Public Key Algorithms

- hash: irreversible transformation(message)
- secret key: reversible transformation(block)

	encryption	digital signatures	authentication
RSA	yes	yes	yes
El Gamal	no	yes	no
Zero-knowledge proofs	no	no	yes

Diffie-Hellman: exchange of secrets

all: pair (public, private) for each *principal*

Modular Addition

- addition modulo (mod) K ➠ (poor) cipher with key K
- *additive inverse*: $-x$: add until modulo (or 0)
- “decrypt” by adding inverse

Modular Multiplication

$\cdot$	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9
2	0	2	4	6	8	0	2	4	6	8
3	0	3	6	9	2	5	8	1	4	7

- multiplication by 1, 3, 7, 9 works as cipher
- multiplicative inverse x^{-1}: $y \cdot x = 1$
- only 1, 3, 7, 9 have multiplicative inverses (e.g., 7 $\leftrightarrow$ 3)
- use *Euclid's Algorithm* to find inverse

Totient Function

- x, m relatively prime = no other common factor than 1
- relatively prime $\neq$ prime (9 rel. prime 10)
- e.g., 6 not relatively prime to 10: 2 divides both 6 and 10
- *totient function* $\phi(n)$: number of numbers less than n relatively prime to n
  - if n prime, $\{1, 2, \ldots, n-1\}$ are rp ➠ $\phi(n) = n - 1$
  - if $n = p \cdot q$, p, q distinct prime ➠ $\phi(n) = (p-1)(q-1)$:
    - $n = pq$ numbers in $\{0, 1, 2, \ldots, n-1\}$; exclude non-rp
    - ➠ exclude multiples of p or q
    - p multiples of $q < pq$ (0,1,...), q multiples of $p < pq$
    - thus, exclude $p + q - 1$ numbers – don't count 0 twice
    - $\phi(pq) = pq - (p + q - 1) = (p-1)(q-1)$

Modular Exponentiation

$x^y \bmod n \neq x^{y+n} \bmod n!$

x^y	0	1	2	$\boxed{3}$	4	5	6	7	8	9	10	11	12
0		0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	2	4	8	6	2	4	8	6	2	4	8	6
3	1	3	9	7	1	3	9	7	1	3	9	7	1
4	1	4	6	4	6	4	6	4	6	4	6	4	6
5	1	5	5	5	5	5	5	5	5	5	5	5	5
6	1	6	6	6	6	6	6	6	6	6	6	6	6
7	1	7	9	3	1	7	9	3	1	7	9	3	1
8	1	8	4	2	6	8	4	2	6	8	4	2	6
9	1	9	1	9	1	9	1	9	1	9	1	9	1

Modular Exponentiation

- encryption: x^3 works, x^2 does not
- exponentiative inverse y of x: $(a^x)^y = a$
- columns: $1 = 5, 2 = 6, 3 = 7, \ldots$
- $x^y \bmod n = x^{y \bmod \phi(n)} \bmod n$
- $rp(10) = \{1, 3, 7, 9\}$ ➠ $\phi(n) = 4$
- true for *almost* all n: any n =product of distinct primes (*square-free*)
- for any y with $y = 1 \pmod{\phi(n)}$ ➠ $x^y \bmod n = x \bmod n$ (e.g., 1, 5 and 9)

RSA

- Rivest, Shamir, Adleman
- variable key length (common: 512 bits)
- ciphertext length = key length
- slow ➠ mostly used to encrypt secret for secret key cryptography

RSA Algorithm

Generate private and public key:

- choose two large primes, p and q, about 256 bits (77 digits) each
- $n = p \cdot q$ (512 bits), don't reveal p and q
- factoring 512 bit number is hard

public key: e rp $\phi(n) = (p-1)(q-1)$ ➠ $\langle e, n \rangle$

private key: $d = (e \bmod \phi(n))^{-1}$ ➠ $\langle d, n \rangle$

encryption: of $m < n$: $c = m^e \bmod n$

decryption: $m = c^d \bmod n$

verification: $m = s^e \bmod n$ (signature s)

RSA example

$$\begin{aligned}
p &= 47 \\
q &= 71 \\
n &= pq = 3337 \\
e &= 79 \text{ prime, i.e., rp to } (p-1)(q-1) \\
d &= 79^{-1} \bmod 3220 = 1019 \\
m &= 688232687666683 \\
m_1 &= 688 \\
c_1 &= 688^{79} \bmod 3337 = 1570 \\
p_1 &= 1570^{1019} \bmod 3337 = 688
\end{aligned}$$

Why does RSA work?

- $n = pq$, $\phi(n) = (p-1)(q-1)$
- $de = 1 \pmod{\phi(n)}$ since e rp $\phi(n)$ and $d = e^{-1}$
- $x^{de} = x \pmod{n} \forall x$
- encryption: x^e
- decryption: $(x^e)^d = x^{ed} = x$
- signature: reverse

Why is RSA secure?

- factor 512-bit number: half million MIPS years (= all US computers for one year)
- given public key $\langle e, n \rangle$
- need to find exponentiative inverse of e
- need to know p, q to compute $\phi(n)$
- abuse: if limited set of messages, can compare ➠ append random number
- 2/2/1999: RSA-140 was factored.

RSA Efficiency: Exponentiating

- $123^{54} \bmod 678 = (123 \cdot 123 \cdots)/678$
- modular reduction after each multiply:
- $(a \cdot b \cdot c) \bmod m = (((a \cdot b) \bmod m) \cdot c) \bmod m$

$$\begin{aligned}
123^2 &= 123 \cdot 123 = 15129 = 213 \pmod{678} \\
123^3 &= 123 \cdot 213 = 26199 = 435 \pmod{678} \\
123^4 &= 123 \cdot 435 = 53505 = 435 \pmod{678}
\end{aligned}$$

- 54 small multiplies, 54 divides
- exponent power of 2: 123^{32}

$$\begin{aligned}
123^2 &= 123 \cdot 123 = 15129 = 213 \pmod{678} \\
123^4 &= 213 \cdot 213 = 45369 = 671 \pmod{678} \\
123^8 &= 621 \cdot 621 = 385641 = 213 \pmod{678}
\end{aligned}$$

- $123^{2x+1} = 123^{2x} \cdot 123$

RSA Efficiency: Exponentiating

$54 = 110110_2$; start with exponent “1”.

$$\begin{array}{rcccl}
10 & \hookleftarrow & 123^2 & = & 123 \cdot 123 = 15129 = 213 \pmod{678} \\
11 & +1 & 123^3 & = & 213 \cdot 123 = 26199 = 435 \pmod{678} \\
110 & \hookleftarrow & 123^6 & = & 435 \cdot 435 = 189225 = 63 \pmod{678} \\
1100 & \hookleftarrow & 123^{12} & = & 63 \cdot 63 = 3969 = 579 \pmod{678} \\
1101 & +1 & 123^{13} & = & 579 \cdot 123 = 71217 = 27 \pmod{678} \\
11010 & \hookleftarrow & 123^{26} & = & 27 \cdot 27 = 729 = 51 \pmod{678} \\
11011 & +1 & 123^{27} & = & 51 \cdot 123 = 6273 = 171 \pmod{678} \\
110110 & \hookleftarrow & 123^{54} & = & 171 \cdot 171 = 29241 = 87 \pmod{678}
\end{array}$$

or $x^{54} = (((((x)^2x)^2)^2x)^2x)^2 = 87 \pmod{678}$

➠ 8 multiplies, 8 divides ➠ linearly with exponent bits

RSA Implementation

public key: $O(k^2)$, private key: $O(k^3)$, key generation: $O(k^4)$

DES	Pijnenburg PCC101	CFB	90 Mb/s
	Vasco CRY12C102	CFB	22 Mb/s
RSA	Pijnenburg PCC202	512	40 kb/s
		1024	25 kb/s
	Vasco PQR512	512	32 kb/s

- fastest RSA hardware: 300 kb/s
- 90 MHz Pentium: throughput (private key) of 21.6 kb/s, 7.4 kb/s per second with a 1024-bit modulus
- DES software: 100 times faster than RSA
- DES hardware: 1,000 to 10,000 times faster

Finding Big Primes p and q

- infinite number of primes, probability $1/\ln n$
- ten-digit number: 1 in 23, hundred-digit: 1 in 230
- pick at random and check if prime
- bad: divide by all $\sqrt{n}$
- *Euler's Theorem:* a rp n ➠ $a^{\phi(n)} = 1 \pmod{n}$
- if n prime, $\phi(n) = n - 1$

Theorem 1 (Fermat's Little Theorem) *If p is prime and $0 < a < p$, $a^{p-1} = 1 \pmod{p}$*

- if p not prime, does not usually hold
- ➠ pick some $a < n$, compute $a^{n-1} \bmod n \overset{?}{\rightarrow} 1$
- probability of accepting bad n: 10^{13} ➠ repeat

Carmichael Numbers

- *Carmichael numbers n:* not prime, but $a^{n-1} = 1 \pmod{n} \forall a$ (where a not a factor in n)
- infinitely many
- first few: 561, 1105, 1729, 2465, 2821, 6601, 8911
- 246,683 below 10^{16}
- example: $7^{560} \bmod 561 = 1$, but $3^{560} \bmod 561 = 375$

Finding Big Primes p and q: Miller and Rabin

Variation on Fermat test:

- express $n-1$ as $2^b c$, where $b \geq 0$
- compute $a^{n-1} \pmod{n}$ (Fermat) as $(a^c)^{2^b} \pmod{n}$
- ➠ square b times
- if not 1 ➠ not prime; if 1, test:
  - if $a^c \pmod{n} \neq 1$ ➠ squaring not-1 $\rightarrow 1$
  - ➠ square root of 1
  - rule: if n is prime $\pmod{n}$, $\sqrt{1}$ are 1 and $-1 (= n-1)$
  - ➠ if $\sqrt{1} \neq \pm 1$, n not prime
  - try many values for a; 75% of a fail the test if n not prime

Big Primes: Implementation

1. pick odd random number n
2. check $n/\{3, 5, 7, 11, \ldots\}$ and try again
3. repeat until failure or confidence:
   (a) pick random a and compute $a^c \pmod n$, with $n - 1 = 2^b c$
   (b) compute a^c, then b times: $(a^c)^2$
   (c) if result = 1: operand = ±1 ? ➠ no prime if not

Finding d and e

- e = any number rp to $(p-1)(q-1)$
- $ed = 1 \pmod{\phi(n)}$ ➠ Euclid's algorithm

Options for picking e:

1. pick randomly until e is rp to $(p-1)(q-1)$
2. choose e and pick p, q so that $(p-1), (q-1)$ are rp to e

Having a Small Constant e

- e same small number
- d can't be small (searchable)
- $e = 3$ or $e = 65537$
- can't use 2: not rp to $(p-1)(q-1)$
- message must be bigger than $\sqrt[3]{n}$
- send copies of message to three people: $e_i = \langle 3, n_i \rangle$
  - Trudy: $m^3 \bmod n_1 n_2 n_3 = m^3$ (Chinese remainder)
  - ➠ choose random/individualized padding

RSA: $e = 3$

- 3 rp to $\phi(n) = (p-1)(q-1)$ since $d = e^{-1}$
- each $p-1$, $q-1$ must be rp to 3
- 3 is factor of x ➠ $x \bmod 3 = 0$
- $(p-1)$ rp 3 ➠ $p = 2 \pmod 3$ ➠ $(p-1) = 1 \pmod 3$
- $(q-1)$ rp 3 ➠ $q = 2 \pmod 3$ ➠ $(q-1) = 1 \pmod 3$
- choose $p = r \cdot 3 + 2$, r random, odd

RSA: $e = 65537$

- 65537 = $2^{16} + 1$, (Mersenne prime: $2^n - 1$!)
- only 17 multiplies to exponentiate: $x^{2^{16}} x$
- random 512-bit number: 768 multiplies
- avoid “3” problems:
  1. few m with $m^{65537} < n$ (512 bits)
  2. have to send to 65,537 recipients
  3. n rp $\phi(n)$ ➠ reject $p, q = 1 \pmod{65537}$

RSA Threats: Smooth Numbers

- product of “small” primes
- signed m_1, m_2 ➠ can compute signatures on $m_1 \cdot m_2, m_1/m_2, m_1^j, m_2^j, m_1^j m_2^k$
- example: $m_1^2 : (m_1^d \bmod n)^2 \bmod n$
- if m_1/m_2 is prime, can fake signature on that prime
- ➠ any product of this collection
- pad with zero on left ➠ small number ➠ smooth $\uparrow$
- pad on right with x bits $\equiv n \cdot 2^x$
- pad on right with random data ➠ cube root problem

RSA Threats: Cube Root Problem

- Carol wants your signature for message with digest h
- message digest h; $h' =$ pad with zeros on *right*
- “signature” $r = \lceil \sqrt[3]{h'} \rceil \Rightarrow r^e = r^3 = h'$

Public Key Cryptography Standards (PKCS)

- operational standards
- deal with threats (smooth numbers, multiple recipients, . . .)
- encryption with PKCS#1
  - random padding prevents guessing from known messages
  - random padding prevents $e = 3$, multiple-recipient attack
  - cube root decryption ➠ longer than 21 bytes ($> 11 +$ data)
- signing with PKCS#2
  - large padding ➠ not smooth
  - include digest algorithm ➠ prevent spoofing

PKCS #1 – RFC 2313

Also X.509:

```
RSAPublicKey ::= SEQUENCE {
  modulus INTEGER,     -- n
  publicExponent INTEGER    -- e
}
```

Encryption block = $00|\mathrm{BT}|\mathrm{PS}|00|D$ with padding PS of $k - 3 - |D|$ octets.

0	private-key	00
1	private-key	FF (large!)
2	public-key	pseudo-random

PKCS #1 Signature

```
DigestInfo ::= SEQUENCE {
  digestAlgorithm DigestAlgorithmIdentifier,
  digest Digest
}
DigestAlgorithmIdentifier ::= AlgorithmIdentifier
AlgorithmIdentifier ::= SEQUENCE {
  algorithm OBJECT IDENTIFIER,
  parameters ANY DEFINED BY algorithm OPTIONAL
}
md5 OBJECT IDENTIFIER ::=
{ iso(1) member-body(2) US(840) rsadsi(113549)
  digestAlgorithm(2) 5 }
Digest ::= OCTET STRING
```

Diffie-Hellman Key Exchange

- shared key, public communication
- no authentication of partners
- p prime, $\approx$ 512 bits, public
- $g < p$, public
- Alice, Bob choose random, secret S_A, S_B
- transmit $T_A = g^{S_A} \bmod p$, $T_B = g^{S_B} \bmod p$
- Alice computes $T_B^{S_A} \pmod p = (g^{S_B})^{S_A} \pmod p$
- both get same number = key
- would need to compute discrete logs to get S_A from g^{S_A}
- not secure against bucket-brigade attacks

- public numbers instead of invention

Bucket Brigade Attack

- “man-in-the-middle”
- X establishes security association with Alice, Bob
- can read/write from/to both
- relays messages, passwords between them
- prevention: make $g^{S_A} \bmod p$ public ➠ can’t be replaced

Diffie-Hellman: Offline

- Bob publishes $\langle p_B, g_B, T_B \rangle$
- Alice computes $K_{AB} = T_B^{S_A} \bmod p_B$
- Alice sends $g_B^{S_A} \bmod p_B$ to Bob

El Gamal Signatures

- D-H: public: $\langle g, p, T\rangle$; private: $S; g^s \mod p = T$
- new public/private key for each message
- compute $T_m = g^{S_m} \mod p$ for random S_m for each msg. m
- digest $d_m = m|T_m$
- signature = $X = S_m + d_m S \pmod{p-1}$
- transmit m, X, T_m
- verification: $\frac{g^X}{=?T_m T^{d_m} \mod p}$

$$g^X = g^{S_m + d_m S} = g^{S_m} g^{S d_m} = T_m T^{d_m} \pmod{p}$$

El Gamal Properties

Exercises:

- message modification ➠ signature won't match
- signature does not divulge S
- don't know S ➠ can't sign

Digital Signature Standard (DSS)

- related to El Gamal, but some computations $\bmod q$, $q = 160$ bits $< |p|$ = 512 bits
- speeded up for signer rather than verifier: chip cards

DSS Algorithm

1. generate public p (512 bit prime) and q (160 bit prime)

$$p = kq + 1$$

2. generate public g

$$g^q = 1 \pmod{p}$$

3. choose long-term $\langle T, S\rangle$ with random S

$$T = g^S \bmod p \text{ for } S < q$$

4. choose $\langle T_m, S_m\rangle$ with random S_m
   - $T_m = ((g^{S_m} \bmod p) \bmod q$
   - calculate $S_m^{-1} \bmod q$
5. calculate d_m = SHS(message)

6. signature $X = S_m^{-1}(d_m + ST_m) \bmod q$
7. transmit m, T_m, X
8. verify based on d_m: $z \overset{?}{=} T_m$

$$\begin{aligned} x &= d_m \cdot X^{-1} \bmod q \\ y &= T_m \cdot X^{-1} \bmod q \\ z &= (g^X \cdot T^y \bmod p) \bmod q \end{aligned}$$

DSS Algebra

$$\begin{aligned}
v &= (d_m + ST_m)^{-1} \bmod q \\
X^{-1} &= (S_m^{-1}(d_m + ST_m))^{-1} = S_m(d_m + ST_m)^{-1} \\
&= S_m v \bmod q \\
x &= d_m X^{-1} = d_m S_m v \bmod q \\
y &= T_m X^{-1} = T_m S_m v \bmod q \\
z &= g^x T^y = g^{d_m S_m v} g^{ST_m S_m v} \\
&= g^{(d_m + ST_m) S_m v} = g^{S_m} = T_m \bmod p \bmod q
\end{aligned}$$

any multiple of q in exponent drops out

RSA vs. DSS

- fixed moduli
- $\langle p, q, g\rangle$ ➠ pick one ➠ juicy target
- trapdoor primes
- slower than RSA($e = 3$), but signatures can be done ahead of time
- needs per-message random secret
- patent (Schnorr)

Zero-Knowledge Proofs

- prove knowledge without revealing it
- RSA signatures
- graph isomorphism: rename vertices
- Alice: graph A and $B \sim A$
- public key: graphs A, B
- private key: mapping between vertices
- Alice: create G_i and sends to Bob
- Bob $\rightarrow$ Alice: how did A or $B \rightarrow G_i$?
- zero-knowledge: Bob knows some G_i's
- Fred can create G_i from either A or B, but not both

Zero-Knowledge Proofs: Fiat-Shamir

- Alice: public key $\langle n, v\rangle$, $n = pq$
- v: Alice knows secret $s = \sqrt{v} \pmod{n}$

1. Alice chooses k random numbers $r_1, \ldots, r_k$
2. Alice sends $r_i^2 \bmod n$
3. Bob chooses a random subset 1 of r_i^2

	subset 1	subset 2
Alice sends	$sr_i \bmod n$	$r_i \bmod n$
Bob checks	$(sr_i)^2$	$(r_i)^2$
=?	vr_i^2	$(r_i)^2$
Fred		$(r_i)^2$ (easy)

4. finding square roots is hard

5. Fred gets some $\langle r_i^2, sr_i \rangle$
6. can use these for subset 1, pick own for subset 2
7. Carol picks which she wants

much faster than RSA: 45 multiplies for Alice, Bob