

- Given a solve for x such that

$$x^2 \equiv a \pmod{N}.$$

- **Case 1:** Easy if N is prime.
- **Case 2:** Easy when factoring N is easy.
- **Case 3:** Hard when factoring N is hard.
  - When N is a product of two distinct primes, finding square-roots is as hard as factoring!
    - We can show that if an efficient algorithm can find square-root of a modulo N (even for just a uniformly random $a \in \mathbb{Z}_N$) then there exists an efficient algorithm for factoring N.

Square Root modulo N

- Given a solve for x such that

$$x^2 \equiv a \pmod{N}.$$

- **Case 1:** Easy if N is prime.
- **Case 2:** Easy when factoring N is easy.
- **Case 3:** Hard when factoring N is hard.
  - When N is a product of two distinct primes, finding square-roots is as hard as factoring!
    - We can show that if an efficient algorithm can find square-root of a modulo N (even for just a uniformly random $a \in \mathbb{Z}_N$) then there exists an efficient algorithm for factoring N.
- When N is a product of distinct odd primes, computing square roots modulo N is equivalent to factoring N.

Cube Root modulo N

- Given $a \in \mathbb{Z}_N^\times$ solve for x such that

$$x^3 \equiv a \pmod{N}.$$

Cube Root modulo N

- Given $a \in \mathbb{Z}_N^\times$ solve for x such that

$$x^3 \equiv a \pmod{N}.$$

- **Euler's Theorem:** For all $a \in \mathbb{Z}_N^\times$

$$a^{\varphi(N)} \equiv 1 \pmod{N}$$

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

Cube Root modulo N

- Given $a \in \mathbb{Z}_N^\times$ solve for x such that

$$x^3 \equiv a \pmod{N}.$$

- **Euler's Theorem:** For all $a \in \mathbb{Z}_N^\times$

$$a^{\varphi(N)} \equiv 1 \pmod{N}$$

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

  - **Consequence**

    - For any $e \in \mathbb{Z}$ and $x \in \mathbb{Z}_{N'}^\times$

$$x^e \equiv x^{e \bmod \varphi(N)} \pmod{N}.$$

Cube Root modulo N

- Given $a \in \mathbb{Z}_N^\times$ solve for x such that

$$x^3 \equiv a \pmod{N}.$$

- **Euler's Theorem:** For all $a \in \mathbb{Z}_N^\times$

$$a^{\varphi(N)} \equiv 1 \pmod{N}$$

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

  - **Consequence**

    - For any $e \in \mathbb{Z}$ and $x \in \mathbb{Z}_{N'}^\times$

$$x^e \equiv x^{e \bmod \varphi(N)} \pmod{N}.$$

    - We compute modulo $\varphi(N)$ "in the exponent".

Cube Root modulo N

- Given $a \in \mathbb{Z}_N^\times$ solve for x such that

$$x^3 \equiv a \pmod{N}.$$

- From Euler's theorem, we compute modulo $\varphi(N)$ "in the exponent", where $\varphi(N) = (p-1)(q-1)$.

Cube Root modulo N

- Given $a \in \mathbb{Z}_N^\times$ solve for x such that

$$x^3 \equiv a \pmod{N}.$$

- From Euler's theorem, we compute modulo $\varphi(N)$ "in the exponent", where $\varphi(N) = (p-1)(q-1)$.
- **Case 1:** If 3 divides $\varphi(N)$ then $x \mapsto x^3$ is many-to-one over $\mathbb{Z}_N^\times$.

Cube Root modulo N

- Given $a \in \mathbb{Z}_N^\times$ solve for x such that

$$x^3 \equiv a \pmod{N}.$$

- From Euler's theorem, we compute modulo $\varphi(N)$ "in the exponent", where $\varphi(N) = (p-1)(q-1)$.
- **Case 1:** If 3 divides $\varphi(N)$ then $x \mapsto x^3$ is many-to-one over $\mathbb{Z}_N^\times$.
  - In this case, computing cube-roots is equivalent to factoring.

Cube Root modulo N

- Given $a \in \mathbb{Z}_N^\times$ solve for x such that

$$x^3 \equiv a \pmod{N}.$$

- From Euler's theorem, we compute modulo $\varphi(N)$ "in the exponent", where $\varphi(N) = (p-1)(q-1)$.
- **Case 1:** If 3 divides $\varphi(N)$ then $x \mapsto x^3$ is many-to-one over $\mathbb{Z}_N^\times$.
  - In this case, computing cube-roots is equivalent to factoring.
  - But for PKE, we want to invert and compute the cube-root x (using secret information).

Cube Root modulo N

- Given $a \in \mathbb{Z}_N^\times$ solve for x such that

$$x^3 \equiv a \pmod{N}.$$

- From Euler's theorem, we compute modulo $\varphi(N)$ "in the exponent", where $\varphi(N) = (p-1)(q-1)$.
- **Case 1:** If 3 divides $\varphi(N)$ then $x \mapsto x^3$ is many-to-one over $\mathbb{Z}_N^\times$.
  - In this case, computing cube-roots is equivalent to factoring.
  - But for PKE, we want to invert and compute the cube-root x (using secret information).
- **Case 2:** If 3 does **not** divide $\varphi(N)$ then $x \mapsto x^3$ is a bijection over $\mathbb{Z}_N^\times$.

Cube Root modulo N

- Given $a \in \mathbb{Z}_N^\times$ solve for x such that

$$x^3 \equiv a \pmod{N}.$$

- From Euler's theorem, we compute modulo $\varphi(N)$ "in the exponent", where $\varphi(N) = (p-1)(q-1)$.
- **Case 1:** If 3 divides $\varphi(N)$ then $x \mapsto x^3$ is many-to-one over $\mathbb{Z}_N^\times$.
  - In this case, computing cube-roots is equivalent to factoring.
  - But for PKE, we want to invert and compute the cube-root x (using secret information).
- **Case 2:** If 3 does **not** divide $\varphi(N)$ then $x \mapsto x^3$ is a bijection over $\mathbb{Z}_N^\times$.
    - Since $\gcd(3, \varphi(N)) = 1$, there exists d such that $3 \cdot d \equiv 1 \pmod{\varphi(N)}$.

Cube Root modulo N

- Given $a \in \mathbb{Z}_N^\times$ solve for x such that

$$x^3 \equiv a \pmod{N}.$$

- From Euler's theorem, we compute modulo $\varphi(N)$ "in the exponent", where $\varphi(N) = (p-1)(q-1)$.
- **Case 1:** If 3 divides $\varphi(N)$ then $x \mapsto x^3$ is many-to-one over $\mathbb{Z}_N^\times$.
  - In this case, computing cube-roots is equivalent to factoring.
  - But for PKE, we want to invert and compute the cube-root x (using secret information).
- **Case 2:** If 3 does **not** divide $\varphi(N)$ then $x \mapsto x^3$ is a bijection over $\mathbb{Z}_N^\times$.
  - Since $\gcd(3, \varphi(N)) = 1$, there exists d such that $3 \cdot d \equiv 1 \pmod{\varphi(N)}$.
  - Given a, a^d computes the cube-root: $a^d \equiv x^{3 \cdot d} \equiv x^{3 \cdot d \bmod \varphi(N)} \equiv x \pmod{N}$.

Cube Root modulo N

- Given $a \in \mathbb{Z}_N^\times$ solve for x such that

$$x^3 \equiv a \pmod{N}.$$

- From Euler's theorem, we compute modulo $\varphi(N)$ "in the exponent", where $\varphi(N) = (p-1)(q-1)$.
- **Case 1:** If 3 divides $\varphi(N)$ then $x \mapsto x^3$ is many-to-one over $\mathbb{Z}_N^\times$.
  - In this case, computing cube-roots is equivalent to factoring.
  - But for PKE, we want to invert and compute the cube-root x (using secret information).
- **Case 2:** If 3 does **not** divide $\varphi(N)$ then $x \mapsto x^3$ is a bijection over $\mathbb{Z}_N^\times$.
  - Since $\gcd(3, \varphi(N)) = 1$, there exists d such that $3 \cdot d \equiv 1 \pmod{\varphi(N)}$.
  - Given a, a^d computes the cube-root: $a^d \equiv x^{3 \cdot d} \equiv x^{3 \cdot d \bmod \varphi(N)} \equiv x \pmod{N}$.
  - Cube-root can be computed efficiently if we know the factorization of N.

Cube Root modulo N

- Given $a \in \mathbb{Z}_N^\times$ solve for x such that

$$x^3 \equiv a \pmod{N}.$$

- From Euler's theorem, we compute modulo $\varphi(N)$ "in the exponent", where $\varphi(N) = (p-1)(q-1)$.
- **Case 1:** If 3 divides $\varphi(N)$ then $x \mapsto x^3$ is many-to-one over $\mathbb{Z}_N^\times$.
  - In this case, computing cube-roots is equivalent to factoring.
  - But for PKE, we want to invert and compute the cube-root x (using secret information).
- **Case 2:** If 3 does **not** divide $\varphi(N)$ then $x \mapsto x^3$ is a bijection over $\mathbb{Z}_N^\times$.
  - Since $\gcd(3, \varphi(N)) = 1$, there exists d such that $3 \cdot d \equiv 1 \pmod{\varphi(N)}$.
  - Given a, a^d computes the cube-root: $a^d \equiv x^{3 \cdot d} \equiv x^{3 \cdot d \bmod \varphi(N)} \equiv x \pmod{N}$.
  - Cube-root can be computed efficiently if we know the factorization of N.

But we no longer know if computing cube-roots is as hard as factoring.

That is, finding cube-roots might be easier.

RSA Assumption

RSA Assumption

Let $\mathsf{GenRSA}(1^\lambda) \to (p, q)$ be a PPT algorithm that outputs two distinct λ-bit odd primes. Then for any non-uniform PPT adversary A

$$\Pr\left[A(N, e, x^e \bmod N) = x : \begin{array}{c}(p,q) \leftarrow \mathsf{GenRSA}(1^\lambda) \\ N = pq \\ e \xleftarrow{\$} \mathbb{Z}^\times_{\varphi(N)} \\ x \xleftarrow{\$} \mathbb{Z}^\times_N\end{array}\right] \leq \mathsf{negl}(\lambda)\,.$$

An efficient adversary cannot compute the e-th root modulo N.

Textbook RSA Encryption

Textbook RSA

Textbook RSA Encryption

Textbook RSA

- KeyGen(1^λ)
  - $(p, q) \leftarrow$ GenRSA(1^λ)
  - Let $N = pq$ and sample $e \xleftarrow{\$} \mathbb{Z}^\times_{\varphi(N)}$. Set pk $:= (N, e)$.
  - Let $d \equiv e^{-1} \pmod{\varphi(N)}$. Set sk $:= (d, N)$.

Textbook RSA Encryption

Textbook RSA

- KeyGen(1^λ)
  - $(p, q) \leftarrow$ GenRSA(1^λ)
  - Let $N = pq$ and sample $e \xleftarrow{\$} \mathbb{Z}^\times_{\varphi(N)}$. Set pk $:= (N, e)$.
  - Let $d \equiv e^{-1} \pmod{\varphi(N)}$. Set sk $:= (d, N)$.
- Enc(pk, m)
  - Output ct $:= m^e \bmod N$.

Textbook RSA Encryption

Textbook RSA

- KeyGen(1^λ)
  - $(p, q) \leftarrow \text{GenRSA}(1^\lambda)$
  - Let $N = pq$ and sample $e \xleftarrow{\$} \mathbb{Z}^\times_{\varphi(N)}$. Set pk $:= (N, e)$.
  - Let $d \equiv e^{-1} \pmod{\varphi(N)}$. Set sk $:= (d, N)$.
- Enc(pk, m)
  - Output ct $:= m^e \bmod N$.
- Dec(sk, ct)
  - Output $m :=$ ct$^d \bmod N$

Textbook RSA Encryption

Textbook RSA

All algorithms are efficient.

- KeyGen(1^λ)
  - $(p, q) \leftarrow \text{GenRSA}(1^\lambda)$
  - Let $N = pq$ and sample $e \xleftarrow{\$} \mathbb{Z}^\times_{\varphi(N)}$. Set pk $:= (N, e)$.
  - Let $d \equiv e^{-1} \pmod{\varphi(N)}$. Set sk $:= (d, N)$.
- Enc(pk, m)
  - Output ct $:= m^e \bmod N$.
- Dec(sk, ct)
  - Output $m :=$ ct$^d \bmod N$

Textbook RSA Encryption

Textbook RSA

- KeyGen(1^λ)
  - $(p, q) \leftarrow \text{GenRSA}(1^\lambda)$
  - Let $N = pq$ and sample $e \xleftarrow{\$} \mathbb{Z}^\times_{\varphi(N)}$. Set pk $:= (N, e)$.
  - Let $d \equiv e^{-1} \pmod{\varphi(N)}$. Set sk $:= (d, N)$.
- Enc(pk, m)
  - Output ct $:= m^e \bmod N$.
- Dec(sk, ct)
  - Output $m :=$ ct$^d \bmod N$

All algorithms are efficient.

Correctness: $x \mapsto x^e$ is a bijection since $e \in \mathbb{Z}^\times_{\varphi(N)}$.

Textbook RSA Encryption

Textbook RSA

- KeyGen(1^λ)
  - $(p, q) \leftarrow \text{GenRSA}(1^\lambda)$
  - Let $N = pq$ and sample $e \xleftarrow{\$} \mathbb{Z}^\times_{\varphi(N)}$. Set pk := (N, e).
  - Let $d \equiv e^{-1} \pmod{\varphi(N)}$. Set sk := (d, N).
- Enc(pk, m)
  - Output ct := $m^e \bmod N$.
- Dec(sk, ct)
  - Output m := $\text{ct}^d \bmod N$

All algorithms are efficient.

Correctness: $x \mapsto x^e$ is a bijection since $e \in \mathbb{Z}^\times_{\varphi(N)}$.

Security?

Hard to recover m given ct due to RSA assumption.

Textbook RSA Encryption

Textbook RSA

- KeyGen(1^λ)
  - $(p, q) \leftarrow \text{GenRSA}(1^\lambda)$
  - Let $N = pq$ and sample $e \xleftarrow{\$} \mathbb{Z}^\times_{\varphi(N)}$. Set pk := (N, e).
  - Let $d \equiv e^{-1} \pmod{\varphi(N)}$. Set sk := (d, N).
- Enc(pk, m)
  - Output ct := $m^e \bmod N$.
- Dec(sk, ct)
  - Output m := $\text{ct}^d \bmod N$

All algorithms are efficient.

Correctness: $x \mapsto x^e$ is a bijection since $e \in \mathbb{Z}^\times_{\varphi(N)}$.

Security?

Hard to recover m given ct due to RSA assumption.

But is it CPA-secure?

Textbook RSA Encryption

Textbook RSA

- KeyGen(1^λ)
  - $(p, q) \leftarrow \text{GenRSA}(1^\lambda)$
  - Let $N = pq$ and sample $e \xleftarrow{\$} \mathbb{Z}^\times_{\varphi(N)}$. Set pk := (N, e).
  - Let $d \equiv e^{-1} \pmod{\varphi(N)}$. Set sk := (d, N).
- Enc(pk, m)
  - Output ct := $m^e \bmod N$.
- Dec(sk, ct)
  - Output m := $\text{ct}^d \bmod N$

All algorithms are efficient.

Correctness: $x \mapsto x^e$ is a bijection since $e \in \mathbb{Z}^\times_{\varphi(N)}$.

Security?

Hard to recover m given ct due to RSA assumption.

But is it CPA-secure?

No, it is deterministic!

Textbook RSA Encryption

Textbook RSA

- KeyGen(1^λ)
  - $(p, q) \leftarrow$ GenRSA(1^λ)
  - Let $N = pq$ and sample $e \xleftarrow{\$} \mathbb{Z}^\times_{\varphi(N)}$. Set pk := (N, e).
  - Let $d \equiv e^{-1} \pmod{\varphi(N)}$. Set sk := (d, N).
- Enc(pk, m)
  - Output ct := $m^e \bmod N$.
- Dec(sk, ct)
  - Output m := ct$^d \bmod N$

All algorithms are efficient.

Correctness: $x \mapsto x^e$ is a bijection since $e \in \mathbb{Z}^\times_{\varphi(N)}$.

Security?

Hard to recover m given ct due to RSA assumption.

But is it CPA-secure?

No, it is deterministic!

Next week's HW: converting textbook RSA to CPA-secure scheme.

Hardness of Factoring

Hardness of Factoring

- Trial Division
    - Iterate x from 2 to $\sqrt{N}$ and check if x divides N.
    - Runtime is $2^{\lambda/2} \cdot \mathsf{poly}(\lambda)$ for $N \leq 2^{\lambda}$.

Hardness of Factoring

- Trial Division
    - Iterate x from 2 to $\sqrt{N}$ and check if x divides N.
    - Runtime is $2^{\lambda/2} \cdot \mathsf{poly}(\lambda)$ for $N \leq 2^{\lambda}$.
- Best known heuristic algorithm $2^{o\left(\sqrt[3]{\lambda \log^2 \lambda}\right)}$.

Hardness of Factoring

- Trial Division
  - Iterate x from 2 to $\sqrt{N}$ and check if x divides N.
  - Runtime is $2^{\lambda/2} \cdot \mathsf{poly}(\lambda)$ for $N \leq 2^{\lambda}$.
- Best known heuristic algorithm $2^{o\left(\sqrt[3]{\lambda \log^2 \lambda}\right)}$.
- Parameters for RSA
  - N is at least 2048 bits.
  - Recommended to use 3072-bit N.