Public-key cryptography from Factoring

What a public-key cryptosystem is. Basic spec of RSA, maybe proof that decryption works. Then some selection of the following: How to make RSA be CPA secure (the PCRSA scheme, and the result that computing the least significant bit is as hard as inverting RSA). How to generate keys and Miller-Rabin primality testing, how to get CCA security: OAEP and the intuition on why it works.

The "Elevator Pitch" (Synthesis)

> **Explain Public-key Crypto based on Factoring to a peer in 3 sentences. Focus on WHY we use it, not just how.**

Core Vocabulary & Syntax (Total Recall)

Public-key cryptosystem

> In "New Directions in Cryptography" from 1976, Diffie and Hellman proposed the idea of public-key encryption for the first time in open research. The basic idea was that one could create a matching pair of a public and a secret key, such that the public key could be used only for encryption, while secret key would be required in order to decrypt. In this way, anyone could encrypt something meant to be received by the owner of the secret key without having to agree on a shared key in advance. They proposed several ideas for realizing this, but one turned out be particularly fruitful, namely the idea of a trapdoor one-way function.

> A public-key aka asymmetric cryptosystem consists of 3 algorithms (G, E, D).

> For a public-key cryptosystem, we always require that for any key pair (pk, sk) output by G, correct decryption is possible, i.e. it holds that for any $x \in P$, $x = D_{sk}(E_{pk}(x))$.

G, algorithm for generating keys

> This algorithm is probabilistic, takes a security parameter k as input and always outputs a pair of keys (pk, sk), the public and secret keys.

> We assume that the public key contains (perhaps implicitly) a description of $\mathscr{P}$, the set of plaintexts and $\mathscr{C}$, the set of ciphertexts. These do not have to be the same for every key.

E, algorithm for encryption

> This algorithm takes as input pk and $x \in \mathscr{P}$ and produces as output $E_{pk}(x) \in \mathscr{C}$. Note that E may be probabilistic: even for fix x and K, many different ciphertexts may be produced as output. In other words, the ciphertext will have a probability distribution that is determined from x and $\mathscr{K}$, typically uniform in some subset of the ciphertexts.

D, algorithm for decryption

> This algorithm takes as input $sk, y \in \mathscr{C}$ and produces as output $D_{sk}(y) \in \mathscr{P}$. It is allowed to be probabilistic, but is in most cases deterministic.

Trapdoor one-way function

> The concept came from Diffie and Hellman's research, but they themselves couldn't come up with any examples.

> An injective function $f : X \to Y$, where X, Y are finite sets, such that computing $f(x)$ from input x is easy, but computing x from $f(x)$, say for a random $x \in X$ is infeasible. f should come with a so-called trapdoor t_f, and given this extra information, it is easy to compute x from $f(x)$.
> Diffie and Hellman suggested that if such pairs (f, t_f) of function and trapdoor could be generated efficiently, then f could serve as the public key.

> in 1977, Rivest, Shamir and Adleman proposed the first construction such functions: RSA.

RSA

> RSA (Rivest-Shamir-Adleman) is a public-key cryptosystem based on the computational difficulty of factoring large numbers into their prime factors.

Key Generation

> - On input (even) security parameter value k, choose random $k/2$-bit primes p, q.
> - Compute the modulus $n = p \times q$. This value is part of both the public and private keys.
> - Choose a public exponent e such that $1 < e < (p-1)(q-1)$ and $\gcd(e, (p-1)(q-1)) = 1$ (aka $e \in \mathbb{Z}^*_{(p-1)(q-1)}$).
> - Compute the private exponent $d \equiv e^{-1} \mod (p-1)(q-1)$, meaning $e \cdot d \equiv 1 \mod (p-1)(q-1)$.

> The **public key** is (e, n) and the **private key** is (d, n).

> For RSA, we always have $\mathscr{P} = \mathscr{C} = \mathbb{Z}_n$.

Encryption

> To encrypt a message $m \in \mathbb{Z}_n$, the sender uses the recipient's public key: $c = m^e \mod n$ where c is the ciphertext.

Decryption

> To decrypt ciphertext c, the recipient uses their private key: $m = c^d \mod n$ This recovers the original message because $(m^e)^d \equiv m^{ed} \mod (p-1)(q-1) \equiv m^1 \equiv m \mod n$.

Security Intuition

> RSA's security relies on the fact that while multiplying two large primes is computationally easy, **factoring** the resulting product n back into p and q is extremely difficult with current algorithms. Without knowing p and q, an attacker cannot compute $(p-1)(q-1)$ or derive the private exponent d from the public exponent e. Modern RSA uses key sizes of 2048 or 4096 bits, making factorization infeasible with today's computational power.

CPA secure RSA; the PCRSA Scheme

> We can construct a probabilistic cryptosystem based on RSA (PCRSA), that is CPA secure. Proof of its CPA security not presented here.

> **Key Generation**: Generates a pair of RSA keys (n, e), (n, d) in the usual way. The set of messages is just $\{0, 1\}$ whereas the set of ciphertexts is $\mathbb{Z}_n^*$.
> **Encryption**: The encryption algorithm will encrypt a bit b by choosing a random number $x_b \in \mathbb{Z}_n^*$ such that the least significant bit of x_b is b. The ciphertext is now $c = x_b^e \mod n$.
> **Decryption**: Compute $x_b = c^d \mod n$ and extract the least significant bit.

> This is of course not very efficient but it doesn't have to be - PCRSA is commonly used just to agree on a shared secret in a symmetric crypto scheme (eg. AES). Efficiency can be increased by considering more than the last bit of x, but this makes it less secure.

Miller-Rabin test

> For using RSA, we need a way to generate new BIG primes on a whim. In theory this is simple:
>
> 1. On input k, select a uniformly random k-bit number x, or more precisely, choose random $x \in [2k, 2k]$.
> 2. Test if x is prime. If yes, then output x and stop, else go to Step 1. Of course, this actually requires "prime testing" to be a solved problem.

> The Miller-Rabin Primality test is one such option. It is fast, but probabilistic - in (arbitrarily) rare cases it may accept composite numbers as primes too.

> 1. On input an odd x, compute a, b such that $x - 1 = 2^a b$ where b is odd (divide $x - 1$ by 2 as many times as possible).
> 2. Choose a non-zero $w \in \mathbb{Z}_x$ at random and compute $d = gcd(w, x)$. If d is not 1, reject x and stop.
> 3. At this point we know that $w \in \mathbb{Z}_x^*$.
> 4. Compute the following list of numbers
>
> $$w^b \mod x, w^{2b} \mod x, w^{2^2 b} \mod x, ..., w^{2^{a-1} b} \mod x$$
>
> If any number on the list is -1 modulo x, or if $w^b \mod x = 1$, then accept x, else reject.

Logic Flow / Mechanism (Process)

RSA Key Generation

1. Choose random $k/2$-bit primes p, q and set $n = pq$.
2. [Choose $e \in \mathbb{Z}^*_{(p-1)(q-1)}$]
3. [Set $p = d^{-1} \mod (p-1)(q-1)$]
4. Output $pk = (n, e)$, $sk = (n, d)$.

RSA Encryption/Decryption Correctness Proof

1. Compute $D(n,d)(E(n,e)(x)) = (x^e \mod n)^d \mod n = x^{ed} \mod n$.
2. [_____]
3. [_____]
4. Conclude $= x$.

Miller-Rabin Primality Test

1. On input odd x, compute a, b such that $x - 1 = 2^a b$ where b is odd.
2. [_____]
3. [_____]
4. Accept if any $w^{2^i b} \mod x = -1$ or $w^b \mod x = 1$, else reject.

Random Prime Generation

1. Select uniform random k-bit x with $2^k \leq x < 2^{k+1}$.
2. [_____]
3. Repeat until accept.

Exam Simulation

1. **Oral Presentation Prompt:** "Walk me through the full specification of a public-key cryptosystem, including the roles of G, E, D and the correctness requirement. Present for up to 18 minutes, then field questions."

2. **Oral Proof Prompt:** "Prove that RSA decryption works for $x \in \mathbb{Z}^*_n$. Start from $D(E(x))$ and use group properties and CRT intuition. Be ready for follow-ups on $\phi(n)$."

3. **Oral Security/Implementation Prompt:** "Explain how to generate RSA keys, including prime generation via Miller-Rabin. Detail the test steps, error analysis for composites, and why repeated iterations suffice in practice."

Source Map

- CryptographyV6.pdf | Page 22-24 | Covers: Chinese Remainder Theorem (special/general case), group exponent arithmetic modulo $|G|$.

- CryptographyV6.pdf | Page 86-94 | Covers: Public-key cryptosystem definition, RSA spec/keygen/encryption/decryption, decryption proof, modular exponentiation, Extended Euclidean, Miller-Rabin test/probabilities, prime generation.