Cryptography, Number Theory, and RSA

Joan Boyar, IMADA, University of Southern Denmark

November/December 2019

Outline

- Symmetric key cryptography
- Public key cryptography
- Introduction to number theory
- RSA
- Digital signatures with RSA
- Combining symmetric and public key systems
- Modular exponentiation
- Greatest common divisor
- Primality testing
- Correctness of RSA

Caesar cipher

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

$$E(m) = m + 3 \pmod{29}$$

Symmetric key systems

Suppose the following was encrypted using a Caesar cipher and the Danish alphabet. The key is unknown. What does it say?

ZQOØQOØ, RI.

Symmetric key systems

Suppose the following was encrypted using a Caesar cipher and the Danish alphabet. The key is unknown. What does it say?

ZQOØQOØ, RI.

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

Symmetric key systems

- Caesar Cipher
-
-
- Enigma
- DES
- Blowfish
- IDEA
- Triple DES
- AES

Public key cryptography

Bob — 2 keys -PK_B,SK_B

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

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

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

Public key cryptography

Bob — 2 keys -PK_B,SK_B

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

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

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

It must be "hard" to compute m from (c, PK_B).

Public key cryptography

Bob — 2 keys -PK_B,SK_B

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

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

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

It must be "hard" to compute m from (c, PK_B).
It must be "hard" to compute SK_B from PK_B.

Introduction to Number Theory

Definition. Suppose $a, b \in \mathbb{Z}$, $a > 0$.
Suppose $\exists c \in \mathbb{Z}$ s.t. $b = ac$. Then a *divides* b.
$a \mid b$.
a is a *factor* of b.
b is a *multiple* of a.
$e \not| f$ 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 $a|c$.

Definition. $p \in \mathbb{Z}$, $p > 1$.
p is *prime* if 1 and p are the only positive integers which divide p.
$2, 3, 5, 7, 11, 13, 17, \dots$
p is *composite* if it is not prime.
$4, 6, 8, 9, 10, 12, 14, 15, 16, \dots$

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

d – divisor
a – dividend
q – quotient
r – remainder $= a \bmod d$

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

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

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

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

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

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

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

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

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

Examples.

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

RSA — a public key system

$N_A = p_A \cdot q_A$, where p_A, q_A prime.
$gcd(e_A, (p_A - 1)(q_A - 1)) = 1$.
$e_A \cdot d_A \equiv 1 \pmod{(p_A - 1)(q_A - 1)}$.

- $PK_A = (N_A, e_A)$
- $SK_A = (N_A, d_A)$

To encrypt: $c = E(m, PK_A) = m^{e_A} \pmod{N_A}$.
To decrypt: $r = D(c, SK_A) = c^{d_A} \pmod{N_A}$.
$r = m$.

RSA — a public key system

$N_A = p_A \cdot q_A$, where p_A, q_A prime.
$gcd(e_A, (p_A - 1)(q_A - 1)) = 1$.
$e_A \cdot d_A \equiv 1 \pmod{(p_A - 1)(q_A - 1)}$.

- $PK_A = (N_A, e_A)$
- $SK_A = (N_A, d_A)$

To encrypt: $c = E(m, PK_A) = m^{e_A} \pmod{N_A}$.
To decrypt: $r = D(c, SK_A) = c^{d_A} \pmod{N_A}$.
$r = m$.

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

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, such that:

- For any message m', $h(m')$ has a fixed length (512 bits?)
- It is "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)

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

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 slow!

Combining symmetric and public key systems

Problem: Public key systems are slow!

Solution: Use symmetric key system for large message.
Encrypt only session key with public key system.

Combining symmetric and public key systems

Problem: Public key systems are slow!

Solution: Use symmetric key system for large message.
Encrypt only session key with public key system.

To encrypt a message m to send to Bob:

- Choose a random *session key* k for a symmetric key system (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 slow!

Solution: Use symmetric key system for large message.
Encrypt only session key with public key system.

To encrypt a message m to send to Bob:

- Choose a random *session key* k for a symmetric key system (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_A and q_A are kept secret with d_A.

Suppose Eve can factor N_A.

Then she can find p_A and q_A.
From them and e_A, she finds d_A.

Then she can decrypt just like Alice.

Factoring must be hard!

Factoring

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

Factor(N)

for $i = 2$ to $\sqrt{N}$ **do**
  check if i divides N
  if it does **then** output $(i, N/i)$
endfor
output -1 if divisor not found

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

Factoring

Check all possible divisors between 2 and $\sqrt{N}$.
Not finished in your grandchildren's life time for N with 3072 bits.

Problem The length of the input is $n = \lceil \log_2(N+1) \rceil$. So the running time is $O(2^{n/2})$ — exponential.

Open Problem Does there exist a polynomial time factoring algorithm?

Use primes which are at least 1024 (or 1536) bits long.
So $2^{1023} \leq p_A, q_A < 2^{1024}$.
So $p_A \approx 10^{308}$.

RSA

How do we implement RSA?

We need to find: p_A, q_A, N_A, e_A, d_A.
We need to encrypt and decrypt.

RSA — encryption/decryption

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

$a^2 \pmod{n} \equiv a \cdot a \pmod{n}$ — 1 modular multiplication

Modular Exponentiation

Theorem. For all nonnegative integers, b, c, m,
$b \cdot c \pmod{m} = (b \pmod{m}) \cdot (c \pmod{m}) \pmod{m}$.

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

$$
\begin{aligned}
8^3 \pmod{55} &= 8 \cdot 8^2 \pmod{55} \\
&= 8 \cdot 64 \pmod{55} \\
&= 8 \cdot (9 + 55) \pmod{55} \\
&= 72 + (8 \cdot 55) \pmod{55} \\
&= 17 + 55 + (8 \cdot 55) \pmod{55} \\
&= 17
\end{aligned}
$$

RSA — encryption/decryption

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

RSA — encryption/decryption

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

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

RSA — encryption/decryption

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

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

This is too many!
$e_A \cdot d_A \equiv 1 \pmod{(p_A - 1)(q_A - 1)}$.
p_A and q_A have ≥ 1024 bits each.
So at least one of e_A and d_A has ≥ 1024 bits.

RSA — encryption/decryption

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

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

This is too many!
$e_A \cdot d_A \equiv 1 \pmod{(p_A - 1)(q_A - 1)}$.
p_A and q_A have ≥ 1024 bits each.
So at least one of e_A and d_A has ≥ 1024 bits.

To either encrypt or decrypt would need $\geq 2^{1023} \approx 10^{308}$ operations (more than number of atoms in the universe).

RSA — encryption/decryption

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

$a^2 \pmod{n} \equiv a \cdot a \pmod{n}$ — 1 modular multiplication
$a^3 \pmod{n} \equiv a \cdot (a \cdot a \pmod{n}) \pmod{n}$ — 2 mod mults
How do you calculate $a^4 \pmod{n}$ in less than 3?

RSA — encryption/decryption

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

$a^2 \pmod{n} \equiv a \cdot a \pmod{n}$ — 1 modular multiplication
$a^3 \pmod{n} \equiv a \cdot (a \cdot a \pmod{n}) \pmod{n}$ — 2 mod mults
How do you calculate $a^4 \pmod{n}$ in less than 3?
$a^4 \pmod{n} \equiv (a^2 \pmod{n})^2 \pmod{n}$ — 2 mod mults

RSA — encryption/decryption

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

$a^2 \pmod{n} \equiv a \cdot a \pmod{n}$ — 1 modular multiplication
$a^3 \pmod{n} \equiv a \cdot (a \cdot a \pmod{n}) \pmod{n}$ — 2 mod mults
How do you calculate $a^4 \pmod{n}$ in less than 3?
$a^4 \pmod{n} \equiv (a^2 \pmod{n})^2 \pmod{n}$ — 2 mod mults
In general: $a^{2s} \pmod{n}$?

RSA — encryption/decryption

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

$a^2 \pmod{n} \equiv a \cdot a \pmod{n}$ — 1 modular multiplication
$a^3 \pmod{n} \equiv a \cdot (a \cdot a \pmod{n}) \pmod{n}$ — 2 mod mults
How do you calculate $a^4 \pmod{n}$ in less than 3?
$a^4 \pmod{n} \equiv (a^2 \pmod{n})^2 \pmod{n}$ — 2 mod mults
In general: $a^{2s} \pmod{n}$?
$a^{2s} \pmod{n} \equiv (a^s \pmod{n})^2 \pmod{n}$

RSA — encryption/decryption

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

$a^2 \pmod{n} \equiv a \cdot a \pmod{n}$ — 1 modular multiplication
$a^3 \pmod{n} \equiv a \cdot (a \cdot a \pmod{n}) \pmod{n}$ — 2 mod mults
How do you calculate $a^4 \pmod{n}$ in less than 3?
$a^4 \pmod{n} \equiv (a^2 \pmod{n})^2 \pmod{n}$ — 2 mod mults
$a^{2s} \pmod{n} \equiv (a^s \pmod{n})^2 \pmod{n}$
In general: $a^{2s+1} \pmod{n}$?

RSA — encryption/decryption

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

$a^2 \pmod{n} \equiv a \cdot a \pmod{n}$ — 1 modular multiplication
$a^3 \pmod{n} \equiv a \cdot (a \cdot a \pmod{n}) \pmod{n}$ — 2 mod mults
How do you calculate $a^4 \pmod{n}$ in less than 3?
$a^4 \pmod{n} \equiv (a^2 \pmod{n})^2 \pmod{n}$ — 2 mod mults
$a^{2s} \pmod{n} \equiv (a^s \pmod{n})^2 \pmod{n}$
$a^{2s+1} \pmod{n} \equiv a \cdot ((a^s \pmod{n})^2 \pmod{n}) \pmod{n}$

Modular Exponentiation

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**
    $c \leftarrow$Exp$(a, k/2, n)$
    return($(c \cdot c) \pmod{n}$)

Modular Exponentiation

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**
    $c \leftarrow$Exp$(a, k/2, n)$
    return($(c \cdot c) \pmod{n}$)

To compute $3^6 \pmod{7}$: Exp$(3, 6, 7)$
$c \leftarrow$Exp$(3, 3, 7)$

Modular Exponentiation

$\text{Exp}(a, k, n)$ { 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($a \cdot \text{Exp}(a, k-1, n) \pmod{n}$)
if k is even **then**
  $c \leftarrow \text{Exp}(a, k/2, n)$
  return($(c \cdot c) \pmod{n}$)

To compute $3^6 \pmod{7}$: $\text{Exp}(3, 6, 7)$
$c \leftarrow \text{Exp}(3, 3, 7) \leftarrow 3 \cdot (\text{Exp}(3, 2, 7) \pmod{7})$

Modular Exponentiation

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**
    $c \leftarrow$Exp$(a, k/2, n)$
    return($(c \cdot c) \pmod{n}$)

To compute $3^6 \pmod{7}$: Exp$(3, 6, 7)$
$c \leftarrow$Exp$(3, 3, 7) \leftarrow 3 \cdot (Exp(3, 2, 7)) \pmod{7})$
$c' \leftarrow$Exp$(3, 1, 7)$

Modular Exponentiation

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**
  $c \leftarrow$Exp$(a, k/2, n)$
  return($(c \cdot c) \pmod{n}$)

To compute $3^6 \pmod{7}$: Exp$(3, 6, 7)$
$c \leftarrow$Exp$(3, 3, 7) \leftarrow 3 \cdot (Exp(3, 2, 7)) \pmod{7})$
$c' \leftarrow$Exp$(3, 1, 7) \leftarrow 3$

Modular Exponentiation

$\text{Exp}(a, k, n)$ { 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($a \cdot \text{Exp}(a, k-1, n) \pmod{n}$)
if k is even **then**
  $c \leftarrow \text{Exp}(a, k/2, n)$
  return($(c \cdot c) \pmod{n}$)

To compute $3^6 \pmod{7}$: $\text{Exp}(3, 6, 7)$
$c \leftarrow \text{Exp}(3, 3, 7) \leftarrow 3 \cdot (\text{Exp}(3, 2, 7)) \pmod{7})$
$c' \leftarrow \text{Exp}(3, 1, 7) \leftarrow 3$
$\text{Exp}(3, 2, 7) \pmod{7}) \leftarrow 3 \cdot 3 \pmod{7} \leftarrow 2$

Modular Exponentiation

$\text{Exp}(a, k, n)$ { 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($a \cdot \text{Exp}(a, k-1, n) \pmod{n}$)
if k is even **then**
  $c \leftarrow \text{Exp}(a, k/2, n)$
  return($(c \cdot c) \pmod{n}$)

To compute $3^6 \pmod{7}$: $\text{Exp}(3, 6, 7)$
$c \leftarrow \text{Exp}(3, 3, 7) \leftarrow 3 \cdot (\text{Exp}(3, 2, 7)) \pmod{7})$
$c' \leftarrow \text{Exp}(3, 1, 7) \leftarrow 3$
$\text{Exp}(3, 2, 7) \pmod{7}) \leftarrow 3 \cdot 3 \pmod{7} \leftarrow 2$
$c \leftarrow 3 \cdot 2 \pmod{7} \leftarrow 6$

Modular Exponentiation

$\text{Exp}(a, k, n)$ { 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($a \cdot \text{Exp}(a, k-1, n) \pmod{n}$)
if k is even **then**
  $c \leftarrow \text{Exp}(a, k/2, n)$
  return($(c \cdot c) \pmod{n}$)

To compute $3^6 \pmod{7}$: $\text{Exp}(3, 6, 7)$
$c \leftarrow \text{Exp}(3, 3, 7) \leftarrow 3 \cdot (\text{Exp}(3, 2, 7)) \pmod{7})$
$c' \leftarrow \text{Exp}(3, 1, 7) \leftarrow 3$
$\text{Exp}(3, 2, 7) \pmod{7}) \leftarrow 3 \cdot 3 \pmod{7} \leftarrow 2$
$c \leftarrow 3 \cdot 2 \pmod{7} \leftarrow 6$
$\text{Exp}(3, 6, 7) \leftarrow (6 \cdot 6) \pmod{7} \leftarrow 1$

Modular Exponentiation

$\text{Exp}(a, k, n)$ $\{$ 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($a \cdot \text{Exp}(a, k-1, n) \pmod{n}$)
if k is even **then**
    $c \leftarrow \text{Exp}(a, k/2, n)$
    return($(c \cdot c) \pmod{n}$)

How many modular multiplications?

Modular Exponentiation

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**
  $c \leftarrow$Exp$(a, k/2, n)$
  return($(c \cdot c) \pmod{n}$)

How many modular multiplications?

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

Modular Exponentiation

```
Exp(a, k, n)     { Compute a^k (mod n) }

if k < 0 then report error
if k = 0 then return(1)
if k = 1 then return(a (mod n))
if k is odd then return(a·Exp(a, k − 1, n) (mod n))
if k is even then
      c ←Exp(a, k/2, n)
      return((c · c) (mod n))
```

How many modular multiplications?

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.

RSA — a public key system

$N_A = p_A \cdot q_A$, where p_A, q_A prime.
$gcd(e_A, (p_A - 1)(q_A - 1)) = 1$.
$e_A \cdot d_A \equiv 1 \pmod{(p_A - 1)(q_A - 1)}$.

- $PK_A = (N_A, e_A)$
- $SK_A = (N_A, d_A)$

To encrypt: $c = E(m, PK_A) = m^{e_A} \pmod{N_A}$.
To decrypt: $r = D(c, SK_A) = c^{d_A} \pmod{N_A}$.
$r = m$.

Try using $N = 35$, $e = 11$ to create keys for RSA.
What is d? Try $d = 11$ and check it.
Encrypt 4. Decrypt the result.

RSA — a public key system

$N_A = p_A \cdot q_A$, where p_A, q_A prime.
$gcd(e_A, (p_A - 1)(q_A - 1)) = 1$.
$e_A \cdot d_A \equiv 1 \pmod{(p_A - 1)(q_A - 1)}$.

- $PK_A = (N_A, e_A)$
- $SK_A = (N_A, d_A)$

To encrypt: $c = E(m, PK_A) = m^{e_A} \pmod{N_A}$.
To decrypt: $r = D(c, SK_A) = c^{d_A} \pmod{N_A}$.
$r = m$.

Try using $N = 35$, $e = 11$ to create keys for RSA.
What is d? Try $d = 11$ and check it.
Encrypt 4. Decrypt the result.
Did you get $c = 9$? And $r = 4$?

RSA

$N_A = p_A \cdot q_A$, where p_A, q_A prime.
$gcd(e_A, (p_A - 1)(q_A - 1)) = 1$.
$e_A \cdot d_A \equiv 1 \pmod{(p_A - 1)(q_A - 1)}$.

- $PK_A = (N_A, e_A)$
- $SK_A = (N_A, d_A)$

To encrypt: $c = E(m, PK_A) = m^{e_A} \pmod{N_A}$.
To decrypt: $r = D(c, SK_A) = c^{d_A} \pmod{N_A}$.
$r = m$.

Greatest Common Divisor

We need to find: e_A, d_A.
$gcd(e_A, (p_A - 1)(q_A - 1)) = 1$.
$e_A \cdot d_A \equiv 1 \pmod{(p_A - 1)(q_A - 1)}$.

Greatest Common Divisor

We need to find: e_A, d_A.
$gcd(e_A, (p_A - 1)(q_A - 1)) = 1$.
$e_A \cdot d_A \equiv 1 \pmod{(p_A - 1)(q_A - 1)}$.

Choose random e_A.
Check that $gcd(e_A, (p_A - 1)(q_A - 1)) = 1$.
Find d_A such that $e_A \cdot d_A \equiv 1 \pmod{(p_A - 1)(q_A - 1)}$.

The Extended Euclidean Algorithm

Theorem. $a, b \in \mathbb{N}$. $\exists\ s, t \in \mathbb{Z}$ s.t. $sa + tb = gcd(a, b)$.
Proof. Let d be the smallest positive integer in
$D = \{xa + yb \mid x, y \in \mathbb{Z}\}$.
$d \in D \ \Rightarrow\ d = x'a + y'b$ for some $x', y' \in \mathbb{Z}$.
$gcd(a, b)|a$ and $gcd(a, b)|b$, so $gcd(a, b)|x'a$, $gcd(a, b)|y'b$, and
$gcd(a, b)|(x'a + y'b) = d$. We will show that $d|gcd(a, b)$, so
$d = gcd(a, b)$. Note $a \in D$.
Suppose $a = dq + r$ with $0 \leq r < d$.

$$\begin{aligned} r &= a - dq \\ &= a - q(x'a + y'b) \\ &= (1 - qx')a - (qy')b \end{aligned}$$

$\Rightarrow\ r \in D$
$r < d \ \Rightarrow r = 0 \ \Rightarrow d|a$.
Similarly, one can show that $d|b$.
Therefore, $d|gcd(a, b)$. □

The Extended Euclidean Algorithm

How do you find d, s and t?

Let $d = gcd(a, b)$. Write b as $b = aq + r$ with $0 \leq r < a$.
Then, $d|b \Rightarrow d|(aq + r)$.
Also, $d|a \Rightarrow d|(aq) \Rightarrow d|((aq + r) - aq) \Rightarrow d|r$.

Let $d' = gcd(a, b - aq)$.
Then, $d'|a \Rightarrow d'|(aq)$
Also, $d'|(b - aq) \Rightarrow d'|((b - aq) + aq) \Rightarrow d'|b$.

Thus, $gcd(a, b) = gcd(a, b \pmod{a})$
$= gcd(b \pmod{a}, a)$. This shows how to reduce to a "simpler" problem and gives us the Extended Euclidean Algorithm.

The Extended Euclidean Algorithm

{ Initialize}

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

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

$n \leftarrow 1$

{ Compute next d}

while $d_n > 0$ **do**

begin

$n \leftarrow n + 1$

{ Compute $d_n \leftarrow d_{n-2} \pmod{d_{n-1}}$}

$q_n \leftarrow \lfloor d_{n-2}/d_{n-1} \rfloor$

$d_n \leftarrow d_{n-2} - q_n d_{n-1}$

$s_n \leftarrow s_{n-2} - q_n s_{n-1}$

$t_n \leftarrow t_{n-2} - q_n t_{n-1}$

end

$s \leftarrow s_{n-1} \qquad t \leftarrow t_{n-1}$

$gcd(a, b) \leftarrow d_{n-1}$

The Extended Euclidean Algorithm

Finding **multiplicative inverses** modulo m:

Given a and m, find x s.t. $a \cdot x \equiv 1 \pmod{m}$.

Should also find a k, s.t. $ax = 1 + km$.
So solve for an s in an equation $sa + tm = 1$.

This can be done if $gcd(a, m) = 1$.
Just use the Extended Euclidean Algorithm.

If the result, s, is negative, add m to s.
Now $(s - m)a + tm \equiv 1 \pmod{m}$.

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 $s \cdot 15 + t \cdot 23 = gcd(15, 23)$

RSA

$N_A = p_A \cdot q_A$, where p_A, q_A prime.
$gcd(e_A, (p_A - 1)(q_A - 1)) = 1$.
$e_A \cdot d_A \equiv 1 \pmod{(p_A - 1)(q_A - 1)}$.

- $PK_A = (N_A, e_A)$
- $SK_A = (N_A, d_A)$

To encrypt: $c = E(m, PK_A) = m^{e_A} \pmod{N_A}$.
To decrypt: $r = D(c, SK_A) = c^{d_A} \pmod{N_A}$.
$r = m$.

Primality testing

We need to find: p_A, q_A — large primes.

Choose numbers at random and check if they are prime?

Questions

1. How many random integers of length 1024 are prime?

Questions

1. How many random integers of length 1024 are prime?

Prime Number Theorem: About $\frac{x}{\ln x}$ numbers $< x$ are prime, so about $\frac{2^{1024}}{709}$

So we expect to test about 709 before finding a prime with 1024 bits.

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

Questions

1. How many random integers of length 1024 are prime?

About $\frac{x}{\ln x}$ numbers $< x$ are prime, so about $\frac{2^{1024}}{709}$

So we expect to test about 709 before finding a prime.

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

Questions

1. How many random integers of length 1024 are prime?

About $\frac{x}{\ln x}$ numbers $< x$ are prime, so about $\frac{2^{1024}}{709}$

So we expect to test about 709 before finding a prime.

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

Quite fast, using randomness.

Method 1

Sieve of Eratosthenes:
Lists:

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

Method 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

Method 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

Method 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

10^{308} — more than number of atoms in universe

So we cannot even write out this list!

Method 2

CheckPrime(n)

for $i = 2$ to $n - 1$ **do**
    check if i divides n
    if it does **then** output i
endfor
output -1 if divisor not found

Check all possible divisors between 2 and n (or $\sqrt{n}$).
Our sun will die before we're done!

Rabin–Miller Primality Testing

In practice, use a randomized primality test.

Miller–Rabin primality test:
Starts with Fermat test:

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

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

Rabin–Miller Primality Test

Fermat test:

Prime(n)

repeat r times
    Choose random $a \in \{1, 2, \ldots, n-1\}$
    if $a^{n-1} \pmod{n} \not\equiv 1$ **then** return(Composite)
end repeat
return(Probably Prime)

Carmichael Numbers Composite n.
For all $a \in \{1, 2, \ldots, n-1\}$ s.t. $\gcd(a, n) = 1$, $a^{n-1} \pmod{n} \equiv 1$.
Example: $561 = 3 \cdot 11 \cdot 17$

Theorem.
If p is prime, $\sqrt{1} \pmod{p} = \{x \mid x^2 \pmod{p} = 1\} = \{1, p-1\}$.
If p has > 1 distinct factors, 1 has at least 4 square roots.

Example: $\sqrt{1} \pmod{15} = \{1, 4, 11, 14\}$

Rabin–Miller Primality Test

Taking square roots of 1 (mod 561):

$50^{560} \pmod{561} \equiv 1$
$50^{280} \pmod{561} \equiv 1$
$50^{140} \pmod{561} \equiv 1$
$50^{70} \pmod{561} \equiv 1$
$50^{35} \pmod{561} \equiv 560$

$2^{560} \pmod{561} \equiv 1$
$2^{280} \pmod{561} \equiv 1$
$2^{140} \pmod{561} \equiv 67$

2 is a witness that 561 is composite.

Rabin–Miller Primality Test

Miller–Rabin(n, r)

Calculate odd m such that $n - 1 = 2^s \cdot m$
repeat r times
  Choose random $a \in \{1, 2, \ldots, n-1\}$
  if $a^{n-1} \pmod{n} \not\equiv 1$ **then** return(Composite)
  if $a^{(n-1)/2} \pmod{n} \equiv n - 1$ **then** continue
  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
  if $a^{(n-1)/4} \pmod{n} \not\equiv 1$ **then** return(Composite)
    ....
  if $a^{m} \pmod{n} \equiv n - 1$ **then** continue
  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, 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 r iterations, a composite n will survive to "return(Probably Prime)" with probability at most $(1/4)^r$. For e.g. $r = 100$, this is less than $(1/4)^{100} = 1/2^{200} < 1/10^{60}$.

A prime n will always survive to "return(Probably Prime)".

Conclusions about primality testing

1. Miller–Rabin is a practical primality test
2. There is a less practical deterministic primality test
3. Randomized algorithms are useful in practice
4. Algebra is used in primality testing
5. Number theory is not useless

Why does RSA work?

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 $1 \leq i \leq k$, and this integer is uniquely determined modulo the product $N = n_1 n_2 ... n_k$.

We 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, $0 \leq x, m \leq N - 1$, so we must have $x = m$.

Fermat's Little Theorem

Why does RSA work? CRT +

Fermat's Little Theorem: p is a prime, $p \not| a$.
Then $a^{p-1} \equiv 1 \pmod{p}$ and $a^p \equiv a \pmod{p}$.

RSA

$N_A = p_A \cdot q_A$, where p_A, q_A prime.
$gcd(e_A, (p_A - 1)(q_A - 1)) = 1$.
$e_A \cdot d_A \equiv 1 \pmod{(p_A - 1)(q_A - 1)}$.

- $PK_A = (N_A, e_A)$
- $SK_A = (N_A, d_A)$

To encrypt: $c = E(m, PK_A) = m^{e_A} \pmod{N_A}$.
To decrypt: $r = D(c, SK_A) = c^{d_A} \pmod{N_A}$.
$r = m$.

Correctness of RSA

Consider $x = D(E(m, PK_A), SK_A)$.
Note $\exists k$ s.t. $e_A d_A = 1 + k(p_A - 1)(q_A - 1)$.
$x \equiv (m^{e_A} \pmod{N_A})^{d_A} \pmod{N_A} \equiv m^{e_A d_A} \equiv m^{1+k(p_A-1)(q_A-1)} \pmod{N_A}$.

Consider $x \pmod{p_A}$.
$x \equiv m^{1+k(p_A-1)(q_A-1)} \equiv m \cdot (m^{(p_A-1)})^{k(q_A-1)} \equiv m \cdot 1^{k(q_A-1)} \equiv m \pmod{p_A}$.

Consider $x \pmod{q_A}$.
$x \equiv m^{1+k(p_A-1)(q_A-1)} \equiv m \cdot (m^{(q_A-1)})^{k(p_A-1)} \equiv m \cdot 1^{k(p_A-1)} \equiv m \pmod{q_A}$.

Apply the Chinese Remainder Theorem:
$gcd(p_A, q_A) = 1, \Rightarrow x \equiv m \pmod{N_A}$.
So $D(E(m, PK_A), SK_A) = m$.