Introduction to Public Key Cryptography

→ Symmetric Cryptography revisited. → Most symmetric algorithms like AES and 3DES are fast, secure & used, but those are some issues

1. Key Distribution Problem: Key must be established between Alice & Bob under Secure channel which may not be possible
2. Number of Keys: Prob w/ large number of keys → comm. in network of n users ⇒ $\frac{n(n-1)}{2}$ keys whose every user stores $(n-1)$ keys securely — lot of overhead
3. No Protection Against Cheating by Alice or Bob

→ Symmetric: Same secret key used for encrypt & decrypt + encrypt & decrypt f^{-1} very similar

→ Principles of Asymmetric Cryptography → Encryption & Key Transport: Not necessary that key possessed by person encrypting message is secret

↳ Receiver must publish public encryption key but also has a matching secret key for decryption ∴ $k_{pub} \neq k_{pr}$

Bob can only decrypt using a secret key

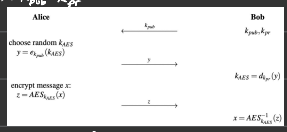
What about Transporting a key if we don't need Secret Channel?

Encrypt symmetric key using a public-key alg.

decrypted by Receiver

Both parties can use Symmetric ciphers

Pro: Payload encrypted w/ symmetric cipher → Fucks



→ In Practice: Symm keys must be shorter than asymm keys & need to be padded to same as input for asymm keys

↳ consider combination of secret key for symm encryption using asym encryption as 1 unified alg

↳ Do not expect msg as input ← Key Encapsulation Mechanism

↳ generate random value for symmetric secret key on the user side in feed into asymm encryption f^{te}

→ Asymmetric Schemes built using One-way F^n → F^n easy to compute on every input but hard to reverse any output of random input

→ Def^n of One-way F^n: F is OWF if:

$y = f(x)$ is computationally feasible

$x = f^{-1}(y)$ is infeasible

Alice just wants to invoke encaps f^n to create random key directly asym. encrypted → Bob runs decapsulate to decrypt and unwrap key

Ex: RSA based Integer Factorization, Discrete Logarithmic Problem, ..

→ Hybrid Protocols must be used to ensure Best of Both Worlds

Public Keys → Long Keys → Slow Computation

→ Main Security Mechanisms of Public-Key Algorithms: Ex: RSA, KEM, DHKE

1. Key Establishment: Protocols of establishing secret keys over insecure channels
2. Non-repudiation: Providing non-repudiation realized w/ Digital Signature Algorithms (DSA)
3. Integrity: DSA also ensure integrity
4. Identification: Identify entities using challenge-and-response protocols + Signatures
5. Encryption

Banking

Symmetric is good

Symmetric is not good

→ To show authenticity, use certificates which bind public key to identity

→ Public Key Algorithm Families:

Integer-Factorization Schemes: Several public-key schemes are based on the fact that it is difficult to factor large integers. The most prominent representative of this algorithm family is RSA.

Discrete Logarithm Schemes: There are several algorithms which are based on what is known as the discrete logarithm problem in finite fields. The most prominent examples include the Diffie–Hellman key exchange, Elgamal encryption and the Digital Signature Algorithm (DSA).

Elliptic Curve (EC) Schemes: A generalization of the discrete logarithm algorithm is elliptic curve public-key schemes. The most popular examples include the Elliptic Curve Diffie–Hellman key exchange (ECDH) and the Elliptic Curve Digital Signature Algorithm (ECDSA).

Each can be used for key establishment, non-repudiation & encryption

assumes absence of Quantum computers

Algorithm Family	Cryptosystems	Security Level (bits)		
		(80)	128	192	256
Integer factorization	RSA	(1024 bits)	3072 bits	7680 bits	15360 bits
Discrete logarithm	DH, DSA, Elgamal	(1024 bits)	3072 bits	7680 bits	15360 bits
Elliptic curves	ECDH, ECDSA	(160 bits)	256 bits	384 bits	512 bits
Symmetric-key	e.g., AES	(80 bits)	128 bits	192 bits	256 bits

Bottleneck for small PCs

→ All 3 families based on number theoretic f^n → very long operands & keys → Highly secure (Security Level of n bits → Best attack requires 2^n steps)

Essential Number Theory:

GCD of $r_0, r_1 \in \mathbb{Z}^+$ $\gcd(r_0, r_1)$ is the largest $q \in \mathbb{Z}^+$ s.t. $q \mid r_0$ and $q \mid r_1$

→ For large numbers, Euclid's Algorithm used: if r_0 & r_1 have common divisor g, any linear combination of r_0 and r_1 is divisible by g → $\gcd(r_0, r_1) = \gcd(r_0 - r_1, r_1) = g$ for $r_0 > r_1$

Say $\gcd(r_0, r_1) = g$. Since $g \mid r_0$ and $g \mid r_1$, $r_0 = g \cdot x$ and $r_1 = g \cdot y$ where $x > y$ and $x \& y$ are coprime numbers → Since $x-y$ and y are coprime, $\gcd(r_0 - r_1, r_1) = \gcd((g \cdot (x-y)), g \cdot y) = g$

→ If we continue this process, $\gcd(r_0, r_1) = \gcd(r_0 - r_1, r_1) = \gcd(r_0 - 2r_1, r_1) = \cdots$

$= \gcd(r_0 - m r_1, r_1)$ where $r_0 - m r_1 > 0$

To choose Max m: $\gcd(r_0, r_1) = \gcd(r_0 \bmod r_1, r_1)$

$= \gcd(r_1, r_0 \bmod r_1)$ — $r_0 \bmod r_1 < r_1$ we swap

Reduce Problem of Finding GCD of 2 numbers → Find GCD of 2 smaller numbers

$\gcd(r_0, r_1) = \cdots = \gcd(r_\ell, 0) = r_\ell$

Euclidean Algorithm
Input: positive integers r_0 and r_1 with $r_0 > r_1$
Output: $\gcd(r_0, r_1)$
Initialization: $i = 1$
Algorithm:

```
1   DO
1.1   i = i + 1
1.2   r_i = r_{i-2} mod r_{i-1}
      WHILE r_i ≠ 0
2   RETURN
      gcd(r_0, r_1) = r_{i-1}
```

no common factors

efficient & widely used

$r_\ell = \gcd(r_0, r_1)$
$= s_\ell r_0 + t_\ell r_1$
$\cdots$
$= s r_0 + t r_1$

→ Extension of Euclid → compute modular inverses used in PKC

↳ EEA computes $\gcd(r_0, r_1) = s \cdot r_0 + t \cdot r_1$ for $s, t \in \mathbb{Z}$ — how to compute s & t

↳ Execute Euclid's but express current $r_i = s_i r_0 + t_i r_1$

Diophantine Equation

→ Derivation of Recursive Formulae for s_i & r_i:

$r_{i-2} = \lfloor s_{i-2} \rfloor r_0 + \lfloor t_{i-2} \rfloor r_1 = q_{i-1} \cdot r_{i-1} + r_i \Rightarrow r_i = r_{i-2} - q_{i-1} \cdot r_{i-1}$

$r_{i-1} = \lfloor s_{i-1} \rfloor r_0 + \lfloor t_{i-1} \rfloor r_1$

Starting Values: $s_0 = 1$, $s_1 = 0$, $t_0 = 0$, $t_1 = 1$

$s_i = s_{i-2} - q_{i-1} s_{i-1}$

$t_i = t_{i-2} - q_{i-1} t_{i-1}$

Valid for $i \geq 2$

Extended Euclidean Algorithm (EEA)
Input: positive integers r_0 and r_1 with $r_0 > r_1$
Output: $\gcd(r_0, r_1)$, as well as s and t such that $\gcd(r_0, r_1) = s \cdot r_0 + t \cdot r_1$.

Initialization:
$s_0 = 1 \quad t_0 = 0$
$s_1 = 0 \quad t_1 = 1$
$i = 1$

Algorithm:

```
1   DO
1.1   i = i + 1
1.2   r_i = r_{i-2} mod r_{i-1}
1.3   q_{i-1} = (r_{i-2} - r_i)/r_{i-1}
1.4   s_i = s_{i-2} - q_{i-1} · s_{i-1}
1.5   t_i = t_{i-2} - q_{i-1} · t_{i-1}
      WHILE r_i ≠ 0
2   RETURN
      gcd(r_0, r_1) = r_{i-1}
      s = s_{i-1}
      t = t_{i-1}
```

→ We want to find modular inverse of integer. Say we want inverse of key value a modulo 26 — only exists if $\gcd(r_0, r_1) = 1$

↳ assume we want inv. of r_1 mod r_0 for $r_1 < r_0$

$s \cdot r_0 + t \cdot r_1 = 1 = \gcd(r_0, r_1)$

$\Rightarrow s \cdot 0 + t \cdot r_1 \equiv 1 \bmod r_0$

$r_1 \cdot t \equiv 1 \bmod r_0$

$t \equiv r_1^{-1} \bmod r_0$

→ Show how EEA can be used to compute multiplicative inverses in $GF(2^m)$

↳ Inputs of EEA are in $A(x)$ and irreducible $P(x)$

EEA computes polynomials $s(x)$ and $t(x)$ alongside $\gcd(P(x), A(x))$ s.t.:

$s(x)P(x) + t(x)A(x) = \gcd(P(x), A(x)) = 1$

$P(x)$ is irreducible $\Rightarrow \gcd = 1$

$s(x)0 + t(x)A(x) \equiv 1 \bmod P(x)$

$t(x) \equiv A^{-1}(x) \bmod P(x)$

Extended Euclidean Algorithm for polynomials over GF(2)
Input: nonzero polynomials $r_0(x)$ and $r_1(x)$ with $\deg(r_0) \geq \deg(r_1)$.
Output: $\gcd(r_0, r_1)$, as well as polynomials $s(x)$ and $t(x)$ such that $\gcd(r_0, r_1) = s(x) \cdot r_0(x) + t(x) \cdot r_1(x)$.

Initialization
$s_0 = 1 \quad t_0 = 0$
$s_1 = 0 \quad t_1 = 1$
$i = 1$

Algorithm

```
1   DO
1.1   i = i + 1
1.2   j = deg(r_{i-2}) − deg(r_{i-1})
1.3   IF j ≥ 0
        r_i = r_{i-2} + x^j r_{i-1}
        s_i = s_{i-2} + x^j s_{i-1}
        t_i = t_{i-2} + x^j t_{i-1}
1.4   ELSE
        r_i = r_{i-1} + x^{-j} r_{i-2}
        s_i = s_{i-1} + x^{-j} s_{i-2}
        t_i = t_{i-1} + x^{-j} t_{i-2}
      WHILE r_i ≠ 0
2   RETURN
      gcd(r_0, r_1) = r_{i-1}
      s(x) = s_{i-1}
      t(x) = t_{i-1}
```

∴ If we need a^{-1} mod m, we apply EEA w/ parameters m & a.

→ Consider $\mathbb{Z}_n = \{0, \ldots, n-1\}$ → want to know how many numbers in $\mathbb{Z}_n$ are relatively prime to n

→ Euler's Phi Function: Number of integers in $\mathbb{Z}_n$ coprime to n is $\phi(n)$

→ Theorem: Let n have canonical factorization

$n = p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots \cdot p_n^{e_n}$ p_i are distinct primes and $e_i \in \mathbb{Z}^+$

then $\phi(n) = \prod_{i=1}^{n} (p_i^{e_i} - p_i^{e_i - 1})$