Speaker's Notes: Public-key Cryptography from Factoring

Introduction - Public-key Cryptosystems (0-2:00)

[BOARD]

> Public-key cryptography (Diffie-Hellman 1976)
>
> - Public key $(pk) \rightarrow$ encryption
> - Secret key $(sk) \rightarrow$ decryption

- Start: "Today discussing factoring-based public-key crypto"
- 1976: Diffie-Hellman revolutionary idea
- Key innovation: **no shared secret needed in advance**
- Anyone can encrypt using public key
- Only secret key holder can decrypt

Definition: Public-key cryptosystem = (G, E, D)

- G: key generation (probabilistic, security parameter k, returns (sk, pk))
- E: encryption algorithm
- D: decryption algorithm
- Correctness: $\forall x \in \mathscr{P} : x = D_{sk}(E_{pk}(x))$

[BOARD]

> Trapdoor one-way function $f : X \rightarrow Y$

- Core concept: trapdoor one-way function
- Easy to compute $f(x)$ from x
- Infeasible to invert $f(x)$ to get x
- BUT: with trapdoor t_f, inversion becomes easy
- This is what public key does - $pk = f$, $sk = t_f$

[TRANSITION] "In 1977, Rivest, Shamir, Adleman answered Diffie-Hellman's challenge with RSA - first practical construction using factoring hardness."

RSA - intro (2:00-3:00)

[BOARD]

> **RSA Security**: Factoring Hardness

> - EASY: $p \times q \to n$
> - HARD: $n \to p, q$

- Security relies on **factoring problem** being computationally hard
- Multiplying two large primes: easy (polynomial time)
- Factoring product n back into p, q: extremely difficult

Why attacker can't break RSA:

- Modern standard: 2048 or 4096 bit primes
- Makes factorization infeasible with current computational power

[TRANSITION] "Let's see how RSA keys are actually generated."

RSA - Key Generation (3:00-6:00)

[BOARD]

> **RSA Key Generation** (security parameter k)
>
> 1. Choose random $k/2$-bit primes p, q
> 2. $n = p \times q$ (modulus)
> 3. Choose e: $\gcd(e, (p-1)(q-1)) = 1$
> 4. Compute $d \equiv e^{-1} \mod (p-1)(q-1)$
>
> Public key: (e, n)
>
> Private key: (d, n)

- Input: security parameter k
- Generate two random $k/2$-bit primes p, q
- Compute modulus: $n = p \times q$
- Important: $\mathscr{P} = \mathscr{C} = \mathbb{Z}_n$ always

Public exponent e:

- Must satisfy: $1 < e < (p-1)(q-1)$
- Must satisfy: $\gcd(e, (p-1)(q-1)) = 1$

Private exponent d:

- Compute: $d \equiv e^{-1} \mod (p-1)(q-1)$
- Meaning: $e \cdot d \equiv 1 \mod (p-1)(q-1)$
- Stress: p, q kept secret, destroyed after computing d
- Public key (e, n) published

- Private key (d, n) kept secret

[TRANSITION] "Once keys generated, encryption and decryption are remarkably simple operations."

RSA - Encryption & Decryption (6:00-8:00)

[BOARD]

> **Encryption:** $c = m^e \mod n$
>
> **Decryption:** $m = c^d \mod n$

Encryption:

- Input: message $m \in \mathbb{Z}_n$
- Compute: $c = m^e \mod n$
- Simple modular exponentiation

Decryption:

- Input: ciphertext c
- Compute: $m = c^d \mod n$
- Deterministic (usually)

Why it works (correctness):

[BOARD]

> $(m^e)^d = m^{ed} = m^{1+k(p-1)(q-1)} \equiv m \mod n$

- $(m^e)^d \equiv m^{ed} \mod n$
- Since $ed \equiv 1 \mod (p-1)(q-1)$
- By Euler's theorem: $m^{ed} \equiv m \mod n$ (That's why we chose $(p-1)(q-1)$)
- Original message recovered!

[TRANSITION] "Simple to describe, but why is this secure? The answer lies in factoring."

Security Intuition (8:00-10:00)

[BOARD]

> **EASY:** $p \times q \to n$
>
> **HARD:** $n \to p, q$ (factoring)

- **Core security assumption**: factoring is hard

Attack scenario:

- Attacker knows: (e, n) (public key)
- Attacker wants: to compute d
- To find d: need $e^{-1} \mod (p-1)(q-1)$
- To find $(p-1)(q-1)$: need to factor n into p, q
- **Factoring bottleneck protects everything**

[TRANSITION] "Basic RSA has weaknesses against chosen-plaintext attacks because it is deterministic on a given (sk, pk) pair. PCRSA fixes this by adding randomness."

PCRSA - CPA Secure RSA (10:00-13:00)

[BOARD]

> **PCRSA:** Probabilistic RSA
>
> Messages: $\{0, 1\}$ (single bits)
>
> Ciphertexts: $\mathbb{Z}_n^*$

- Plain RSA is deterministic $\rightarrow$ not CPA secure
- PCRSA: probabilistic variant that IS CPA secure

Key Generation:

- Same as standard RSA: generates (n, e), (n, d)

Encryption (bit b):

- Choose random $x_b \in \mathbb{Z}_n^*$ where LSB$(x_b) = b$
- Ciphertext: $c = x_b^e \mod n$
- Randomness in choice of x_b makes it probabilistic

[BOARD]

> **Encrypt bit b:**
>
> 1. Choose random $x_b \in \mathbb{Z}_n^*$ with $\text{LSB}(x_b) = b$
> 2. $c = x_b^e \mod n$

Decryption:

- Compute $x_b = c^d \mod n$
- Extract least significant bit
- That's the message bit!

Practical use:

- Not efficient for bulk encryption (1 bit at a time!)

- Main use: hybrid cryptosystems
- Example: encrypt AES key using PCRSA
- Then use AES for actual data (symmetric, fast)

[TRANSITION] "For RSA to work, we need large primes. How do we find them efficiently? Miller-Rabin test."

Miller-Rabin Primality Test (13:00-15:30)

[BOARD]

> **Prime generation:**
>
> 1. Pick random k-bit number $x \in [2^k, 2^{k+1}]$
> 2. Test if prime (Miller-Rabin)
> 3. If yes $\rightarrow$ output; else repeat

- Need to generate random k-bit primes
- Simple strategy: random x, test primality, repeat if composite
- Requires efficient primality testing

Miller-Rabin: fast but probabilistic

- May rarely accept composites (very low probability)
- Run multiple times to increase confidence

[BOARD]

> **Miller-Rabin(x):**
>
> 1. Write $x - 1 = 2^a \times b$ (b odd)
> 2. Choose random $w \in \mathbb{Z}_x$
> 3. If $\gcd(w, x) \neq 1 \rightarrow$ reject
> 4. Compute: $w^b, w^{2b}, \ldots, w^{2^{a-1}b} \mod x$
> 5. Accept if: any $= -1$ or $w^b = 1$

Algorithm steps:

1. Input: odd number x
2. Write $x - 1 = 2^a b$ where b odd
3. Choose random non-zero $w \in \mathbb{Z}_x$
4. Check $\gcd(w, x)$: if $\neq 1$, reject (found factor!)
5. Now know $w \in \mathbb{Z}_x^*$
6. Compute sequence: $w^b, w^{2b}, w^{2^2 b}, \ldots, w^{2^{a-1}b} \mod x$
7. **Accept if**: any element is $-1 \mod x$, OR $w^b \equiv 1 \mod x$
8. Otherwise reject

- Based on properties of quadratic residues
- If x prime, certain patterns must appear
- If composite, likely to fail test
- Repeat k times for 2^{-k} error probability

Conclusion (15:30-16:00)

Summary:

- RSA: first practical public-key cryptosystem
- Security from factoring hardness
- PCRSA adds CPA security via randomization
- Miller-Rabin enables efficient prime generation
- Foundation of modern public-key infrastructure

Questions?