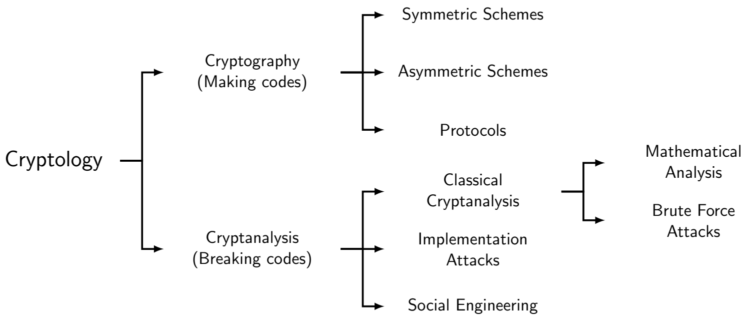
Cryptography, Number Theory, and RSA

Slides by Joan Boyar
IMADA
University of Southern Denmark
(with edits by Rolf Fagerberg)

Outline of lectures

- Cryptography vs. Cryptanalysis
- Symmetric key cryptography
- Public key cryptography
- Recap of number theory
- RSA
- Digital signatures with RSA
- Combining symmetric and public key systems
- Modular exponentiation
- Greatest common divisor
- Primality testing
- Correctness of RSA

Cryptography vs. Cryptanalysis



Symmetric key cryptography

Alice and Bob share a *single* secret key SK.

For Alice to send message m to Bob in encrypted form, Alice computes:
$c = E(m, SK)$.

To decrypt c, Bob computes:
$r = D(c, SK)$.

Of course, $r = m$ must be guaranteed by the pair of functions E and D constituting the cryptosystem.

Example of a symmetric key system: Caesar cipher

Idea: shift cyclically all letters of the alphabet by the same amount. The secret key SK is the shift. For SK = 3, encryption is given by the following table (A becomes D, B becomes E, etc.):

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

P	Q	R	S	T	U	V	W	X	Y	Z	Æ	Ø	Å
15	16	17	18	19	20	21	22	23	24	25	26	27	28
S	T	U	V	W	X	Y	Z	Æ	Ø	Å	A	B	C
18	19	20	21	22	23	24	25	26	27	28	0	1	2

Example of a symmetric key system: Caesar cipher

Suppose the following was encrypted using a Caesar cipher and the Danish alphabet. The key is unknown. How would you try to decrypt it?

ZQOØQOØ, RI.

Example of a symmetric key system: Caesar cipher

Suppose the following was encrypted using a Caesar cipher and the Danish alphabet. The key is unknown. How would you try to decrypt it?

ZQOØQOØ, RI.

What does this say about how many keys should be possible?

Some symmetric key systems

- ~~Caesar Cipher (< 100 BC, unknown)~~
- ...
- ...
- ~~Enigma (1930-40, German army)~~
- ~~DES (1976, IBM)~~
- ~~Triple DES (1978-81, Walter Tuchman, Ralph Merkle and Martin Hellman)~~
- IDEA (1991, James Massey, Xuejia Lai)
- Blowfish (1993, Bruce Schneider)
- AES (2001, Joan Daemen and Vincent Rijmen)

Crossed out systems are considered broken by now.

Public key cryptography [Hellman, Diffie, Merkle, 1976]

Bob — *two* keys: PK_B,SK_B

PK_B — Bob's public key
SK_B — Bob's private (secret) key

For Alice to send message m to Bob, Alice computes:
$c = E(m, PK_B)$.

To decrypt c, Bob computes:
$r = D(c, SK_B)$.

Of course, $r = m$ must be guaranteed by the pair of functions E and D constituting the cryptosystem.

It must also be "hard" to compute SK_B from PK_B.

[Public key cryptography is also called asymmetric key cryptography.]

Recap of Number Theory

Definition. Suppose $a, b \in \mathbb{Z}$, $a > 0$.
The terminology/notation below all mean the same, namely that $\exists c \in \mathbb{Z}$ such that $b = ac$.

- *a divides b*
- $a \mid b$
- a is a *factor* of b
- b is a *multiple* of a

The notation $e \nmid f$ means e does not divide f.

Theorem. $a, b, c \in \mathbb{Z}$. Then

1. if $a|b$ and $a|c$, then $a|(b + c)$
2. if $a|b$, then $a|bc \ \forall c \in \mathbb{Z}$
3. if $a|b$ and $b|c$, then $a|c$.

Recap of Number Theory

Definition. For $p \in \mathbb{Z}$, $p > 1$ we say that

- p is *prime* if 1 and p are the only positive integers which divide p.
  $2, 3, 5, 7, 11, 13, 17, \ldots$
- p is *composite* if it is not prime.
  $4, 6, 8, 9, 10, 12, 14, 15, 16, \ldots$

Recap of Number Theory

Theorem. $a \in \mathbb{Z}$, $d \in \mathbb{N}$
$\exists$ unique q, r, $0 \leq r < d$ such that $a = dq + r$

d – divisor
a – dividend
q – quotient
r – remainder $=$ a mod d

Definition. $gcd(a, b) =$ greatest common divisor of a and b
$=$ largest $d \in \mathbb{Z}$ such that $d|a$ and $d|b$

If $gcd(a, b) = 1$, then a and b are *relatively prime*.

Recap of Number Theory

Definition. $a \equiv b \pmod{m}$ — a is congruent to b modulo m if $m \mid (a - b)$.

$m \mid (a - b) \;\Leftrightarrow\; \exists k \in \mathbb{Z}$ such that $a = b + km$.

Theorem. $a \equiv b \pmod{m} \qquad c \equiv d \pmod{m}$
Then $a + c \equiv b + d \pmod{m}$ and $ac \equiv bd \pmod{m}$.

Proof.(of first) $\exists k_1, k_2$ such that
$a = b + k_1 m \qquad c = d + k_2 m$

$$\begin{aligned} a + c &= b + k_1 m + d + k_2 m \\ &= b + d + (k_1 + k_2)m \end{aligned} \qquad \square$$

Recap of Number Theory

Definition. $a \equiv b \pmod{m}$ — a is congruent to b modulo m if $m \mid (a - b)$.

$m \mid (a - b) \;\Rightarrow\; \exists k \in \mathbb{Z}$ such that $a = b + km$.

Examples.

1. $15 \equiv 22 \pmod{7}$?
2. $15 \equiv 1 \pmod{7}$?
3. $15 \equiv 37 \pmod{7}$?
4. $58 \equiv 22 \pmod{9}$?

Recap of Number Theory

Definition. $a \equiv b \pmod{m}$ — a is congruent to b modulo m if $m \mid (a - b)$.

$m \mid (a-b) \;\Rightarrow\; \exists k \in \mathbb{Z}$ such that $a = b + km$.

Examples.

1. $15 \equiv 22 \pmod{7}$?
2. $15 \equiv 1 \pmod{7}$?
3. $15 \equiv 37 \pmod{7}$?
4. $58 \equiv 22 \pmod{9}$?

Note the difference to:

1. $15 = 1 \bmod 7$?
2. $1 = 15 \bmod 7$?

RSA — a public key system [Rivest, Shamir, Adleman, 1977]

Choose two primes p, q. Set $N = p \cdot q$.
Find $e > 1$ such that $gcd(e, (p-1)(q-1)) = 1$.
Find d such that $e \cdot d \equiv 1 \pmod{(p-1)(q-1)}$.

- $PK = (N, e)$
- $SK = (N, d)$

To encrypt: $c = E(m, PK) = m^e \pmod{N}$.
To decrypt: $r = D(c, SK) = c^d \pmod{N}$.

One can prove that $r = m$ (if $0 \leq m < N$).

Here, m is the message, c is the cryptotext (the encrypted message). Note: any message of less than $\log_2 N$ bits can be seen as a binary number m with $0 \leq m < N$. For longer messages, chop it up and encrypt each part individually.

RSA, an example

Recap:
Choose two primes p, q. Set $N = p \cdot q$.
Find $e > 1$ such that $gcd(e, (p-1)(q-1)) = 1$.
Find d such that $e \cdot d \equiv 1 \pmod{(p-1)(q-1)}$.

- $PK = (N, e)$
- $SK = (N, d)$

To encrypt: $c = E(m, PK) = m^e \pmod{N}$.
To decrypt: $r = D(c, SK) = c^d \pmod{N}$.

Example:
$p = 5$, $q = 11$ (hence $N = 55$), $e = 3$, $d = 27$, $m = 8$.
Then $gcd(e, (p-1)(q-1)) = gcd(3, 4 \cdot 10) = 1$, as required.
Then $e \cdot d = 81$, so $e \cdot d \equiv 1 \pmod{4 \cdot 10}$, as required.
To encrypt m: $c = 8^3 \pmod{55} \ = \ 17$.
To decrypt c: $r = 17^{27} \pmod{55} \ = \ 8$.

RSA, one example more

Recap:
$N = p \cdot q$, where p, q prime.
$gcd(e, (p-1)(q-1)) = 1$.
$e \cdot d \equiv 1 \pmod{(p-1)(q-1)}$.

- $PK = (N, e)$
- $SK = (N, d)$

To encrypt: $c = E(m, PK) = m^e \pmod{N}$.
To decrypt: $r = D(c, PK) = c^d \pmod{N}$.

Try using $N = 35$, $e = 11$ as keys.
Factor 35 and check the requirement on e.
What is d? Try $d = 11$ and check the requirement on d.
Encrypt $m = 4$. Decrypt the result.

RSA, one example more

Recap:
$N = p \cdot q$, where p, q prime.
$gcd(e, (p-1)(q-1)) = 1$.
$e \cdot d \equiv 1 \pmod{(p-1)(q-1)}$.

- $PK = (N, e)$
- $SK = (N, d)$

To encrypt: $c = E(m, PK) = m^e \pmod{N}$.
To decrypt: $r = D(c, PK) = c^d \pmod{N}$.

Try using $N = 35$, $e = 11$ as keys.
Factor 35 and check the requirement on e.
What is d? Try $d = 11$ and check the requirement on d.
Encrypt $m = 4$. Decrypt the result.

Did you get $c = 9$? And $r = 4$?

An Application: Digital Signatures with RSA

Suppose Alice wants to sign a document m such that:

- No one else could forge her signature
- It is easy for others to verify her signature

Note m has arbitrary length.
RSA is used on fixed length messages.
Alice uses a cryptographically secure hash function h, meaning that:

- For any message m', $h(m')$ has a fixed length (e.g., 2048 bits)
- It is conjectured "hard" for anyone to find 2 messages (m_1, m_2) such that $h(m_1) = h(m_2)$.

Digital Signatures with RSA

Then Alice "decrypts" $h(m)$ with her secret RSA key (N_A, d_A)

$$s = (h(m))^{d_A} \pmod{N_A}$$

and publishes the document m and the signature s.

Bob verifies her signature using her public RSA key (N_A, e_A) and h:

$$c = s^{e_A} \pmod{N_A}$$

He accepts if and only if

$$h(m) = c$$

. This works because $s^{e_A} \pmod{N_A} =$

$$((h(m))^{d_A})^{e_A} \pmod{N_A} = ((h(m))^{e_A})^{d_A} \pmod{N_A} = h(m).$$

Combining symmetric and public key systems

Problem: Public key systems are slower in practice than symmetric key systems.

Combining symmetric and public key systems

Problem: Public key systems are slower in practice than symmetric key systems.

Solution: Use a symmetric key system for large messages.
For each such symmetric key system session, create a new key for it and start by sending this key *encrypted with a public key system*.

Combining symmetric and public key systems

Problem: Public key systems are slower in practice than symmetric key systems.

Solution: Use a symmetric key system for large messages.
For each such symmetric key system session, create a new key for it and start by sending this key *encrypted with a public key system*.

I.e., to encrypt a message m to send to Bob:

- Choose a random *session key* k for a symmetric key system (e.g., AES)
- Encrypt k with Bob's public key — Result k_e
- Encrypt m with k — Result m_e
- Send k_e and m_e to Bob

Combining symmetric and public key systems

Problem: Public key systems are slower in practice than symmetric key systems.

Solution: Use a symmetric key system for large messages.
For each such symmetric key system session, create a new key for it and start by sending this key *encrypted with a public key system*.

I.e., to encrypt a message m to send to Bob:

- Choose a random *session key* k for a symmetric key system (e.g., AES)
- Encrypt k with Bob's public key — Result k_e
- Encrypt m with k — Result m_e
- Send k_e and m_e to Bob

How does Bob decrypt? Why is this efficient?

Security of RSA

The primes p and q are kept secret along with d.

Suppose Eve can factor N.

Then she can find p and q (as these are the factors of N). From them and e, she can find d (using the same method as Alice, to be described later).

Then she can decrypt just like Alice!

So factoring must be hard, or RSA will be insecure (here, hard means very time-consuming).

Also, N must be sufficiently big to use hardness of factoring. Current recommendations are to choose p and q with at least 1024 bits each, making $N = p \cdot q$ have at least 2048 bits.

Factoring (naive approach)

Theorem N composite $\Rightarrow$ N has a prime divisor $\leq \sqrt{N}$

Factor(N)

for $i = 2$ to $\sqrt{N}$
check if i divides N
if it does **then** output divisor i and stop
output "Prime" if divisor not found

Corollary There is an algorithm for factoring N (or verifying primality) which does $O(\sqrt{N})$ tests of divisibility.

Complexity of factoring

Assume that we use primes which are at least 1024 bits long (the current recommendation for RSA). The naive approach does up to $\sqrt{N} = \sqrt{2^{2048}} = (2^{2048})^{1/2} = 2^{2048/2} = 2^{1024} = (10^{\log_{10} 2})^{1024}$ $= (10^{0.301\ldots})^{1024} = (10^{1024 \cdot 0.301\ldots}) > 10^{308}$ tests of divisibility.

This is 10^{291} years of CPU time (assuming 10^9 tests per second). Even having one CPU per human available, this would take more than 10^{281} years (while the universe is only around 10^{10} years old).

Complexity of factoring

Assume that we use primes which are at least 1024 bits long (the current recommendation for RSA). The naive approach does up to $\sqrt{N} = \sqrt{2^{2048}} = (2^{2048})^{1/2} = 2^{2048/2} = 2^{1024} = (10^{\log_{10} 2})^{1024}$ $= (10^{0.301\ldots})^{1024} = (10^{1024 \cdot 0.301\ldots}) > 10^{308}$ tests of divisibility.

This is 10^{291} years of CPU time (assuming 10^9 tests per second). Even having one CPU per human available, this would take more than 10^{281} years (while the universe is only around 10^{10} years old).

The input length in bits is $n = \log_2(N)$. So the running time above is $O(\sqrt{N}) = O(\sqrt{2^n}) = O((2^n)^{1/2}) = O(2^{n/2}) = O((2^{1/2})^n) =$ $O((\sqrt{2})^n) = O((1.4142\ldots)^n)$. This is *exponential* in n.

Complexity of factoring

Assume that we use primes which are at least 1024 bits long (the current recommendation for RSA). The naive approach does up to $\sqrt{N} = \sqrt{2^{2048}} = (2^{2048})^{1/2} = 2^{2048/2} = 2^{1024} = (10^{\log_{10} 2})^{1024}$ $= (10^{0.301\ldots})^{1024} = (10^{1024 \cdot 0.301\ldots}) > 10^{308}$ tests of divisibility.

This is 10^{291} years of CPU time (assuming 10^9 tests per second). Even having one CPU per human available, this would take more than 10^{281} years (while the universe is only around 10^{10} years old).

The input length in bits is $n = \log_2(N)$. So the running time above is $O(\sqrt{N}) = O(\sqrt{2^n}) = O((2^n)^{1/2}) = O(2^{n/2}) = O((2^{1/2})^n) = O((\sqrt{2})^n) = O((1.4142\ldots)^n)$. This is *exponential* in n.

Open Problem: Does there exist a factoring algorithm with running time *polynomial* in n?

(Note: if large enough quantum computers can be built, the answer is yes [Peter Shor, 1994].)

RSA, implementation details

How do we implement RSA?

- We need to find p, q
- We need to find e, d
- To encrypt and decrypt, we need to compute $a^k \pmod{n}$ for large a, k, and n

We will now discuss how this is done (going backwards through the list of tasks).

RSA — encryption/decryption, space usage

Computing $a^k \pmod{n}$:

$e \cdot d \equiv 1 \pmod{(p-1)(q-1)}$.
p and q have ≥ 1024 bits each (using current recommendations).
So at least one of e and d has ≥ 1024 bits.

We want to compute $a^k \pmod{n}$ for $a = m$, $n = N$ and $k = d, e$.
The message m usually has the same number of bits as $N = p \cdot q$, which is at least 2048 bits.

Hence, we are dealing with a value of a^k on the order of $(2^{2048})^{2^{1024}}$ which has $\log_2((2^{2048})^{2^{1024}}) = 2^{1024} \log_2(2^{2048}) = 2^{1024} \cdot 2048$ bits, which is more than 10^{311} bits. This number cannot be stored even if using all the RAM existing in the world.

Keeping sizes of numbers down

Theorem [From DM549]

For all nonnegative integers, b, c, m:

$$b \cdot c \pmod{n} = (b \pmod{n}) \cdot (c \pmod{n}) \pmod{n}$$

Example: $a \cdot a^2 \pmod{n} = (a \pmod{n})(a^2 \pmod{n}) \pmod{n}$.

This allows us to take (mod n) after every multiplication without changing the result. This will ensure that all numbers dealt with have approximately the same number of bits as n (e.g., 2048 bits).

Space (RAM) problem solved.

A multiplication followed by (mod n) is called a modular multiplication.

RSA — encryption/decryption, time usage

We need to encrypt and decrypt: compute $a^k \pmod{n}$.

$a^2 \pmod{n} = a \cdot a \pmod{n}$: 1 modular multiplication
$a^3 \pmod{n} = a \cdot (a \cdot a \pmod{n}) \pmod{n}$: 2 mod mults
$\vdots$
$a^k \pmod{n}$: $k - 1$ modular multiplications.

This is way too many:

$e \cdot d \equiv 1 \pmod{(p-1)(q-1)}$.
p and q have ≥ 1024 bits each (using current recommendations).
So at least one of e and d has ≥ 1024 bits, and we have $k = d$ and $k = e$.

To either encrypt or decrypt would need $\geq 2^{1024} - 1 \approx 10^{308}$ operations. Much more time than the age of the universe.

Keeping time down

We need to encrypt and decrypt: compute $a^k \pmod{n}$.

$a^2 \pmod{n} = a \cdot a \pmod{n}$ — 1 modular multiplication
$a^3 \pmod{n} = a \cdot (a \cdot a \pmod{n}) \pmod{n}$ — 2 mod mults

How do you calculate $a^4 \pmod{n}$ in less than 3 mod mults?

Keeping time down

We need to encrypt and decrypt: compute $a^k \pmod{n}$.

$a^2 \pmod{n} = a \cdot a \pmod{n}$ — 1 modular multiplication
$a^3 \pmod{n} = a \cdot (a \cdot a \pmod{n}) \pmod{n}$ — 2 mod mults

How do you calculate $a^4 \pmod{n}$ in less than 3 mod mults?

Observation:
$a^4 \pmod{n} = (a^2 \pmod{n})^2 \pmod{n}$ — 2 mod mults

Keeping time down

We need to encrypt and decrypt: compute $a^k \pmod{n}$.

$a^2 \pmod{n} = a \cdot a \pmod{n}$ — 1 modular multiplication
$a^3 \pmod{n} = a \cdot (a \cdot a \pmod{n}) \pmod{n}$ — 2 mod mults

How do you calculate $a^4 \pmod{n}$ in less than 3 mod mults?

Observation:
$a^4 \pmod{n} = (a^2 \pmod{n})^2 \pmod{n}$ — 2 mod mults

In general: $a^{2s} \pmod{n}$?

Keeping time down

We need to encrypt and decrypt: compute $a^k \pmod{n}$.

$a^2 \pmod{n} = a \cdot a \pmod{n}$ — 1 modular multiplication
$a^3 \pmod{n} = a \cdot (a \cdot a \pmod{n}) \pmod{n}$ — 2 mod mults

How do you calculate $a^4 \pmod{n}$ in less than 3 mod mults?

Observation:
$a^4 \pmod{n} = (a^2 \pmod{n})^2 \pmod{n}$ — 2 mod mults

In general: $a^{2s} \pmod{n}$?

$$a^{2s} \pmod{n} = (a^s \pmod{n})^2 \pmod{n}$$

Keeping time down

We need to encrypt and decrypt: compute $a^k \pmod{n}$.

$a^2 \pmod{n} = a \cdot a \pmod{n}$ — 1 modular multiplication
$a^3 \pmod{n} = a \cdot (a \cdot a \pmod{n}) \pmod{n}$ — 2 mod mults

How do you calculate $a^4 \pmod{n}$ in less than 3 mod mults?

Observation:
$a^4 \pmod{n} = (a^2 \pmod{n})^2 \pmod{n}$ — 2 mod mults

In general: $a^{2s} \pmod{n}$?

$$a^{2s} \pmod{n} = (a^s \pmod{n})^2 \pmod{n}$$

In general: $a^{2s+1} \pmod{n}$?

Keeping time down

We need to encrypt and decrypt: compute $a^k \pmod{n}$.

$a^2 \pmod{n} = a \cdot a \pmod{n}$ — 1 modular multiplication
$a^3 \pmod{n} = a \cdot (a \cdot a \pmod{n}) \pmod{n}$ — 2 mod mults

How do you calculate $a^4 \pmod{n}$ in less than 3 mod mults?

Observation:
$a^4 \pmod{n} = (a^2 \pmod{n})^2 \pmod{n}$ — 2 mod mults

In general: $a^{2s} \pmod{n}$?

$$a^{2s} \pmod{n} = (a^s \pmod{n})^2 \pmod{n}$$

In general: $a^{2s+1} \pmod{n}$?

$$a^{2s+1} \pmod{n} = a \cdot (a^{2s} \pmod{n}) \pmod{n}$$

Modular Exponentiation

Resulting algorithm:

Exp(a, k, n) $\{$ Compute $a^k \pmod n$ $\}$

if $k < 0$ **then** report error
if $k = 0$ **then** return(1)
if $k = 1$ **then** return($a \pmod n$)
if k is odd **then** return($a \cdot$Exp$(a, k-1, n) \pmod n$)
if k is even **then**
    $c =$ Exp$(a, k/2, n)$
    return($(c \cdot c) \pmod n$)

Modular Exponentiation

Resulting algorithm:

Exp(a, k, n) { Compute $a^k \pmod{n}$ }

if $k < 0$ **then** report error
if $k = 0$ **then** return(1)
if $k = 1$ **then** return($a \pmod{n}$)
if k is odd **then** return($a \cdot$Exp$(a, k-1, n) \pmod{n}$)
if k is even **then**
  $c =$ Exp$(a, k/2, n)$
  return($(c \cdot c) \pmod{n}$)

How many modular multiplications?

Modular Exponentiation

Resulting algorithm:

Exp(a, k, n) { Compute $a^k \pmod{n}$ }

if $k < 0$ **then** report error
if $k = 0$ **then** return(1)
if $k = 1$ **then** return($a \pmod{n}$)
if k is odd **then** return($a \cdot$Exp$(a, k-1, n) \pmod{n}$)
if k is even **then**
    $c =$ Exp$(a, k/2, n)$
    return($(c \cdot c) \pmod{n}$)

How many modular multiplications?

We divide exponent by 2 every other time. How many times can we do that?

Modular Exponentiation

Resulting algorithm:

$\text{Exp}(a, k, n) \quad \{ \text{ Compute } a^k \pmod{n} \}$

if $k < 0$ **then** report error
if $k = 0$ **then** return(1)
if $k = 1$ **then** return($a \pmod{n}$)
if k is odd **then** return($a \cdot \text{Exp}(a, k-1, n) \pmod{n}$)
if k is even **then**
    $c = \text{Exp}(a, k/2, n)$
    return($(c \cdot c) \pmod{n}$)

How many modular multiplications?

We divide exponent by 2 every other time. How many times can we do that? $\lfloor \log_2(k) \rfloor$.

So at most $2\lfloor \log_2(k) \rfloor$ modular multiplications in total. Time problem solved, since $2\lfloor \log_2(k) \rfloor \approx 2 \log_2(2^{1024}) = 2 \cdot 1024 = 2048$.

Finding e and d

We need to find: e, d.
$gcd(e, (p-1)(q-1)) = 1$.
$e \cdot d \equiv 1 \pmod{(p-1)(q-1)}$.

Finding e and d

We need to find: e, d.
$gcd(e, (p-1)(q-1)) = 1$.
$e \cdot d \equiv 1 \pmod{(p-1)(q-1)}$.

Choose random e.
Check that $gcd(e, (p-1)(q-1)) = 1$. If not, repeat.

Find d such that $e \cdot d \equiv 1 \pmod{(p-1)(q-1)}$.

Finding multiplicative inverses modulo n

Finding **multiplicative inverses** modulo n:

Given e and n, find d such that $e \cdot d \equiv 1 \pmod{n}$.

Solved if we can find s and t fulfilling $s \cdot e + t \cdot n = 1$ (we can then use s as our d).

This can be done via the Extended Euclidean Algorithm if $gcd(e, n) = 1$.

Finding multiplicative inverses modulo n

Finding **multiplicative inverses** modulo n:

Given e and n, find d such that $e \cdot d \equiv 1 \pmod{n}$.

Solved if we can find s and t fulfilling $s \cdot e + t \cdot n = 1$ (we can then use s as our d).

This can be done via the Extended Euclidean Algorithm if $gcd(e, n) = 1$.

The Euclidean Algorithm finds $gcd(a, b)$ for positive integers a and b.

The Extended Euclidean Algorithm also finds integers s and t such that $s \cdot a + t \cdot b = gcd(a, b)$.

Finding multiplicative inverses modulo n

Finding **multiplicative inverses** modulo n:

Given e and n, find d such that $e \cdot d \equiv 1 \pmod{n}$.

Solved if we can find s and t fulfilling $s \cdot e + t \cdot n = 1$ (we can then use s as our d).

This can be done via the Extended Euclidean Algorithm if $gcd(e, n) = 1$.

The Euclidean Algorithm finds $gcd(a, b)$ for positive integers a and b.

The Extended Euclidean Algorithm also finds integers s and t such that $s \cdot a + t \cdot b = gcd(a, b)$.

The (Extended) Euclidean Algorithm is fast: it runs in time $\log(\max(a, b))$ [Lamé, 1844].

The extended Euclidean algorithm (two-pass)

Euclidean algorithm by example: Find $\gcd(75, 42)$

$$
\begin{aligned}
d_0 &= 75 & & \\
d_1 &= 42 & & (75 = 1 \cdot 42 + 33) \\
d_2 &= 33 & & (42 = 1 \cdot 33 + 9) \\
d_3 &= 9 & & (33 = 3 \cdot 9 + 6) \\
d_4 &= 6 & & (9 = 1 \cdot 6 + 3) \\
d_5 &= 3 & & (6 = 2 \cdot 3 + 0) \\
d_6 &= 0 & & \text{Stop and return previous } d \text{ (here } d_5\text{) as gcd.}
\end{aligned}
$$

The extended Euclidean algorithm (two-pass)

Euclidean algorithm by example: Find $\gcd(75, 42)$

$$
\begin{aligned}
d_0 &= 75 \\
d_1 &= 42 && (75 = 1 \cdot 42 + 33) \\
d_2 &= 33 && (42 = 1 \cdot 33 + 9) \\
d_3 &= 9 && (33 = 3 \cdot 9 + 6) \\
d_4 &= 6 && (9 = 1 \cdot 6 + 3) \\
d_5 &= 3 && (6 = 2 \cdot 3 + 0) \\
d_6 &= 0 && \text{Stop and return previous } d \text{ (here } d_5\text{) as gcd.}
\end{aligned}
$$

Extension: Find s and t using the equations from bottom to top:

$$
\begin{gathered}
\gcd(75, 42) = 3 \\
= 9 - 6 = 9 - (33 - 3 \cdot 9) = -33 + 4 \cdot 9 \\
= -33 + 4 \cdot (42 - 33) = 4 \cdot 42 - 5 \cdot 33 = 4 \cdot 42 - 5 \cdot (75 - 42) \\
= -5 \cdot 75 + 9 \cdot 42 = s \cdot 75 + t \cdot 42
\end{gathered}
$$

The extended Euclidean algorithm (single pass)

{ Initialize}

$d_0 = b \qquad s_0 = 0 \qquad t_0 = 1$

$d_1 = a \qquad s_1 = 1 \qquad t_1 = 0$

$i = 1$

{ Compute next d}

while $d_i > 0$ **do**

begin

$i = i + 1$

{ Compute $d_i = d_{i-2} \pmod{d_{i-1}}$}

$q_i = \lfloor d_{i-2}/d_{i-1} \rfloor$

$d_i = d_{i-2} - q_i d_{i-1}$

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

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

end

return: $gcd(b, a) = d_{i-1}, \qquad s = s_{i-1}, \qquad t = t_{i-1}$

The extended Euclidean algorithm (single pass)

The single pass algorithm maintains the following invariant (from which correctness of s and t follows):

$$d_i = s_i a + t_i b$$

This invariant is proved by induction on i:

Initialization ($i = 0$, $i = 1$):

$$d_0 = b = 0 \cdot a + 1 \cdot b = s_0 a + t_0 b$$
$$d_1 = a = 1 \cdot a + 0 \cdot b = s_1 a + t_1 b$$

Step ($i \geq 2$):

$$\begin{aligned} d_i = d_{i-2} - q_i d_{i-1} &= (s_{i-2} a + t_{i-2} b) - q_i (s_{i-1} a + t_{i-1} b) \\ &= (s_{i-2} - q_i s_{i-1}) a + (t_{i-2} - q_i t_{i-1}) b = s_i a + t_i b \end{aligned}$$

Examples

Calculate the following:

1. $gcd(6, 9)$
2. s and t such that $s \cdot 6 + t \cdot 9 = gcd(6, 9)$
3. $gcd(15, 23)$
4. s and t such that $s \cdot 15 + t \cdot 23 = gcd(15, 23)$

Primality testing

We also need to find the large primes p, q.

Plan: Choose numbers at random and check if they are prime.

Questions

1. How many random integers with 1024 bits are prime?

Questions

1. How many random integers with 1024 bits are prime?

Prime Number Theorem: about $\frac{x}{\ln x}$ numbers $< x$ are prime

Questions

1. How many random integers with 1024 bits are prime?

Prime Number Theorem: about $\frac{x}{\ln x}$ numbers $< x$ are prime

In our situation we have $x = 2^{1024}$. As $\ln 2^{1024} = 1024 \cdot \ln 2 \approx 710$, the average distance between primes with (up to) 1024 bits is around 710.

Questions

1. How many random integers with 1024 bits are prime?

Prime Number Theorem: about $\frac{x}{\ln x}$ numbers $< x$ are prime

In our situation we have $x = 2^{1024}$. As $\ln 2^{1024} = 1024 \cdot \ln 2 \approx 710$, the average distance between primes with (up to) 1024 bits is around 710.

Hence, we can expect to test about 710 random numbers with 1024 bits before finding a prime number.

(This holds because if the probability of "success" is p, the expected number of tries until the first "success" is $1/p$.)

Questions

1. How many random integers with 1024 bits are prime?

Prime Number Theorem: about $\frac{x}{\ln x}$ numbers $< x$ are prime

In our situation we have $x = 2^{1024}$. As $\ln 2^{1024} = 1024 \cdot \ln 2 \approx 710$, the average distance between primes with (up to) 1024 bits is around 710.

Hence, we can expect to test about 710 random numbers with 1024 bits before finding a prime number.

(This holds because if the probability of "success" is p, the expected number of tries until the first "success" is $1/p$.)

2. How fast can we test if a number is prime?

Questions

1. How many random integers with 1024 bits are prime?

Prime Number Theorem: about $\frac{x}{\ln x}$ numbers $< x$ are prime

In our situation we have $x = 2^{1024}$. As $\ln 2^{1024} = 1024 \cdot \ln 2 \approx 710$, the average distance between primes with (up to) 1024 bits is around 710.

Hence, we can expect to test about 710 random numbers with 1024 bits before finding a prime number.

(This holds because if the probability of "success" is p, the expected number of tries until the first "success" is $1/p$.)

2. How fast can we test if a number is prime?

Quite fast, it turns out (in practice using randomness). See the following pages.

Suggestion 1

Sieve of Eratosthenes:

Lists:

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

Suggestion 1

Sieve of Eratosthenes:

Lists:

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	3		5		7		9		11		13		15		17		19

Suggestion 1

Sieve of Eratosthenes:
Lists:

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	3		5		7		9		11		13		15		17		19
			5		7				11		13				17		19

Suggestion 1

Sieve of Eratosthenes:

Lists:

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	3		5		7		9		11		13		15		17		19
			5		7				11		13				17		19
					7				11		13				17		19

Suggestion 1

Sieve of Eratosthenes:
Lists:

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	3		5		7		9		11		13		15		17		19
			5		7				11		13				17		19
					7				11		13				17		19

We need to go up to $2^{1024} = 10^{308}$ — more than the number of atoms in universe.

So we cannot even write out the first list in the sieve of Eratosthenes. Not practical.

Suggestion 2

Use our naive factoring algorithm:

```
CheckPrime(n)
for i = 2 to √n do
    check if i divides n
    if it does then return(Composite)
return(Prime)
```

As we saw earlier, this takes much more time than the age of the universe. Not practical.

Rabin–Miller Primality Testing

This is a practical, randomized primality test.

Starting point:

Fermat's Little Theorem: Suppose n is a prime. Then for all $1 \le a \le n-1$, $a^{n-1} \pmod{n} = 1$.

A Fermat test based on $a = 2$:

$2^{14} \pmod{15} \equiv 4 \neq 1$.
So 15 is not prime.

We say that 2 is a Fermat witness that 15 is composite.

Rabin–Miller Primality Test

First attempt: repeat Fermat test with many a's:

Prime(n)

repeat k times
    Choose random a with $1 \leq a \leq n-1$
    if $a^{n-1} \pmod{n} \not\equiv 1$ **then** return(Composite)
return(Probably Prime)

Rabin–Miller Primality Test

First attempt: repeat Fermat test with many a's:

Prime(n)

repeat k times

Choose random a with $1 \leq a \leq n-1$

if $a^{n-1} \pmod{n} \not\equiv 1$ **then** return(Composite)

return(Probably Prime)

Unfortunately not efficient on all numbers. E.g. not on Carmichael Numbers: a composite n, where for all a with $1 \leq a \leq n-1$ and a relatively prime to n, we have $a^{n-1} \pmod{n} \equiv 1$. Example of a Carmichael number: $561 = 3 \cdot 11 \cdot 17$

Rabin–Miller Primality Test

First attempt: repeat Fermat test with many a's:

Prime(n)

repeat k times

Choose random a with $1 \leq a \leq n-1$

if $a^{n-1} \pmod{n} \not\equiv 1$ **then** return(Composite)

return(Probably Prime)

Unfortunately not efficient on all numbers. E.g. not on Carmichael Numbers: a composite n, where for all a with $1 \leq a \leq n-1$ and a relatively prime to n, we have $a^{n-1} \pmod{n} \equiv 1$. Example of a Carmichael number: $561 = 3 \cdot 11 \cdot 17$

Add to the picture this **Theorem:**
If n is prime, then $x^2 \pmod{n} \equiv 1$ implies $x \pmod{n} \in \{1, n-1\}$.
If n is composite, odd, and has two distinct factors, then $x^2 \pmod{n} \equiv 1$ implies at least four values possible for $x \pmod{n}$.

Example: $x^2 \pmod{15} \equiv 1 \Rightarrow x \in \{1, 4, 11, 14\}$

Rabin–Miller Primality Test

Idea: Start with Fermat test for some a. Then "take square roots of 1 (mod n)" as long as we have an x with $x^2 \pmod{n} \equiv 1$.

Rabin–Miller Primality Test

Idea: Start with Fermat test for some a. Then "take square roots of 1 (mod n)" as long as we have an x with $x^2 \pmod{n} \equiv 1$.
Example with $n = 561$ and $a = 50$:

$50^{560} \pmod{561} \equiv 1$ [i.e., $(50^{280})^2 \pmod{561} \equiv 1$ (so $x = 50^{280}$)]
$50^{280} \pmod{561} \equiv 1$ [i.e., $(50^{140})^2 \pmod{561} \equiv 1$ (so $x = 50^{140}$)]
$50^{140} \pmod{561} \equiv 1$ $\vdots$
$50^{70} \pmod{561} \equiv 1$
$50^{35} \pmod{561} \equiv 560$ [Process stops (35 is odd and $560 \neq 1$).]

If n is prime, we can only end in $\equiv 1$ or $\equiv n-1$ (for all a). Above, this also happened for the composite $n = 561$ when $a = 50$.

Rabin–Miller Primality Test

Idea: Start with Fermat test for some a. Then "take square roots of 1 (mod n)" as long as we have an x with $x^2 \pmod{n} \equiv 1$.
Example with $n = 561$ and $a = 50$:

$50^{560} \pmod{561} \equiv 1$ [i.e., $(50^{280})^2 \pmod{561} \equiv 1$ (so $x = 50^{280}$)]
$50^{280} \pmod{561} \equiv 1$ [i.e., $(50^{140})^2 \pmod{561} \equiv 1$ (so $x = 50^{140}$)]
$50^{140} \pmod{561} \equiv 1$ $\vdots$
$50^{70} \pmod{561} \equiv 1$
$50^{35} \pmod{561} \equiv 560$ [Process stops (35 is odd and $560 \neq 1$).]

If n is prime, we can only end in $\equiv 1$ or $\equiv n - 1$ (for all a). Above, this also happened for the composite $n = 561$ when $a = 50$.
Let's now try $a = 2$:

$2^{560} \pmod{561} \equiv 1$
$2^{280} \pmod{561} \equiv 1$
$2^{140} \pmod{561} \equiv 67$ [Not just 1 or $n - 1$. Busted!]

We say that 2 is a Rabin–Miller witness that 561 is composite.

Rabin–Miller Primality Test

Resulting algorithm:

Miller–Rabin(n, k)

Calculate odd m such that $n - 1 = 2^s \cdot m$
repeat k times
  Choose random a with $1 \leq a \leq n - 1$
  if $a^{n-1} \pmod{n} \not\equiv 1$ **then** return(Composite)
  if $a^{(n-1)/2} \pmod{n} \equiv n - 1$ **then** continue [⇒ next iteration]
  if $a^{(n-1)/2} \pmod{n} \not\equiv 1$ **then** return(Composite)
  if $a^{(n-1)/4} \pmod{n} \equiv n - 1$ **then** continue [⇒ next iteration]
  if $a^{(n-1)/4} \pmod{n} \not\equiv 1$ **then** return(Composite)
    ⋮
  if $a^{m} \pmod{n} \equiv n - 1$ **then** continue [⇒ next iteration]
  if $a^{m} \pmod{n} \not\equiv 1$ **then** return(Composite)
end repeat
return(Probably Prime)

Rabin–Miller Primality Test

Theorem: If n is composite and odd, at most $1/4$ of the a's with $1 \leq a \leq n-1$ will not end in "return(Composite)" during an iteration of the **repeat**-loop.

This means that with k iterations, a composite odd n will survive to return(Probably Prime) (making the algorithm return a wrong answer) with probability at most $(1/4)^k$. Otherwise, the algorithm returns the correct answer "Composite". Even numbers are always composite, so we don't need to test them.

For e.g. $k = 100$, the probability of a wrong answer is therefore less than $(1/4)^{100} = 1/2^{200} < 1/10^{60}$.

A prime n will always survive to "return(Probably Prime)", which is the correct answer.

Conclusions about primality testing

1. Miller–Rabin is a practical, randomized primality test
2. In 2002, a deterministic primality test was given [Agrawal, Kayal, Saxena]. It is less practical, though.
3. Randomized algorithms may be prefered over deterministic ones, even if they (with very low probability) can make errors.
4. Number theory has practical uses.

Why does RSA work?

Ingredient 1:

Thm (**The Chinese Remainder Theorem**) Let $n_1, n_2, ..., n_k$ be pairwise relatively prime. For any integers $x_1, x_2, ..., x_k$, there exists $x \in \mathbb{Z}$ s.t. $x \equiv x_i \pmod{n_i}$ for $1 \leq i \leq k$. Also, x is uniquely determined modulo the product $N = n_1 n_2 ... n_k$: If $x' \in \mathbb{Z}$ s.t. $x \equiv x_i \pmod{n_i}$ for $1 \leq i \leq k$, then $x \equiv x' \pmod{N}$.

We for RSA consider the special case where $n_1 = p$ and $n_2 = q$ are two primes (hence $N = pq$), and where $x_1 = x_2 = m$.

Clearly, $m \equiv m \pmod{p}$ and $m \equiv m \pmod{q}$ for any m. So if x fulfills $x \equiv m \pmod{p}$ and $x \equiv m \pmod{q}$, then $x \equiv m \pmod{N}$.

In particular, if $0 \leq x, m \leq N - 1$, then we must have $x = m$.

Why does RSA work?

Ingredient 2:

Fermat's Little Theorem: If p is a prime, $p \nmid a$, then

$$a^{p-1} \equiv 1 \pmod{p} \text{ and } a^p \equiv a \pmod{p}$$

RSA

Recap:

$N = p \cdot q$, where p, q prime.
$gcd(e, (p-1)(q-1)) = 1$.
$e \cdot d \equiv 1 \pmod{(p-1)(q-1)}$.

- $PK = (N, e)$
- $SK = (N, d)$

To encrypt: $c = E(m, PK) = m^e \pmod{N}$.
To decrypt: $r = D(c, PK) = c^d \pmod{N}$.

RSA

Recap:

$N = p \cdot q$, where p, q prime.
$gcd(e, (p-1)(q-1)) = 1$.
$e \cdot d \equiv 1 \pmod{(p-1)(q-1)}$.

- $PK = (N, e)$
- $SK = (N, d)$

To encrypt: $c = E(m, PK) = m^e \pmod{N}$.
To decrypt: $r = D(c, PK) = c^d \pmod{N}$.

We now show correctness of RSA, i.e., that $r = m$.

Correctness of RSA

Let $x = D(E(m, PK), SK)$. Then
$x = (m^e \pmod N)^d \pmod N = m^{ed} \pmod N$.

Recall that $\exists k$ s.t. $ed = 1 + k(p-1)(q-1)$.

If $p \not| m$ then by Fermat's little theorem:

$m^{ed} \equiv m^{1+k(p-1)(q-1)} \equiv m \cdot (m^{(p-1)})^{k(q-1)} \equiv m \cdot 1^{k(q-1)} \equiv m \pmod p$.

Similarly, if $q \not| m$:

$m^{ed} \equiv m^{1+k(p-1)(q-1)} \equiv m \cdot (m^{(q-1)})^{k(p-1)} \equiv m \cdot 1^{k(p-1)} \equiv m \pmod q$.

From the Chinese Remainder Theorem: $m^{ed} \equiv m \pmod N$.
Hence, $x = m^{ed} \pmod N = m \pmod N = m$, where the last equality holds if $0 \leq m < N$ (which we require in RSA).

Correctness of RSA

For the remaining cases: assume $p|m$

Then $m = pk$ for some k, so for any t we have $m^t = (pk)^t = pk'$ for some k'.

Hence, $m^{ed} \equiv 0 \equiv m \pmod{p}$.

If $q|m$, we can similarly show $m^{ed} \equiv 0 \equiv m \pmod{q}$.

Thus, in *all* cases, $m^{ed} \equiv m \pmod{p}$ and $m^{ed} \equiv m \pmod{q}$, so the Chinese Remainder Theorem gives that $m^{ed} \equiv m \pmod{N}$ and the argument at the bottom of the previous slide holds.