Public–key cryptography

- Suggested by Diffie & Hellman 1976
- Instead of one secret, shared key (with the associated problems of key distribution):
- Use a key pair (e,d) for each user
  - one for encryption, one for decryption
  - one private (secret), one public
  - s.t. $c = E_e(m)$, $m = D_d(c)$
  - in some cases $E=D$ and

$$m = D_e(E_d(m)) = E_e(D_d(m)) = D_d(E_e(m))$$

i.e. the keys (e,d) are inverses of each other

Both confidentiality and authenticity

- A has (e_A, d_A), B has (e_B, d_B)
  - where e is private, d public
- Confidentiality A → B: $c = E_{dB}(m)$
  - can only be decrypted by D_{eB}
- Authenticity A → B: $c = E_{eA}(m)$
  - can be decrypted by anyone, but can only have been encrypted by E_{eA}
- Both conf&auth A → B: $c = E_{dB}(E_{eA}(m))$
  - decrypted by $D_{dA}(D_{eB}(c))$

Requirements on PKS

1. Easy to generate (e,d)
2. Easy to encrypt $E_k(m)$ given k and m
3. Easy to decrypt $D_k(c)$ given k and c
4. Computationally infeasible to find e given d
5. Computationally infeasible to find m given e and $c = E_e(m)$
6. $m = D_e(E_d(m)) = E_e(D_d(m)) = D_d(E_e(m))$

   (not always)

One–way trapdoor functions

- A *one–way* function f is a (1–1) function s.t.
  - $y = f(x)$ is easy to compute, but $x = f^{-1}(y)$ infeasible
- A *trapdoor* function f is a function s.t.
  - $x = f_k^{-1}(y)$ is easy <u>iff</u> k is known (the key)
- *Easy*: computable in polynomial time, proportional to n^a: n length of input, a constant
- *Infeasible*: not computable in polynomial time, e.g. only in 2^n

Examples of one–way trapdoors

- Breaking a leg
- Squeezing toothpaste out of a tube
- Mixing colours
- Multiplication of large prime numbers
  - factorization is hard
- Exponentiation of large numbers
  - discrete logarithms are hard

Exponential cryptography

- RSA: for $\boldsymbol{M}=\boldsymbol{C}=\boldsymbol{Z}_n$
  - $c = m^e \bmod n$
  - $m = c^d \bmod n$
- Example: $e = 5, d = 77, n = 119, m = 19$
  - $c = 19^5 = 2476099 \bmod 119 = 66$
  - $m = 66^{77} \approx 1.27 \cdot 10^{140} \bmod 119 = 19$
- Seems impractical?
- How do we find (e,d) pairs s.t. it works?

Review: Modular arithmetic

- $a \equiv b \pmod{n}$ if $a-b = kn$ for some k
  - e.g. $17 \equiv 7 \pmod{5}$
- Write $a \bmod n = r$
  if r is the (positive) residue of a/n
  - implies $a \equiv r \pmod{n}$
- Let $\Diamond$ be an operation: $+, -, \cdot$. Then

  $(a \Diamond b) \bmod n = ((a \bmod n) \Diamond (b \bmod n)) \bmod n$
- $(\boldsymbol{Z}_n, \{+, -, \cdot\})$ is a commutative ring:
  usual commutative, associative, distributive laws

Efficient exponentiation mod n

- $(a \cdot b) \bmod n = ((a \bmod n) \cdot (b \bmod n)) \bmod n$,
  so
  $a^b \bmod n$ can be computed without generating astronomical numbers:
  - $3^5 \bmod 7 = 243 \bmod 7 = 5$
    $3^5 \bmod 7 = (3^2)^2 \cdot 3 \bmod 7$
    $= ((3^2 \bmod 7) \cdot (3^2 \bmod 7) \bmod 7) \cdot 3 \bmod 7$
    $= ((9 \bmod 7) \cdot (9 \bmod 7) \bmod 7) \cdot 3 \bmod 7$
    $= (2 \cdot 2 \bmod 7) \cdot 3 \bmod 7 = 12 \bmod 7 = 5$
- Algorithm description in figure 6.7

Rivest, Shamir, Adleman

- RSA:
  - $c = m^e \bmod n$
  - $m = c^d \bmod n$
  - $m = (m^e \bmod n)^d \bmod n = m^{ed} \bmod n \; (= m^{de} \bmod n)$
- Find such e,d, and n using Euler's theorem

Review: Modular arithmetic (cont)

x is the multiplicative inverse of a modulo n, written a^{-1}, if $ax \equiv 1 \pmod{n}$

- Ex: $3 \cdot 5 \equiv 1 \pmod{14}$

The reduced set of residues modulo n is $\boldsymbol{Z}^*_n = \{\ \mathrm{x} \in \boldsymbol{Z}_n - \{0\} : \gcd(x,n) = 1\ \}$

Euler's totient function $\phi(n)$ is the cardinality of $\boldsymbol{Z}^*_n$

Ex: $\boldsymbol{Z}^*_{24} = \{\ 1, 5, 7, 11, 13, 17, 19, 23\ \}$, $\phi(24)=8$

Euler and primes

Lemma: If p and q are prime, then

$\phi(pq) = (p-1)\cdot(q-1) = \phi(p)\cdot\phi(q)$

Proof: in $\boldsymbol{Z}_{pq} = [0, pq-1]$, the numbers <u>not</u> relatively prime to pq are (in addition to 0):

- $q, 2q, \ldots, (p-1)q$
- $p, 2p, \ldots, (q-1)p$

so $\phi(pq) = pq - ((p-1)+(q-1)+1) = pq - p - q + 1$
$= (p-1)(q-1)$

Note: $\phi(p)=p-1$, for p a prime

Euler's theorem

Theorem: for all a and n s.t. $\gcd(a,n) = 1$ (they are relatively prime),
$a^{\phi(n)} \bmod n = 1$

Corollary: for p and q primes, $n=pq$ and $0<m<n$,
$m^{\phi(n)+1} = m^{(p-1)(q-1)+1} \equiv m \pmod{n}$

If $ed \bmod \phi(n) = 1$, then $ed = t\phi(n)+1$ for some t, so (e,d) is a working key pair (by the corollary).

Making RSA key pairs

$ed \bmod \phi(n) = 1$, and if $\gcd(d,\phi(n))$, Euler's theorem then gives
$e = d^{\phi(\phi(n))-1} \bmod \phi(n)$

Computing e from d and $\phi(n)$ is easy, and even more efficient with an extension of Euclid's algorithm for $\gcd(d,\phi(n))$ (see section 7.5)

Having $\phi(n)$ makes RSA easy to break; $\phi(n)=(p-1)(q-1)$, so p and q must be secret, while $n=pq$ must be public.

Factorizing products of large (prime) numbers is hard!

Factorization

- Factorization of $n=pq$ (to find $\phi(n)$) is difficult if p and q are large
  - August 1999: 155–digit (512–bit) n factorized
    - 35.7 CPU–years (7.4 months) using 160 workstations, 120 PII, 12 strong workstations, and one Cray
  - February 1999: 140–digit n factorized
    - 8.9 CPU–years (9 weeks) using 125 workstations, 60 Pcs, and one Cray
  - 1024–bit n expected to be 40 million times harder than 140–bit

Finding large primes

- Naıve methods too time–consuming
- Guess a number and test it many times
  - gives high probability of primeness
    - more likely that a bit is flipped by cosmic radiation
  - for 200 digits, approx 70 guesses each tested 100 times is enough
- Desired properties to make factorization harder
  - p, q of different length
  - $(p-1)$ and $(q-1)$ with large prime factors
  - $\gcd(p-1, q-1)$ small

RSA cryptanalysis

- Brute force not feasible with large keys (typically 1024–2048 bits)
- Factorization difficult, but mathematical advances may make it significantly easier
  - 1977 challenge: 428–bit *n* would take 40 quadrillion years – took 8 months (1994)
- Timing attack
  - based on the time to decrypt (ciphertext–only attack)
  - countermeasures: random delay, "blinding"

Simple RSA key exchange

- A sends public key d_A and id_A to B
- B selects a random session key k_S
- B sends $c = E_{dA}(k_S)$ to A
- A decrypts $k_S = D_{eA}(c)$

Vulnerable to man–in–the–middle attack

Generators and discrete logarithms

- a is a *primitive root* (or *generator*) modulo p if $\mathbf{Z}_p^*$ is generated by exponentiation of a mod p
  - ex: 2 is a primitive root mod 11:
    $\mathbf{Z}_{11}^* = \{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 \}$
    $= \{ 2^{10}, 2^1, 2^8, 2^2, 2^4, 2^9, 2^7, 2^3, 2^6, 2^5\}$ mod 11
- For any b, and a a generator mod p, a unique i exists s.t. $b=a^i$ mod p.
- i is the *discrete logarithm* (index) of b for base a, mod p

  write $i = \mathrm{ind}_{a,p}(b)$

Diffie–Hellman key exchange

- Public: prime q, generator a modulo q.
- User A selects private, random $x_A < q$, and computes $y_A = a^{xA} \bmod q$
- User B selects and computes x_B and y_B same way
- Each sends his y value to the other, and computes the shared key:
  - $K = (y_B)^{xA} \bmod q = (a^{xB} \bmod q)^{xA} \bmod q$
    $= (a^{xB \cdot xA}) \bmod q = (a^{xA \cdot xB}) \bmod q = (a^{xA} \bmod q)^{xB} \bmod q$
    $= (y_A)^{xB} \bmod q = K$

Diffie–Hellman cryptanalysis

- Known: q, a, y_A, y_B
- To get k, need x_A or x_B

  $x_A = \mathrm{ind}_{a,q}(y_B)$

- For q a large prime, this is computationally infeasible

ElGamal PKS

- Like Diffie–Hellman, but after exchanging *y* values, a message $m < q$ can be encrypted:
  - select random k in $[1, q-1]$
  - compute $K = y_B^{\,k} \bmod q$
  - send (C_1, C_2) where
    - $C_1 = a^k \bmod q$
    - $C_2 = Km \bmod q$
  - decryption:
    - $K = C_1^{\,xB} \bmod q$
    - $m = C_2 K^{-1} \bmod q$