Public key encryption

RSA public key encryption

1. Choose a bit-length $k$
2. Choose two primes $p$ and $q$ which can be represented with at most $k$ bits
3. Let $n = pq$ and $\phi(n) = (p-1)(q-1)$
4. Find $d$ such that $0 < d < n$ and $\gcd(d, \phi(n)) = 1$
5. Find $e$ such that $de \equiv 1 \pmod{\phi(n)}$
6. private key $= (d,n)$ and public key $= (e,n)$
7. $\text{encrypt}(m) = m^e \pmod{n}$, $\text{decrypt}(z) = z^d \pmod{n}$
Cracking RSA

1. Choose a bit-length \( k \)
2. Choose two primes \( p \) and \( q \) which can be represented with at most \( k \) bits
3. Let \( n = pq \) and \( \phi(n) = (p-1)(q-1) \)
4. Find \( d \) such that \( 0 < d < n \) and \( \gcd(d,\phi(n)) = 1 \)
5. Find \( e \) such that \( de \mod \phi(n) = 1 \)
6. private key = \( (d, n) \) and public key = \( (e, n) \)

Say I maliciously intercept an encrypted message. How could I decrypt it? (Note, you can also assume that we have the public key \( (e, n) \).)

Cracking RSA

encrypt(m) = m^e \mod n

Idea 1: undo the mod operation, i.e. mod^{-1} function

If we knew \( m^e \) and \( e \), we could figure out \( m \)

Do you think this is possible?

Cracking RSA

encrypt(m) = m^e \mod n

Idea 1: undo the mod operation, i.e. mod^{-1} function

If we knew \( m^e \) and \( e \), we could figure out \( m \)

Generally, no, if we don’t know anything about the message.

The challenge is that the mod operator maps many, many numbers to a single value.

Security of RSA

\( p \): prime number
\( q \): prime number
\( n = pq \)
\( \phi(n) = (p-1)(q-1) \)
\( d \): 0 < \( d \) < \( n \) and \( \gcd(d,\phi(n)) = 1 \)
\( e \): \( de \mod \phi(n) = 1 \)

private key \( (d, n) \) public key \( (e, n) \)

Assuming you can’t break the encryption itself (i.e. you cannot decrypt an encrypted message without the private key)

How else might you try and figure out the encrypted message?
### Security of RSA

**p**: prime number

\( \psi(n) = (p-1)(q-1) \)

**q**: prime number

\( d_1 \) \( 0 < d < n \) and \( \gcd(d, \psi(n)) = 1 \)

\( n = pq \)

\( e_2 \) \( de \mod \psi(n) = 1 \)

<table>
<thead>
<tr>
<th>private key</th>
<th>(d, n)</th>
<th>public key</th>
<th>(e, n)</th>
</tr>
</thead>
</table>

Assuming you can’t break the encryption itself (i.e. you cannot decrypt an encrypted message without the private key)

Idea 2: Try and figure out the private key!

How would you do this?

---

### Security of RSA

**p**: prime number

\( \psi(n) = (p-1)(q-1) \)

**q**: prime number

\( d_1 \) \( 0 < d < n \) and \( \gcd(d, \psi(n)) = 1 \)

\( n = pq \)

\( e_2 \) \( de \mod \psi(n) = 1 \)

<table>
<thead>
<tr>
<th>private key</th>
<th>(d, n)</th>
<th>public key</th>
<th>(e, n)</th>
</tr>
</thead>
</table>

Already know e and n.

If we could figure out p and q, then we could figure out the rest (i.e. d)!

---

### Security of RSA

For every prime p (2, 3, 5, 7, ...):

- If \( n \mod p = 0 \) then \( q = n / p \)

Why do we know that this must be p and q?
Security of RSA

| p: prime number | ϕ(n) = (p−1)(q−1) |
| q: prime number | d: 0 < d < n and gcd(d, ϕ(n)) = 1 |
| n = pq | e: de mod ϕ(n) = 1 |

private key (d, n) public key (e, n)

For every prime p (2, 3, 5, 7 ...):
- If n mod p = 0 then q = n / p

Since p and q are both prime, there are no other numbers that divide them evenly, therefore no other numbers divide n evenly.

Security of RSA

| p: prime number | ϕ(n) = (p−1)(q−1) |
| q: prime number | d: 0 < d < n and gcd(d, ϕ(n)) = 1 |
| n = pq | e: de mod ϕ(n) = 1 |

private key (d, n) public key (e, n)

For every number p (2, 3, 4, 5, 6, 7 ...):
- If n mod p = 0 then q = n / p

How long does this take?
I.e. how many p do we need to check in the worst case assuming n has k bits?

Security of RSA

| p: prime number | ϕ(n) = (p−1)(q−1) |
| q: prime number | d: 0 < d < n and gcd(d, ϕ(n)) = 1 |
| n = pq | e: de mod ϕ(n) = 1 |

private key (d, n) public key (e, n)

For every number p (2, 3, 4, 5, 6, 7 ...):
- If n mod p = 0 then q = n / p

Currently, there are no known “efficient” methods for factoring a number into its primes.
This is the key to why RSA works!
Implementing RSA

1. Choose a bit-length $k$

   For generating the keys, this is the only input the algorithm has

Implementing RSA

2. Choose two primes $p$ and $q$ which can be represented with at most $k$ bits

   Ideas?

Finding primes

2. Choose two primes $p$ and $q$ which can be represented with at most $k$ bits

   Idea: pick a random number and see if it’s prime

   How do we check if a number is prime?

Finding primes

2. Choose two primes $p$ and $q$ which can be represented with at most $k$ bits

   Idea: pick a random number and see if it’s prime

   isPrime(num):
   for $i = 2$ to $\sqrt{num}$:
   if num $\% i = 0$:
   return false
   return true

   If the number is $k$ bits, how many numbers (worst case) might we need to examine?
2. Choose two primes $p$ and $q$ which can be represented with at most $k$ bits

   Idea: pick a random number and see if it's prime

   - Again, with $k$ bits we can represent numbers up to $2^k$
   - Counting up to $\sqrt{2^k} = (2^k)^{1/2} = 2^{k/2}$

Finding primes

<table>
<thead>
<tr>
<th>Finding primes</th>
</tr>
</thead>
</table>
| Primality test for $num$:
|   - pick a random number $a$
|   - perform $test(num, a)$
|     - if test fails: return false
|     - if test passes: return true
|
| If $num$ is not prime, what are the chances that we incorrectly say $num$ is a prime? |

0.5 (50%)

Can we do any better?

Finding primes

<table>
<thead>
<tr>
<th>Finding primes</th>
</tr>
</thead>
</table>
| Primality test for $num$:
|   - pick a random number $a$
|   - perform $test(num, a)$
|     - if test fails: return false
|     - if test passes: return true
|
| Does this help us? |

Finding primes

<table>
<thead>
<tr>
<th>Finding primes</th>
</tr>
</thead>
</table>
| Primality test for $num$:
|   - pick a random number $a$
|   - perform $test(num, a)$
|     - if test fails: return false
|     - if test passes: return true
|
| Can we do any better? |
Finding primes

If $num$ is not prime, what are the chances that we incorrectly say $num$ is a prime?

Primality test for $num$:
- Repeat 2 times:
  - pick a random number $a$
  - perform $test(num, a)$
  - if test fails: return false
  - return true

If $num$ is not prime, what are the chances that we incorrectly say $num$ is a prime?

Primality test for $num$:
- Repeat 2 times:
  - pick a random number $a$
  - perform $test(num, a)$
  - if test fails: return false
  - return true

$p(0.25)$
- Half the time we catch it on the first test
- Of the remaining half, again, half (i.e. a quarter total) we catch it on the second test
- $\frac{1}{4}$ we don’t catch it

Finding primes

Primality test for $num$:
- Repeat 3 times:
  - pick a random number $a$
  - perform $test(num, a)$
  - if test fails: return false
  - return true

If $num$ is not prime, what are the chances that we incorrectly say $num$ is a prime?

Primality test for $num$:
- Repeat 3 times:
  - pick a random number $a$
  - perform $test(num, a)$
  - if test fails: return false
  - return true

$p(1/8)$
Finding primes

Primality test for \(num\):
- Repeat \(m\) times:
  - pick a random number \(a\)
  - perform \(test(num, a)\)
  - if test fails: return false
- return true

If \(num\) is not prime, what are the chances that we incorrectly say \(num\) is a prime?

Primality test for \(num\):
- Repeat \(m\) times:
  - pick a random number \(a\)
  - perform \(test(num, a)\)
  - if test fails: return false
- return true

\[p(\frac{1}{2^m})\]

For example, \(m = 20\): \(p(\frac{1}{2^{20}}) = p(\frac{1}{1,000,000})\)

Fermat’s little theorem:
If \(p\) is a prime number, then for all integers \(a\):
\[a^p \equiv a \pmod{p}\]

How does this help us?

Fermat’s little theorem:
If \(p\) is a prime number, then for all integers \(a\):
\[a^p \equiv a \pmod{p}\]

\(test(num, a)\):
- generate a random number \(a < p\)
- check if \(a^p \mod p = a\)
Implementing RSA

1. Choose a bit-length \( k \)
2. Choose two primes \( p \) and \( q \) which can be represented with at most \( k \) bits
3. Let \( n = pq \) and \( \varphi(n) = (p-1)(q-1) \)

   How do we do this?

Implementing RSA

4. Find \( d \) such that \( 0 < d < n \) and \( \gcd(d,\varphi(n)) = 1 \)
5. Find \( e \) such that \( de \mod \varphi(n) = 1 \)

   How do we do these steps?

Greatest Common Divisor

A useful property:

If two numbers are relatively prime (i.e. \( \gcd(a,b) = 1 \)), then there exists \( a \) \( c \) such that

\[ a^c \mod b = 1 \]

Greatest Common Divisor

A more useful property:

two numbers are relatively prime (i.e. \( \gcd(a,b) = 1 \)) \( iff \) there exists \( a \) \( c \) such that \( a^c \mod b = 1 \)

What does iff mean?
Greatest Common Divisor

A more useful property:

1. If two numbers are relatively prime (i.e. \( \gcd(a,b) = 1 \)), then there exists a \( c \) such that \( a \cdot c \mod b = 1 \)

2. If there exists a \( c \) such that \( a \cdot c \mod b = 1 \), then the two numbers are relatively prime (i.e. \( \gcd(a,b) = 1 \))

We’re going to leverage this second part

Implementing RSA

4. Find \( d \) such that \( 0 < d < n \) and \( \gcd(d,\phi(n)) = 1 \)

5. Find \( e \) such that \( de \mod \phi(n) = 1 \)

If there exists a \( c \) such that \( a \cdot c \mod b = 1 \), then the two numbers are relatively prime (i.e. \( \gcd(a,b) = 1 \))

To find \( d \) and \( e \):
- pick a random \( d \), \( 0 < d < n \)
- try and find an \( e \) such that \( de \mod \phi(n) = 1 \)
  - if none exists, try another \( d \)
  - if one exists, we’re done!

Known problem with known solutions

For the assignment, I’ve provided you with a function: \texttt{inversemod}
Option type

Look at option.sml
http://www.cs.pomona.edu/~dkauchak/classes/cs52/examples/option.sml

option type has two constructors:
- NONE (representing no value)
- SOME v (representing the value v)

case statement

case _______ of
  pattern1 => value
  | pattern2 => value
  | pattern3 => value
  ...

inversemod

```sml
* inversemod : nat * nat * nat * nat
  * inversemod a b (c, d) returns (c / d) when a * c = b * d
  * inversemod a b returns SOME v if 0 < v < |a| and a * v = 1
  * inversemod a b returns NONE if there is no such v.
  *<
```

Signing documents

If a message is encrypted with the private key how can it be decrypted?

Hint:
- \((m^e)^d = m^{ed} = m \pmod n\)
- encrypt(m, (e, n)) = m^e \pmod n
- decrypt(z, (d, n)) = z^d \pmod n
Signing documents

- \( (m^e)^d = m^{ed} \equiv m \pmod{n} \)
- \( \text{encrypt}(m, (e, n)) = m^e \pmod{n} \)
- \( \text{decrypt}(z, (d, n)) = z^d \pmod{n} \)

encrypt\((m, (d, n)) = m^d \pmod{n} \)

decrypt\((m^d \pmod{n}, (e, n)) = (m^e)^d \pmod{n} \)
\[ = m^{ed} \pmod{n} \]
\[ = m \pmod{n} \]
\[ = m \] if \( m < n \)

What does this do for us?

If the message can be decrypted with the public key then the sender must have had the private key
This is a way to digitally sign a document!
Signing documents

Confirmed: batman likes bananas

I like bananas

Send signed message

I like bananas

Public key encryption

Share your public key with everyone

How does this happen?

Anything we have to be careful of?