Intro to public-key ciphers

A **symmetric** or **private-key** cipher is one in which knowledge of the *encryption* key is explicitly or implicitly equivalent to knowing the *decryption* key.

A **asymmetric** or **public-key** cipher is one in which the *encryption* key is effectively public knowledge, without giving any useful information about the *decryption* key.

Until 30 years ago all ciphers were private-key.

The very possibility of public-key crypto did not exist until the secret work of some British CESG-at-GCHQ people Ellis-Cocks-Williamson in the 1960's, and public-domain work of Merkle, Diffie-Hellman, and Rivest-Shamir-Adleman in the 1970's.

Examples of symmetric/private-key ciphers

Cryptograms (substitution ciphers) [*broken: letter frequencies, small words*]

Anagrams (permutation ciphers) [*broken: double anagramming*]

Vigenère [*broken: Kasiski attack, Friedman attack*]

Enigma, Purple [*broken: key distribution problems, too small keyspace*]

DES [*broken: too small keyspace*]

3DES [*slow*]

Blowfish [*in use*], Arcfour [*in use*], TEA, IDEA

Serpent, Twofish, RC6, MARS [*AES finalists*]

AES (Rijndael)

Examples of asymmetric/public-key ciphers

RSA (Rivest-Shamir-Adlemen)

ElGamal

Elliptic curve cipher
($\sim$ abstracted ElGamal)

Knapsack ciphers [*discredited*]

Coding-theory ciphers [*out of fashion...*]

NTRU

Arithmetica (word-problem ciphers)

RSA overview

One-time preparation: *Alice* chooses two large random primes p, q, from 10^{100} to 10^{600} depending on the desired security. She computes the **RSA modulus** $n = p \cdot q$. She chooses encryption exponent e (often $e = 3$), and computes the multiplicative inverse d of e modulo $(p-1)(q-1)$. She publishes n, e (on her web page?) and keeps d secret. Primes p and q are thrown away.

Encryption: Bob wishes to encrypt a *plaintext* message x and send it to Alice on an insecure channel. Suppose $1 < x < n$ for simplicity. Bob computes and transmits $y = x^e \,\%\, n$.

Decryption: When Alice receives the ciphertext y, she computes $y^d \,\%\, n$, which *is* the plaintext x.

Why is this ok?

Why is it feasible for Alice to find two primes $\sim 10^{200}$ or so?

Why is it feasible for Alice to compute $e^{-1} \,\%\, (p-1)(q-1)$?

Why is it feasible for Bob to compute $x^e \,\%\, n$?

Why is it feasible for Alice to compute $y^d \,\%\, n$?

Why is $y^d \,\%\, n = x$?

Why is it *not* feasible for Eve (the eavesdropper) to compute d from n and e?

Why is it *not* feasible for Eve to compute x from $x^e \,\%\, n$?

How do we get a good supply of *random numbers*?

Minor qualifications about RSA

Want p and q equal to 3 mod 4.

In fact, maybe want p and q to be **strong primes**, namely so that $p-1$ and $q-1$ are not exclusively composed of small prime factors.

Want to be sure that e is relatively prime to $(p-1)(q-1)$: if we want $e=3$ or some other pre-specified number, must tweak p and q. Otherwise, tweak e.

Very unlikely that $\gcd(x,n)>1$, so ignore this.

Need **good-quality randomization** for choice of p and q. Else potential for *catastrophic failure.* (Related recent examples in software implementations of various security protocols.)

Diffie-Hellman Key Exchange

Alice and Bob have never met, and can only communicate across an insecure channel on which Eve is eavesdropping.

Eve has considerably greater computational power than Alice and Bob, and hears everything they say to each other.

Yet Alice and Bob can establish a **shared secret** which Eve cannot also acquire (assuming the difficulty of computing discrete logs).

The shared secret is then typically used as a *key* for a symmetric/private-key cipher to encrypt a subsequent conversation.

Alice and Bob agree on a large random prime p ($\sim 10^{130}$ or larger) and a random base g in the range $1 < g < p$. Alice secretly chooses a random a in the range $1 < a < p$ and computes $A = g^a \,\%\, p$. Similarly, Bob secretly chooses a random b in the range $1 < b < p$ and computes $B = g^b \,\%\, p$. Alice sends A over the channel, and Bob sends B over the channel.

So Alice knows p, g, a, A, B, Bob knows p, g, A, b, B, and Eve knows p, g, A, B.

Alice computes

$$K_A = B^a \,\%\, p$$

and Bob computes

$$K_B = A^b \,\%\, p$$

Since

$$K_A = K_B \,\%\, p$$

Alice and Bob now have a shared secret which it is *infeasible* for Eve to obtain.

Why do Alice and Bob get the same secret K?

This is a corollary of so-called *Laws of Exponents*, mediated by the good interaction of reduction modulo p with arithmetic operations.

Alice's secret value is a, Bob's is b.

Alice publishes $A = g^a \,\%\, p$, Bob publishes $B = g^b \,\%\, p$.

Then

$$\text{Alice's computation} = B^a \,\%\, p$$

$$= (g^b \,\%\, p)^a \,\%\, p = (g^b)^a \,\%\, p = g^{ba} \,\%\, p$$

$$= (g^a)^b \,\%\, p = (g^a \,\%\, p)^b \,\%\, p$$

$$= A^b \,\%\, p = \text{Bob's computation}$$

Thus, Alice and Bob *do* have a common value to be used as a key.

Why is this ok?

Why is it feasible for Alice and Bob to find a random $g \sim 10^{200}$?

Why is it feasible for Alice and Bob to find a random *prime* $\sim 10^{200}$?

Why is it feasible for Alice and Bob to acquire good-quality random numbers $a, b \sim 10^{200}$?

Why is it feasible for Alice to compute $g^a \,\%\, p$ with $a \sim 10^{200}$? (And similarly for Bob.)

Why is it *not* feasible for Eve (the eavesdropper) to compute a from $A = g^a \,\%\, p$ nor b from $B = g^b \,\%\, p$?

How does one get a good supply of random numbers?

Ingredient:
Fast exponentiation algorithm
(also called *square-and-multiply*)

To compute $b^n \,\%\, n$, with $n \sim 10^{100}$ or larger, do **not** multiply 10^{100} times.

Rather, note that **repeated squaring** reduces the number of operations:

$$b^{69} = b^{2^6+2^2+2^0} = (((((b^2)^2)^2)^2)^2)^2 \cdot (b^2)^2 \cdot b$$

To compute $x^e \,\%\, n$

initialize $(X, E, Y) = (x, e, 1)$
while $E > 0$
    if E is even
        replace X by $X^2 \,\%\, n$
        replace E by $E/2$
    elsif E is odd
        replace Y by $X \cdot Y \,\%\, n$
        replace E by $E - 1$

The final value of Y is $x^e \,\%\, n$.

For example, to compute $2^{17} \,\%\, 29$,
Initialize $(X, E, Y) = (2, 17, 1)$

With $(X, E, Y) = (2, 17, 1)$, $E = 17$ is odd

replace $E = 17$ by $17 - 1 = 16$

replace $Y = 1$ by $X * Y \,\%\, 29 = 2$

$E = 16$ is even

replace $E = 16$ by $16/2 = 8$

replace $X = 2$ by $X * X \,\%\, 29 = 4$

$E = 8$ is even

replace $E = 8$ by $8/2 = 4$

replace $X = 4$ by $X * X \,\%\, 29 = 16$

$E = 4$ is even

replace $E = 4$ by $4/2 = 2$

replace $X = 16$ by $X * X \,\%\, 29 = 24$

$E = 2$ is even

replace $E = 2$ by $2/2 = 1$

replace $X = 24$ by $X * X \,\%\, 29 = 25$

With $(X, E, Y) = (25, 1, 2)$, $E = 1$ is odd

replace $E = 1$ by $1 - 1 = 0$

replace $Y = 2$ by $X * Y \,\%\, 29 = 21$

Now $(X, E, Y) = (25, 0, 21)$, $E = 0$, so

$2^{17} \,\%\, 29 =$ current value $Y = 21$

Ingredient: Euclidean Algorithm

To compute *gcd*'s, and to compute $e^{-1} \,\%\, m$, use the familiar Euclidean algorithm. To compute $\gcd(x, y)$ takes at most $2 \log_2 y$ steps, if $x \geq y$.

To compute $\gcd(x, y)$:

Initialize $X = x$, $Y = y$, $R = X \,\%\, Y$
while $R > 0$
replace X by Y
replace Y by R
replace R by $X \,\%\, Y$
When $R = 0$, $Y = \gcd(x, y)$

This gives the familiar pattern: for example to compute $\gcd(1477, 721)$:

$1477 - 2 \cdot 721 = 35$
$721 - 20 \cdot 35 = 21$
$35 - 1 \cdot 21 = 14$
$21 - 1 \cdot 14 = 7$
$14 - 2 \cdot 7 = 0$

And 7 is the *gcd*.

Multiplicative inverses via Euclid

To compute $e^{-1} \,\%\, x$ with $\gcd(e, x) = 1$, minimizing memory use, rewrite each of the steps in the previous as

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & -q \\ 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \end{pmatrix} = \begin{pmatrix} \text{new } X \\ \text{new } Y \end{pmatrix}$$

where $R = X - qY$ with $|R| < Y$.

Thus, we obtain an integral matrix $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ with determinant ± 1 such that

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ e \end{pmatrix} = \begin{pmatrix} \gcd(x, e) \\ 0 \end{pmatrix}$$

When $\gcd(x, e) = 1$, we have

$$ax + be = 1$$

and thus

$$b = e^{-1} \,\%\, x$$

Ingredient: Euler's Theorem

Let $\varphi(n)$ be Euler's *totient function*, which counts the integers ℓ in the range $1 \leq \ell \leq n$ which are relatively prime to n.

Theorem: For $\gcd(x, n)=1$, $x^{\varphi(n)}=1\%n$.

(This is an immediate corollary of Lagrange's theorem, from *group theory*, applied to the group $\mathbf{Z}/n^{\times}$.)

This proves that **RSA decryption works**, using $\varphi(pq) = (p-1)(q-1)$: with $y = x^e \,\%\, n$, letting $ed = 1+M\varphi(n)$, all equalities modulo n,

$$y^d = (x^e)^d = x^{1+M\cdot\varphi(n)}$$

$$= x \cdot (x^{\varphi(n)})^M \,\%\, n = x \cdot 1^M = x$$

Ingredient: Infeasibility of factoring n

No proof exists that factoring is hard, but there is much practical evidence.

Several decades of new insights into factoring have yielded very clever factorization algorithms, which are *sub-exponential*, but still *super-polynomial*.

If large quantum computers ever exist, Shor's quantum factoring algorithm will break RSA. Some public-key systems (e.g., lattice-based ciphers such as NTRU) are not known to have fast quantum algorithms to break them.

With quantum computers, Grover's $\sqrt{n}$-time quantum search of n unordered things would require increased key size for nearly all ciphers.

Unauthorized computation of RSA decryption exponent?

For an eavesdropper to compute the RSA decryption exponent

$$d = e^{-1} \bmod (p-1)(q-1)$$

it would suffice (since the Euclid algorithm computes inverses quickly) to know the quantity $m = (p-1)(q-1)$.

But observe that knowledge of the RSA modulus $n = pq$ and of $m = (p-1)(q-1)$ would amount to knowing the factors p, q, since the roots of the polynomial equation

$$X^2 + (m - n + 2)X + n = 0$$

are p, q.

Presumably factoring is hard, so this is evidence (?) that an eavesdropper will not find a trick whereby to obtain $(p-1)(q-1)$, *from which to obtain the decryption exponent.*

Ingredient: big primes?

To acquire **200-digit prime numbers**, trial division would not succeed in the lifetime of the universe using all the computational power of the internet.

Trial division confirms that a number is prime by **failing to factor it**.

It turns out that **primality testing is much easier than factoring**.

Factoring big numbers is hard, despite striking (and wacky) modern factorization techniques much better than trial division.

Even more surprising are fast modern **probabilistic primality tests**.

(And, one can *construct* large primes with accompanying **certificates of primality** indicating how to reprove their primality upon demand.)

Failure of trial division:

Trial division attempts to divide a given number N by integers from 2 up through $\sqrt{N}$. Either we find a proper factor of N, or N is prime. (If N has a proper factor ℓ larger than $\sqrt{N}$, then $N/\ell \leq \sqrt{N}$.) The extreme case takes roughly $\sqrt{N}$ steps, or at least $\sqrt{N}/\ln N$.

If $N \sim 10^{200}$ is prime, or if it is the product of two primes each $\sim 10^{100}$, then it will take about 10^{100} trial divisions to discover this. Even if we're clever, it will take more than 10^{98} trial divisions.

If we could do 10^{12} trials per second, and if there were a 10^{12} hosts on the internet, with $< 10^8$ seconds per year, a massively parallel trial division would take ...

$$10^{66} \text{ years}$$

Examples of trial division

What are the practical limitations of trial division? On a 2.5 Gigahertz machine, code in C++ using GMP

1002904102901 has factor 1001401

('instantaneous')

100001220001957 has factor 10000019

(3 seconds)

10000013000000861 has factor 100000007

(27 seconds)

1000000110000000721 has factor 1000000007

(4 minutes)

Nowhere near 10^{200} ...

What were the previous examples?

By *other* means, *not trial division* I generated pairs of primes p, q of whatever size I wanted, and then tried to factor the product $p \cdot q$ by trial division.

These *other means* include the **Fermat pseudoprime** test, and the **Miller-Rabin** test for **strong pseudoprimes**.

These pseudoprime tests simply tell whether the test number n is *composite* or *probably prime*.

A pseudoprime test may tell that the test number n is *composite*, but **will not tell a proper factor of it**.

Facts about primes

The number $\pi(N)$ of primes less than N is

$$\pi(N) \sim \frac{N}{\log N}$$

This is the *Prime Number Theorem* (Hadamard and de la Vallée Poussin, 1896).

Riemann observed (1858) that *if* all the complex zeros of the **zeta function** $\zeta(s) = \sum n^{-s}$ lay on the line $\mathrm{Re}(s) = \frac{1}{2}$ *then* (as refined...)

$$\pi(N) = \frac{N}{\log N} + O(\sqrt{N}\,\log N)$$

The conjecture on the location of the zeros is the **Riemann Hypothesis**.

No result approaching this is known: there is *no* known zero-free region $\mathrm{Re}(s) \geq \sigma$ for $\sigma < 1$.

The Prime Number Theorem uses the non-vanishing of $\zeta(s)$ on $\mathrm{Re}(s) = 1$.

Hunting for primes

Nevertheless, when developing expectations for hunting for primes, we pretend that primes are distributed as evenly as possible.

Note: it is *not true* that primes are distributed evenly, even under the Riemann Hypothesis.

But if primes *were* evenly distributed, then near x primes would be about $\ln x$ apart.

Thus, in hunting for primes near x expect to examine $\frac{1}{2}\ln x$ candidates:

For $x \sim 10^{20}$ we have $\frac{1}{2}\ln x \sim 23$

For $x \sim 10^{100}$ we have $\frac{1}{2}\ln x \sim 115$

For $x \sim 10^{500}$ we have $\frac{1}{2}\ln x \sim 575$

Fermat pseudoprime test

Bargain-basement pseudoprime test

Fermat's Little theorem: If p is prime, then for any integer $1 < b < p$

$$b^p \,\%\, p = b$$

Thus, if n is an integer and $b^n \,\%\, n \neq b$ for some b, then n is **composite**.

The *converse* is false, but not *very* false...

Thus, we have

Converse-with-disclaimer: If $b^p \,\%\, p = b$ then p is *fairly likely* to be prime, but may not be.

Failure rate of Fermat pseudoprime test

The only *non-prime* $n < 5000$ with $2^n = 2 \bmod n$ are 341 561 645 1105 1387 1729 1905 2047 2465 2701 2821 3277 4033 4369 4371 4681

Requiring also $3^n \,\%\, n = 3$ leaves 561 1105 1729 2465 2701 2821

Requiring also $5^n \,\%\, n = 5$ leaves 561 1105 1729 2465 2821

Compared with 669 primes under 5000, this is a *false positive* failure rate of less than 1%.

n is a **Fermat pseudoprime base** b if $b^n \,\%\, n = b$.

Terminology

Usage is not consistent.

My usage is that a number that has passed a primality test (Fermat, Miller-Rabin, etc.) is a **pseudoprime**.

Sometimes a *pseudoprime* is meant to be a *non-prime* which has nevertheless passed a primality test such as Fermat. But for large numbers which have passed pseudoprimality tests we may never know *for sure* whether or not they're prime or composite ...

Another usage is to call a number that has passed a test a **probable prime**.

But this is dangerously close to **provable prime**, which is sometimes used to describe primes with accompanying certificates of their primality.

There are only 172 non-prime Fermat pseudoprimes base 2 under 500,000 versus 41,538 primes, a false positive rate of less than 0.41%

There are only 49 non-prime Fermat pseudoprimes base 2 and 3 under 500,000, a false positive rate of less than 0.118%

There are only 32 non-prime Fermat pseudoprimes base 2, 3, 5 under 500,000

There are still 32 non-prime Fermat pseudoprimes base 2, 3, 5, 7, 11, 13, 17 under 500,000

561 1105 1729 2465 2821 6601 8911 10585 15841 29341 41041 46657 52633 62745 63973 75361 101101 115921 126217 162401 172081 188461 252601 278545 294409 314821 334153 340561 399001 410041 449065 488881

Adding more such requirements does not shrink these lists further.

n is a **Carmichael number** if it is a *non-prime* Fermat pseudoprime to *every* base b.

In 1994 Alford, Granville, and Pomerance showed that there are infinitely-many Carmichael numbers.

And it appears that among *large* numbers Carmichael numbers become more common.

Nevertheless, the Fermat test is a very fast way to test for *compositeness*, and is so easy and cheap that it is still the best first approximation to primality.

It is **cheap** because $b^n \,\%\, n$ can be computed in $\sim \log n$ steps, not n...

Better primality test: Miller-Rabin (1978)

If $n = r \cdot s$ is composite (with $\gcd(r, s) = 1$) then by Sun-Ze's theorem there are at least 4 solutions to

$$x^2 = 1 \bmod n$$

namely the 4 choices of sign in

$$x = \pm 1 \bmod r \quad x = \pm 1 \bmod s$$

Thus, if we find $b \neq \pm 1 \bmod n$ such that $b^2 = 1 \bmod n$, n is definitely *not* composite.

Roughly, the **Miller-Rabin test** looks for such extra square roots of 1 modulo n (details below).

Theorem: (Miller-Rabin) For composite n, at least $3/4$ of b in the range $1 < b < n$ will detect the compositeness (via the Miller-Rabin test)

Pseudo-corollary If n passes the Miller-Rabin test with k random bases b, then (*exercise: explain the fallacy*)

$$\text{probability}(n \text{ is prime}) \geq 1 - \left(\frac{1}{4}\right)^k$$

Miller-Rabin test base b:
factor $n - 1 = 2^s \cdot m$ with m odd
replace b by $b^m \bmod n$
    if $b = \pm 1 \bmod n$ **stop:** n is 3/4 prime
    else continue
set $r = 0$
while $r < s$
    replace b by $b^2 \bmod n$
    if $b = -1 \bmod n$ **stop:** n is 3/4 prime
    elsif $b = +1 \bmod n$ **stop:** n is composite
    else continue
    replace r by $r + 1$
if we fall out of the loop, n is composite.

If n passes this test it is a
strong pseudoprime base b.

Failure rate of Miller-Rabin?

The fraction of b's which detect compositeness is apparently much greater than 3/4. For $n = 21311$ the detection rate is 0.9976. For 64777 the detection rate is 0.99972. For 1112927 the detection rate is 0.9999973

For $n < 50,000$ there are only 9 non-prime strong pseudoprimes base 2, namely 2047 3277 4033 4681 8321 15841 29341 42799 49141

For $n < 500,000$ there are only 33 non-prime strong pseudoprimes base 2.

For $n < 500,000$ there are *no* non-prime strong pseudoprimes base 2 and 3

For $100,000,000 < n < 101,000,000$ there are 3 strong pseudoprimes base 2 whose compositeness is detected base 3, namely 100463443 100618933 100943201

Some big strong pseudoprimes

Not trial division, but instead primality testing Fermat base 2, Miller-Rabin base 2, 3, 5, to find next prime after...

('instantaneous')
First prime after 10^{21} is $10^{21} + 117$

('instantaneous')
First prime after 10^{50} is $10^{50} + 151$

('hint of time taken')
First prime after 10^{100} is $10^{100} + 267$

(3 seconds)
First prime after 10^{200} is $10^{200} + 357$

(8 seconds)
First prime after 10^{300} is $10^{300} + 331$

(97 seconds)
First prime after 10^{1000} is $10^{1000} + 453$

More issues

Random numbers?

Relative failure of modern *factorization* attacks?