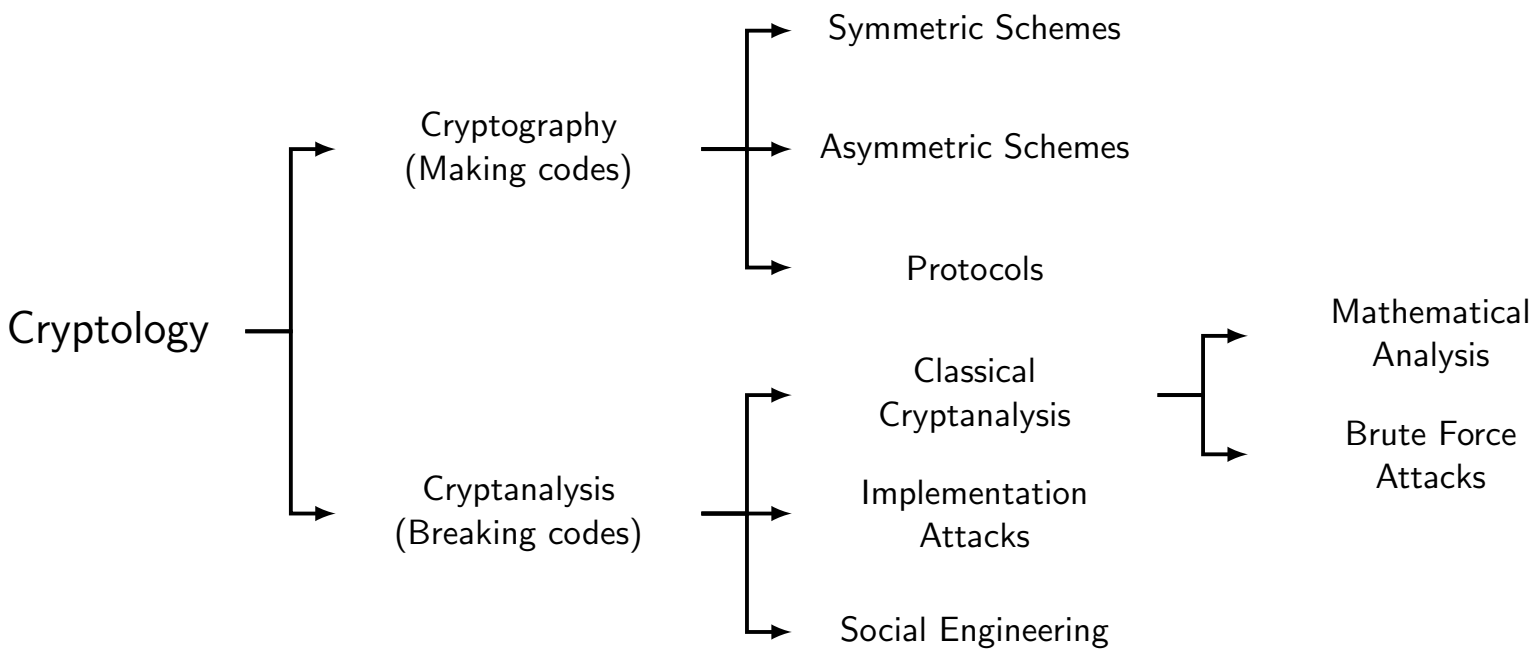
Cryptography, Number Theory, and RSA

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

Outline of lectures

- Cryptography vs. Cryptanalysis
- Symmetric key cryptography
- Public key cryptography
- Combining symmetric and public key systems
- Recap of number theory from DM549
- RSA
- Modular exponentiation
- Euclidean algorithm
- Primality testing
- Correctness of RSA
- Digital signatures with RSA

Cryptography vs. Cryptanalysis



Symmetric key cryptography

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

For Alice (or anyone else) to send message m to Bob in encrypted form, Alice computes:
$c = \text{Encrypt}(m, \text{SK})$.

To decrypt c, Bob computes:
$r = \text{Decrypt}(c, \text{SK})$.

Of course, $r = m$ must be guaranteed by the pair of functions Encrypt and Decrypt 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.

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 has **two** keys: PK, SK

- PK is Bob's public key
- SK is Bob's secret key

For Alice to send message m to Bob, Alice computes:
$c = \text{Encrypt}(m, \text{PK})$.

To decrypt c, Bob computes:
$r = \text{Decrypt}(c, \text{SK})$.

Of course, $r = m$ must be guaranteed by the pair of functions Encrypt() and Decrypt() constituting the cryptosystem.

It must also be computationally hard to compute SK from PK.

Core concept: Anyone can encode messages with the public key. Only persons knowing the private key are (expected to be) able to decode it.

Combining symmetric and public key systems

A number of public key systems have been developed. One of those is RSA, which is the subject of the rest of these slides.

Problem: Public key systems are usually 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 large 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 (call the result k_e).
- Encrypt m with k (call the result m_e).
- Send k_e and m_e to Bob.

Recap of number theory from DM549

Before describing RSA, we recap some simple notions from DM549.

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

- p is *prime* if 1 and p are the only positive integers that 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 from DM549

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

d: divisor
a: dividend
q: quotient $= \lfloor a/d \rfloor$
r: remainder $= a \bmod d$

Notation. We write $d|a$ if $r = 0$ in the theorem above.

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 called *relatively prime*.

Recap of number theory from DM549

Notation. $a \equiv b \pmod{m}$ means $a \bmod m = b \bmod m$.

We say "a is congruent to b modulo m".

Examples.

1. $15 \equiv 22 \pmod{7}$? Yes
2. $15 \equiv 1 \pmod{7}$? Yes
3. $15 \equiv 37 \pmod{7}$? No
4. $58 \equiv 22 \pmod{9}$? Yes

1. $15 = 1 \bmod 7$? No
2. $1 = 15 \bmod 7$? Yes

Theorem. If $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$ then

$$a + c \equiv b + d \pmod{m}$$

and

$$ac \equiv bd \pmod{m}.$$

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

Choose two primes p, q $(p \neq q)$.
Set $N = p \cdot q$ and $N' = (p-1) \cdot (q-1)$.
Find $e > 1$ such that $\gcd(e, N') = 1$.
Find d such that $e \cdot d \bmod N' = 1$.

- $\mathsf{PK} = (N, e)$
- $\mathsf{SK} = (N, d)$

$\text{Encrypt}(m, \mathsf{PK}) = m^e \bmod N = c$.
$\text{Decrypt}(c, \mathsf{SK}) = c^d \bmod N = r$.

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, example

RSA (recap):

Choose two primes p, q. Set $N = p \cdot q$ and $N' = (p-1) \cdot (q-1)$.
Find $e > 1$ such that $\gcd(e, N') = 1$.
Find d such that $e \cdot d \bmod N' = 1$.

- $\mathrm{PK} = (N, e)$
- $\mathrm{SK} = (N, d)$

$\mathrm{Encrypt}(m, \mathrm{PK}) = m^e \bmod N = c$.
$\mathrm{Decrypt}(c, \mathrm{SK}) = c^d \bmod N = r$.

Example:

$p = 5$, $q = 11$ (hence $N = 55$, $N' = 40$), $e = 3$, $d = 27$, $m = 8$.
Check: $\gcd(e, N') = \gcd(3, 40) = 1$, as required.
Check: $e \cdot d = 81$, so $e \cdot d \bmod 40 = 1$, as required.
To encrypt m: $c = 8^3 \bmod 55 = 17$.
To decrypt c: $r = 17^{27} \bmod 55 = 8 = m$.

RSA, exercise

RSA (recap):

Choose two primes p, q. Set $N = p \cdot q$ and $N' = (p-1) \cdot (q-1)$.
Find $e > 1$ such that $\gcd(e, N') = 1$.
Find d such that $e \cdot d \bmod N' = 1$.

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

Encrypt$(m, \text{PK}) = m^e \bmod N = c$.
Decrypt$(c, \text{SK}) = c^d \bmod N = r$.

Exercise:

Try using $N = 35$, $e = 11$ as the public key PK.
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 $r = 4$?

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 computationally hard, or RSA will be insecure.

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).

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 $a^k \bmod n$ 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

On computing $a^k \bmod n$: we have

$$e \cdot d \bmod N' = 1,$$

where $e > 1$ and where N' has ≥ 2048 bits (using current recommendations). So at least one of e and d has ≥ 1024 bits, and we want to compute both $a^e \bmod n$ and $a^d \bmod n$.

In more detail, we want to compute $a^k \bmod n$ for $a = m$, $n = N$ and $k = d, e$. The message m usually has the same number of bits as N, 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. Thus, the number a^k itself cannot be stored even if using all the RAM existing in the world. However, note that we want $a^k \bmod n$.

Keeping sizes of numbers down

Theorem [DM549]

For all nonnegative integers, b, c, n:

$$b \cdot c \bmod n = (b \bmod n) \cdot (c \bmod n) \bmod n$$

Example: $a^3 \bmod n = (a \bmod n) \cdot (a^2 \bmod n) \bmod n =$
$(a \bmod n) \cdot ((a \bmod n) \cdot (a \bmod n) \bmod n) \bmod n$.

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 (here 2048 bits).

Space (RAM) problem solved.

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

RSA encryption/decryption, time usage

To encrypt and decrypt, we need to compute $a^k \bmod n$.

$a \bmod n = a \bmod n$ [0 modular multiplications]
$a^2 \bmod n = (a \bmod n) \cdot (a \bmod n) \bmod n$ [1 mod mult]
$a^3 \bmod n = (a \bmod n) \cdot (a^2 \bmod n) \bmod n$ [2 mod mults]
$\vdots$
$a^k \bmod n$ [$k-1$ mod mults]

This is way too many:

$$e \cdot d \bmod N' = 1,$$

where $e > 1$ and where N' has ≥ 2048 bits (using current recommendations). So at least one of e and d has ≥ 1024 bits, and we want to compute both $a^e \bmod n$ and $a^d \bmod n$.

So to either encrypt or decrypt, the above method needs $\geq 2^{1024} - 1 \approx 10^{308}$ operations. More time than age of universe.

Keeping time down

To encrypt and decrypt, we need to compute $a^k \bmod n$.

$a^2 \bmod n = (a \bmod n) \cdot (a \bmod n) \bmod n$ [1 mod mult]
$a^3 \bmod n = (a \bmod n) \cdot (a^2 \bmod n) \bmod n$ [2 mod mults]

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

Observation:
$a^4 \bmod n = (a^2 \bmod n)^2 \bmod n$ [2 mod mults]

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

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

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

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

Modular exponentiation

Resulting algorithm:

$\mathrm{Exp}(a, k, n)$ $\quad$ { Compute $a^k \bmod n$ }

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

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 such that (with $N' = (p-1) \cdot (q-1)$):

- $\gcd(e, N') = 1$.
- $e \cdot d \bmod N' = 1$.

Method:

- Choose random e below N'.
- Check that $\gcd(e, N') = 1$. If not, repeat.

For the check, we can use the **Euclidean Algorithm** [DM549].

Number of repeats? If choice of e hits a prime number, then $\gcd(e, N') = 1$, unless N' is a multiple of e. The latter is very unlikely for random e. Hitting a prime number occurs fairly often (see later).

Finding d such that $e \cdot d \bmod N' = 1$ is called finding a **multiplicative inverse** modulo N' [DM549].

Finding multiplicative inverses modulo n

Finding **multiplicative inverses** modulo n:

Given e and n, find d such that $e \cdot d \bmod n = 1$.

Solved if we can find s and t fulfilling $s \cdot e + t \cdot n = 1$, as then $s \cdot e \equiv s \cdot e + t \cdot n = 1 \pmod{n}$, so we can then use s as our d.

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

Recall from DM549:

- 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 $s \cdot a + t \cdot b = \gcd(a, b)$.

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

The extended Euclidean algorithm [DM549]

Recap of **Euclidean algorithm by example**. Find $\gcd(75, 42)$:

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

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

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

Primality testing

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

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

Questions

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

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

In our situation we have $x = 2^{1024}$. As

$$\ln 2^{1024} = 1024 \cdot \ln 2 = 1024 \cdot 0.693\ldots \approx 710,$$

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.

Primality testing, first suggestion

Use our naive factoring algorithm:

CheckPrime(n)

for $i = 2$ to $\sqrt{n}$ **do**
    check if i divides n
    if it does **then return**(Composite)
return(Prime)

As we saw earlier, this in our setting takes much more time than the age of the universe.

Primality testing, second suggestion

Sieve of Eratosthenes: [År 300 f.v.t.]

Consider the list of all integers ≥ 2. Repeat: remove from the (remaining) list all multiples of the (current) first number.

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 $2^{1024} = 10^{308}$. This is more than the number of atoms in universe.

So in our setting we cannot even write out the first list in the sieve of Eratosthenes.

Third suggestion: Miller–Rabin primality test [1980]

This is a practical, randomized primality test.

Starting point:

Fermat's Little Theorem: Suppose n is a prime. Then for all integers a where $1 \leq a \leq n-1$, the following holds:

$$a^{n-1} \bmod n = 1.$$

A Fermat test based on $a = 2$:

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

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

Miller–Rabin primality test

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

Fermat(n)

repeat k times

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

if $a^{n-1} \bmod n \neq 1$ **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 $a^{n-1} \bmod n = 1$. This means that Carmichael numbers potentially can have very few Fermat witnesses a. Example of a Carmichael number: $561 = 3 \cdot 11 \cdot 17$

Now add to the picture this motivating **Theorem:**
If n is *prime*, then $x^2 \bmod n = 1$ implies $x \bmod n \in \{1, n-1\}$.
If n is *composite*, odd, and has two distinct prime factors, then $x^2 \bmod n = 1$ implies at least four different values for $x \bmod n$.

Examples: $x^2 \bmod 11 = 1 \Leftrightarrow x \bmod 11 \in \{1, 10\}$
$x^2 \bmod 15 = 1 \Leftrightarrow x \bmod 15 \in \{1, 4, 11, 14\}$

Miller–Rabin primality test

Final version: Start with Fermat test for some a. Then, as long we have an x with $x^2 \bmod n = 1$, we look at $x \bmod n$.
Example with $n = 561$ and $a = 50$:

$50^{560} \bmod 561 = 1$ [i.e., $(50^{280})^2 \bmod 561 = 1$ (so $x = 50^{280}$)]
$50^{280} \bmod 561 = 1$ [i.e., $(50^{140})^2 \bmod 561 = 1$ (so $x = 50^{140}$)]

$50^{140} \bmod 561 = 1$ ⋮
$50^{70} \bmod 561 = 1$
$50^{35} \bmod 561 = 560$ [The process stops]

[We must stop both because 35 is odd and because 560 is n-1, not 1).]

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

$2^{560} \bmod 561 = 1$
$2^{280} \bmod 561 = 1$
$2^{140} \bmod 561 = 67$ [Not 1 or $n - 1$. Busted, it's a composite!]

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

Miller–Rabin primality test

Resulting algorithm:

Miller–Rabin(n, k)

Calculate odd m such that $n - 1 = 2^s \cdot m$ [s = number of possible divisions by two]

repeat k times

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

if $a^{n-1} \bmod n \neq 1$ **then** return(Composite)

if $a^{(n-1)/2} \bmod n = n - 1$ **then** continue [$\Rightarrow$ next iteration]

if $a^{(n-1)/2} \bmod n \neq 1$ **then** return(Composite)

if $a^{(n-1)/4} \bmod n = n - 1$ **then** continue [$\Rightarrow$ next iteration]

if $a^{(n-1)/4} \bmod n \neq 1$ **then** return(Composite)

$\vdots$

if $a^m \bmod n = n - 1$ **then** continue [$\Rightarrow$ next iteration]

if $a^m \bmod n \neq 1$ **then** return(Composite)

end repeat

return(Probably Prime) [None of the k iterations returned "composite"]

Miller–Rabin 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 for an odd composite number is therefore less than $(1/4)^{100} = 1/4^{100}$ $= 1/(10^{\log_{10} 4})^{100} < 1/(10^{0.602})^{100} = 1/(10^{0.602 \cdot 100}) < 1/10^{60}$.

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:

Lemma: If some m' fulfills $m' \equiv m \pmod{p}$ and $m' \equiv m \pmod{q}$, then $m' \equiv m \pmod{N}$.

Proof: We have $p|(m' - m)$ and $q|(m' - m)$. Hence, both p and q are factors in the prime factorization of $m' - m$. Since p and q are primes and $p \neq q$, we have $(pq)|(m' - m)$. Since $N = pq$, this is the same as $m' \equiv m \pmod{N}$.

Why does RSA work?

Ingredient 2:

Fermat's Little Theorem: If p is a prime and a is any integer for which $p \not| a$, then

$$a^{p-1} \bmod p = 1$$

RSA

RSA (recap):

Choose two primes p, q. Set $N = p \cdot q$ and $N' = (p-1) \cdot (q-1)$.
Find $e > 1$ such that $\gcd(e, N') = 1$.
Find d such that $e \cdot d \bmod N' = 1$.

- $\mathsf{PK} = (N, e)$
- $\mathsf{SK} = (N, d)$

$\mathrm{Encrypt}(m, \mathsf{PK}) = m^e \bmod N = c$.
$\mathrm{Decrypt}(c, \mathsf{SK}) = c^d \bmod N = r$.

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

Correctness of RSA

Let $r = \text{Decrypt}(\text{Encrypt}(m, \text{PK}), \text{SK})$.

In other words, $r = (m^e \bmod N)^d \bmod N = m^{ed} \bmod N$.

Recall that $\exists k$ s.t. $ed = 1 + k(p-1)(q-1)$, by the choice of d.

Assume $p \not| m$. Then by Fermat's little theorem:

$$\begin{aligned} m^{ed} &= m^{1+k(p-1)(q-1)} = m \cdot \left(m^{(p-1)}\right)^{k(q-1)} \\ &\equiv m \cdot 1^{k(q-1)} = m \qquad \pmod{p} \end{aligned}$$

Similarly, if $q \not| m$:

$$\begin{aligned} m^{ed} &= m^{1+k(p-1)(q-1)} = m \cdot \left(m^{(q-1)}\right)^{k(p-1)} \\ &\equiv m \cdot 1^{k(p-1)} = m \qquad \pmod{q} \end{aligned}$$

From the lemma above with $m' = m^{ed}$ we get $m^{ed} \equiv m \pmod{N}$.

Hence, $r = m^{ed} \bmod N = m \bmod N = m$, 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, we have

$$m^{ed} \equiv m \pmod{p}$$
$$m^{ed} \equiv m \pmod{q}$$

and the argument at the bottom of the previous slide holds.

An application of RSA: digital signatures

Suppose Bob wants to sign a document m such that:

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

Note: m has arbitrary length, while RSA is used on fixed length messages.

Bob uses a so-called cryptographically secure hash function h, meaning that:

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

[Such hash functions have been developed, such as SHA-256. See more at `https://en.wikipedia.org/wiki/Cryptographic_hash_function`.]

Digital signatures with RSA

Then Bob "decrypts" $h(m)$ with his secret RSA key $SK = (N, d)$

$$s = (h(m))^d \bmod N$$

and publishes the document m and the signature s.

Alice verifies his signature using his public RSA key $PK = (N, e)$ and h:

$$c = s^e \bmod N$$

She also computes $h(m)$ and then accepts if and only if

$$h(m) = c$$

This works because $s^e \bmod N =$

$$((h(m))^d)^e \bmod N = ((h(m))^e)^d \bmod N = h(m).$$