Algoritmid ja andmestruktuurid

- Arvuteoreetilised algoritmid
  - Eukleidese algoritm ja selle laiendus
  - modulaarne aritmeetika
  - algarvulisuse kontroll
- Mittesümmeetrilise võtmega krüptosüsteem RSA

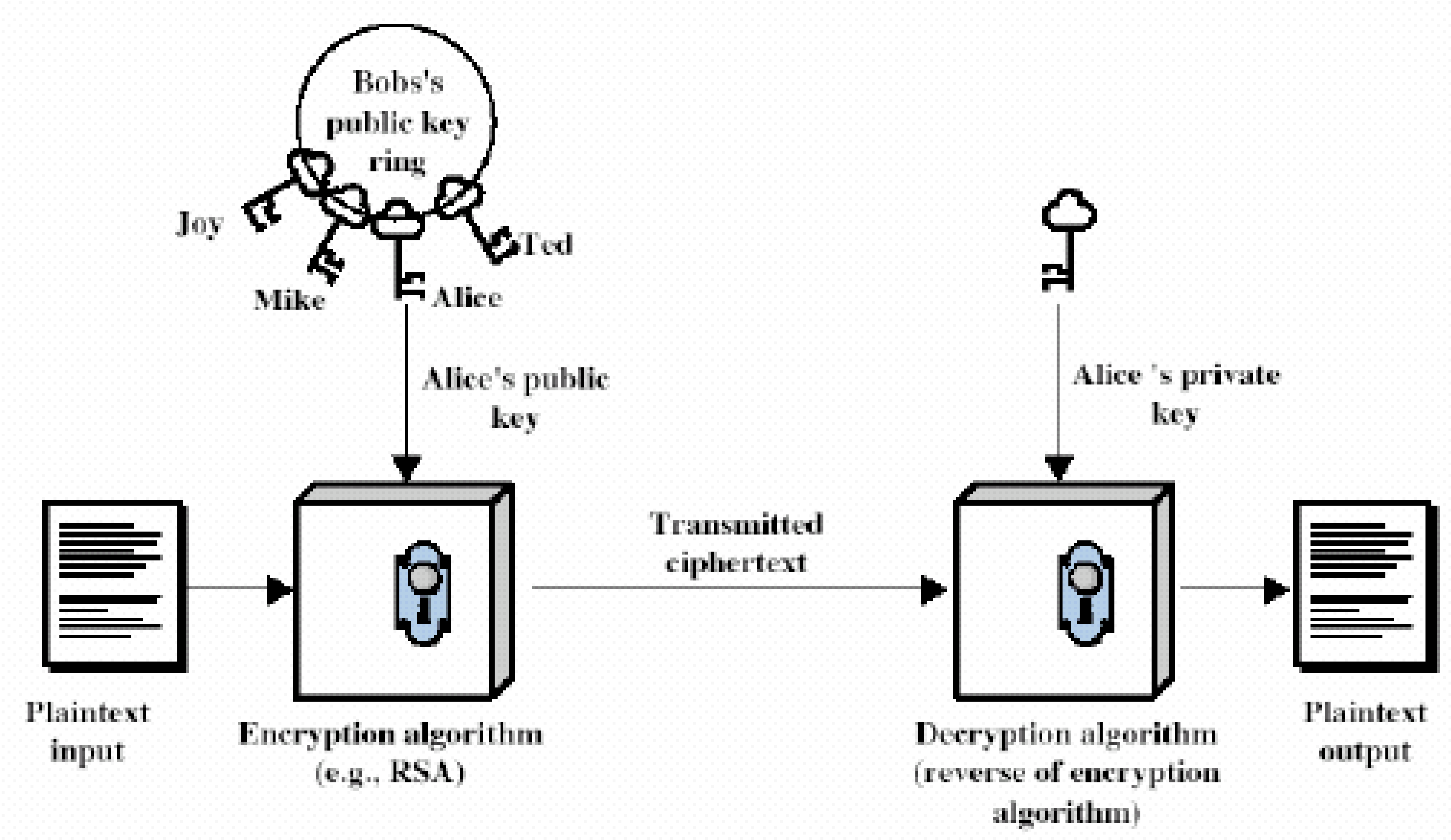
Teated

- Eksamid
  - 4. ja 11. jaanuaril kell 14
  - 19. jaanuar kell 12
  - Konsultatsioonid eksamile eelneval päeval
- Eksami eelduseks on
  - 3 programmeerimistööd kaitstud/hinnatud
  - 10 punkti kontrolltööst
  - 3.5 punkti veebitestidest
  - 4 punkti praktikumi ülesannetest

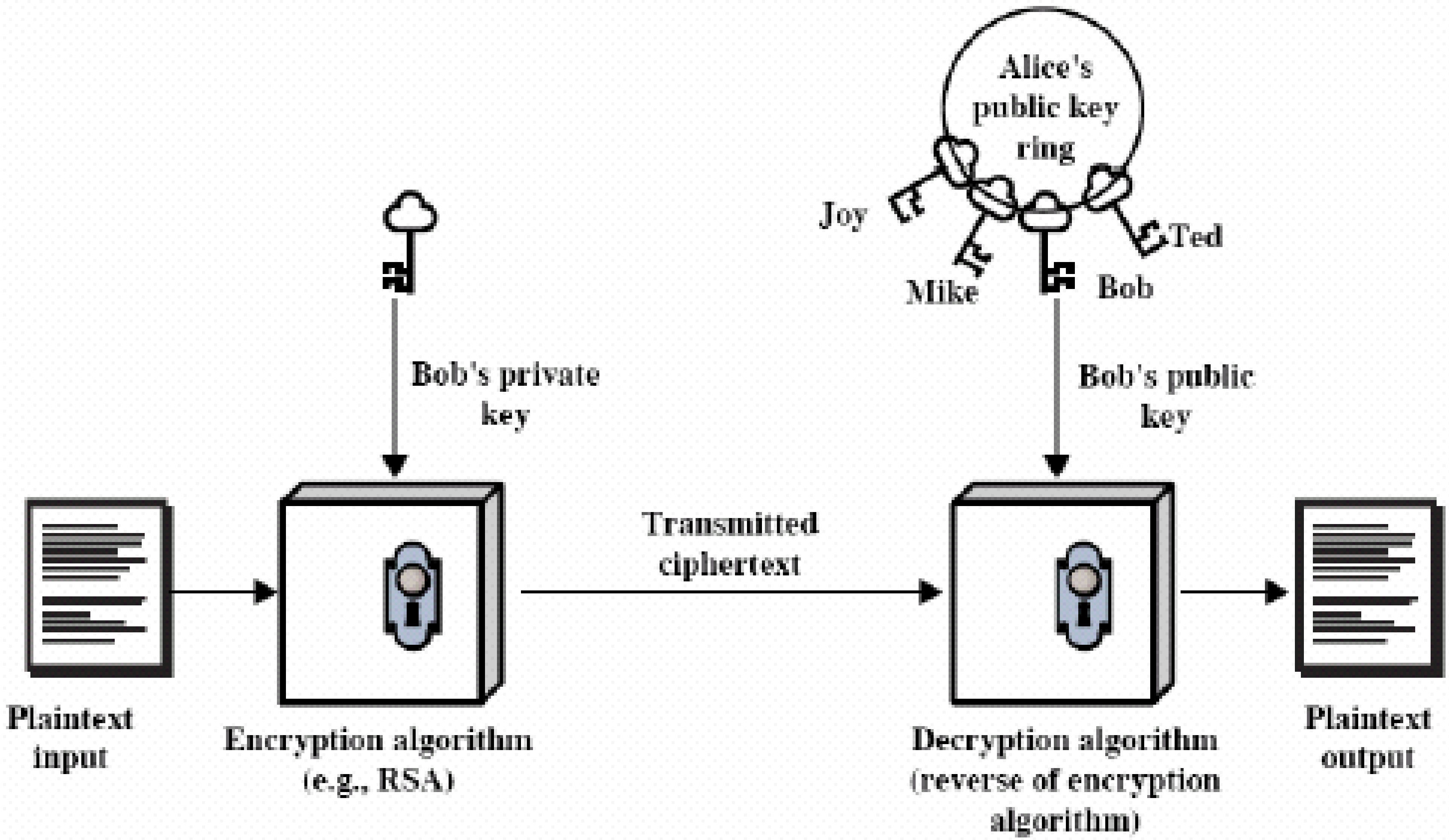
Public Key Cryptography

- **Public-key/two-key/asymetric** cryptography involves the use of two keys:
  - a **public key**, which may be known to anybody, and can be used to **encrypt messages**, and **verify signatures**
  - a **private key**, known only to the owner, used to **decrypt** messages, and **sign** (create) **signatures**
- Is **asymetric** because
  - those who encrypt messages or verify signatures cannot decrypt messages or create signatures

Encryption


Bobs's public key ring
Joy
Mike
Alice
Ted
Alice's public key
Alice 's private key
Transmitted ciphertext
Plaintext input
Encryption algorithm (e.g., RSA)
Decryption algorithm (reverse of encryption algorithm)
Plaintext output

Authentication



Public Key Characteristics

Public-key algorithms rely on two keys with the characteristics that it is:

- computationally infeasible to find private key knowing only algorithm and public key
- computationally easy to en/decrypt messages when the relevant key is known
- either of the two related keys can be used for encryption, with the other used for decryption

  Convention sometimes used:
  - public key - encryption key
  - private key - decryption key

Prime Numbers

- prime numbers only have divisors of 1 and self
  - they cannot be written as a product of other numbers
  - note: 1 is prime, but is generally not of interest
- eg. 2,3,5,7 are prime, 4,6,8,9,10 are not
- prime numbers are central to number theory
- list of prime number less than 200 is:

```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59
61 67 71 73 79 83 89 97 101 103 107 109 113
127 131 137 139 149 151 157 163 167 173 179
181 191 193 197 199
```

Prime Factorisation

- to **factor** a number n is to write it as a product of other numbers: $n = a \times b \times c$
- note that factoring a number is relatively hard compared to multiplying the factors together to generate the number
- the **prime factorization** of a number n is when its written as a product of primes

  eg. $91 = 7 \times 13$; $3600 = 2^4 \times 3^2 \times 5^2$

Relatively Prime Numbers

- two numbers a, b are **relatively prime** if they have **no common divisors** apart from 1
  - eg. 8 & 15 are relatively prime since factors of 8 are 1,2,4,8 and of 15 are 1,3,5,15 and 1 is the only common factor

GCD - Greatest Common Divisor

- Definition

  Given *a* and *b* we want to find *c*, the largest number that exactly divides both *a* and *b*.

- We can determine the greatest common divisor by comparing their prime factorizations and using least powers

  eg. $300=2^1\times3^1\times5^2$ $18=2^1\times3^2$ hence $GCD(18,300)=2^1\times3^1\times5^0=6$

- If c=1, then *a* and *b* are *relatively prime*.
- GCD can be used to reduce fractions:
  - The GCD of numerator and denominator cancel $\frac{18}{300}=\frac{3}{50}$

Naive GCD algorithm

```
gcd(a, b)
for i = min(a, b) downto 1 do
  if a==0(mod i) and b==0(mod i)
    then return i
```

- Not very efficient if we have to find a gcd of LARGE values ($>10^{100}$)

Euclid algoritm

```
very_long_int gcd(very_long_ int a, very_long_ int b)
{ if( b=0 )
    then return a;
    else return gcd(b, a % b);
}
```

Modular reduction for large values *a* and *b* is expensive!

a	b	a mod b
1071	1029	42
1029	42	21
42	21	0
21	0	

Why Euclid algorithm works

- Remainder of the division of a by b is t

  $a=q*b+t$, where q is the quotient of the division

- Any divisor of both a and b also divides t.

  $t=a-q*b$

- In the same way, any common divisor of b and t will also divide a.

  $\gcd(a, b) = \gcd(b, t)$

Improvement of Euclid algorithm

- Computers work in base-2 - some operations are fast even for large numbers
  - division by two is shift by one bit
  - checking if a value is odd or even is just a test of the least significant bit
- Several reduction rules
  - *a* and *b* are even: $\gcd(a, b) = 2 \cdot \gcd(a/2, b/2)$
  - *a* is even and *b* is odd: $\gcd(a, b) = \gcd(a/2, b)$
  - *a* is odd and *b* is even: $\gcd(a, b) = \gcd(a, b/2)$
  - *a* and *b* are odd: $\gcd(a, b) = \gcd(|a-b|/2, b)$
- Constant factor speedup

Extended Euclid algoritm - XGCD

returns [t,x,y] such that gcd(a,b) = t = ax+by

```
very_long_int[3] xgcd(very_long_int a, very_long_int b)
{ int t, x, y;
  if( b==0 )
    then return [a, 1, 0];
    else
      [t, x, y] = xgcd(b, a % b);
      return [t, y, x-[a/b]*y );
}
```

1071*-24+1029*25=21

a	*b*	*a* mod *b*	*t*	*x*	*y*
1071	1029	42	21	-24	25
1029	42	21	21	1	-24
42	21	0	21	0	1
21	0		21	1	0

Modular Arithmetic

- Public key cryptoalgorithms are based on modular arithmetic.
  - All computations are done modulo n

    $x = x \bmod n$

    Results in $\{0, \ldots, n\text{-}1\}$
  - Based on algebraic group theory
- Modular addition.
- Modular multiplication.
- Modular exponentiation.

Modular Addition

- Addition modulo (mod) K

  $(a+b) \bmod K = (a \bmod K + b \bmod K) \bmod K$

- Additive inverse: addition $\bmod K$ yields 0.
  - Poor cipher with $c = (d_k + m) \bmod K$, e.g., if $K=10$ and d_k is the key.
  - "Decrypt" by adding inverse.

    $d_d = (K - d_k) \bmod K$

    $m = (d_d + c) \bmod K$
  - This chiper is very poor

    $d_k = (K - d_d) \bmod K = (c - m) \bmod K$

Modular Multiplication

- Multiplication modulo *K*

  $(a*b) \bmod K = (a \bmod K * b \bmod K) \bmod K$
- Multiplicative inverse:
  multiplication mod *K* yields 1
- Only some numbers have inverse
- Use *Euclid*'s algorithm to find inverse
  - Given *e*, *K*, it finds *d* such that $e*d \bmod K = 1$
- All number *relatively prime* to *K* will have mod *K* multiplicative inverse

Finding Multiplicative Inverse

$(e*d) \bmod k = 1$

To find multiplicative inverse d of e over module k find

$$(t, x, y) = xgcd(k,e)$$

- $t = 1$ k, e are relatively prime

  $kx + ey = t$

  $kx + ey = 1$

- $kx \bmod k = 0$

  $(kx + ey) \bmod k = ey \bmod k$

  $ey \bmod k = 1$

- $d = y \bmod k$ d is the inverse we are looking for

Modular Exponentiation

- Can be done using modular multiplication, but it is inefficient
- Repeated Squaring - efficient way to compute

  $a^b (\text{mod } n)$, a, b, n are positive integers

  - uses binary representation of b

$$7^{11} = ((7^2)^{2} * 7)^{2} * 7 = 7^{(2*2+1)*2+1} = 7^{11}$$

  - 5 multiplications instead of 11

  multiplication has exp-complexity on size of arguments

Modular Exponentiation

modular-exp(a, b, n) //finds $s = a^b \bmod n$

c ← 0; d ← 1

let $<b_k, b_{k-1}, \ldots , b_0>$ be the binary repr of b

for $i \leftarrow k$ **downto** 0 **do**

$c \leftarrow 2c$ // c is not needed in the algorithm

$s \leftarrow (s*s) \bmod n$

if b_i =1 **then**

$c \leftarrow c + 1$

$s \leftarrow (s*a) \bmod n$

return s

$\mathbf{7^{560} \bmod 561}$

i	9	8	7	6	5	4	3	2	1	0
b_i	1	0	0	0	1	1	0	0	0	0
c	1	2	4	8	17	35	70	140	280	560
s	7	49	157	526	160	241	298	166	67	1

Fermat's Little Theorem

- $a^{n-1} \bmod n = 1$
  - *n* is prime
  - *a* and *n* are relatively prime gcd(*a*,*n*) = 1
    ie *n* is not a factor of *a*

- Useful in asymetric crytography and primality testing

Probabilistic primality tests

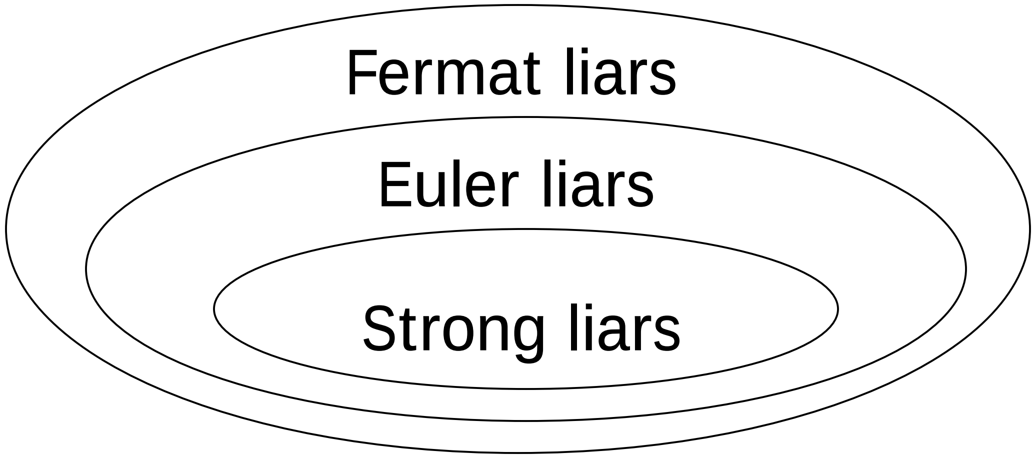
- Fermat's test
  - We have that if n is prime and $1 \leq a \leq n - 1$, then $a^{n-1} \equiv 1 \pmod{n}$
  - If n is composite, then for some a if
    $$a^{n-1} \neq 1 \pmod{n}$$
    a is called a *witness* for n. Conversely, if $a^{n-1} \equiv 1 \pmod{n}$ then a is called a *liar* for n
  - ```
    For i = 1 to t
        choose random a, 2 ≤ a ≤ n-2
        r = a^(n-1) mod n
        if r ≠ 1, return composite
    return prime
    ```
    ```

Probabilistic primality tests

- Carmichael numbers are composite integers such that $a^{n-1} \equiv 1 \pmod{n}$ for all integers a st. $\gcd(a,n) = 1$
- Although Carmichael numbers are quite few in numbers they demonstrate that Fermat's test behaves poorly in the worst case.
- There are other probabilistic primality tests that have less *liars*.

Probabilistic primality tests

- Comparison



Totient Function

- *x*, *m* relative prime: no other common factor than 1
- Totient function $ø(n)$: number of integers less than *n* relatively prime to *n*
  - if *n* is prime, $ø(n)=n-1$
  - if $n=p*q$, and *p*, *q* are primes, $ø(n)=(p-1)(q-1)$
- Examples:
  - ø(37) = 36
  - ø(14) = (2-1) * (7-1) = 6

Euler's Theorem

- A generalisation of Fermat's Theorem

  $a^{\text{ø}(n)} \bmod n = 1,$

  where $\gcd(a, n) = 1$

- eg
  - a =3; n = 10; ø(10) = 4;
  - hence $3^4 = 81 = 1 \bmod 10$

- Corollary

  $x^y \bmod n = x^{y \bmod \text{ø}(n)} \bmod n$, for n prime

Public Key Crypto Characteristics

Public-key algorithms rely on two keys with the characteristics that it is:

- computationally infeasible to find private key knowing only algorithm and public key
- computationally easy to en/decrypt messages when the relevant key is known
- either of the two related keys can be used for encryption, with the other used for decryption

Convention sometimes used:

  - public key - encryption key
  - private key - decryption key

RSA (Rivest, Shamir, Adleman)

- The most popular one.
- Support both public key encryption and digital signature.
- Assumption/theoretical basis:
  - Factoring a big number is hard.
- Variable key length (usually 1024 or 2048 bits).
- Variable plaintext block size.
  - Plaintext must be “smaller” than the key.
  - Ciphertext block size is the same as the key length.

What Is RSA?

- To generate key pair:
  - Pick large primes (>= 512 bits each) *p* and *q*
  - Let $n = p*q$, keep your *p* and *q* to yourself!
  - For public key, choose *e* that is relatively prime to $ø(n) = (p-1)(q-1)$, let pub = $\langle e,n \rangle$
  - For private key, find *d* that is the multiplicative inverse of *e* mod $ø(n)$, i.e., $e*d \bmod ø(n) = 1$, let priv = $\langle d,n \rangle$

How Does RSA Work?

- Given pub = <e, n> and priv = <d, n>
  - encryption: $c = m^e \bmod n$, $m < n$
  - decryption: $m = c^d \bmod n$
  - signature: $s = m^d \bmod n$, $m < n$
  - verification: $m = s^e \bmod n$

Why Does RSA Work?

- Given pub = <e, n> and priv = <d, n>
  - $n = p*q$, $ø(n) = (p-1)(q-1)$
  - $e*d$ mod $ø(n) = 1$
  - $x^{e*d} \pmod{n} = x^{e*d \bmod ø(n)} \pmod{n} = x$
  - encryption: $c = m^e \bmod n$
  - decryption:
    $m = c^d = m^{e*d} = m \pmod{n}$
    $m \pmod{n} = m$ (since $m < n$)
  - digital signature (similar)

RSA example

1. Select primes: p=17 & q=11
2. Compute $n = pq$ =17×11=187
3. Compute $\varnothing(n)=(p-1)(q-1)$=16×10=160
4. Select e : gcd(e,160)=1; choose e=7
5. Determine d: $de=1 \mod 160$ and $d < 160$ Value is d=23 since 23×7=161= 10×160+1
6. Publish public key KU={7,187}
7. Keep secret private key KR={23,187}

RSA example

- sample RSA encryption/decryption is:
- given message $M = 88$ (nb. $88<187$)
- encryption:

$$C = 88^7 \bmod 187 = 11$$

- decryption:

$$M = 11^{23} \bmod 187 = 88$$

RSA example 2

- Communication between Alice and Bob
- Alice chooses:
  - p=7, q=11. So n=77 and $\varnothing(n)=60$
  - gcd(e, 60)=1; choose e = 17
  - $de=1 \bmod 60$ and $d < 60$ Value is d=53 since $53 \times 17 = 901 = 15 \times 60 + 1$
- Plaintext: HELLO WORLD
- A – 00, …, Z –25, blank – 26
- Plaintext: 07 04 11 11 14 26 22 14 17 11 03

RSA example 2

- $07^{17} \bmod 77 = 28$, …, $03^{17} \bmod 77 = 75$
- Ciphertext : 28 16 44 44 42 38 22 42 19 44 75
- No one would use RSA on blocks of small size
- Open to rearrangements of blocks and language analysis
- The plaintext is padded with random data to make up a block
- Include information about the order of blocks. If rearranged the receiver will be aware

Why Is RSA Secure?

- Factoring 512-bit number is hard, factoring 1024 -bit number is infeasible now! 768-bit number factored in 2009
- But if you can factor big number n then given public key $<e,n>$, you can find d, hence the private key by:
  - Knowing factors p, q, such that, $n = p*q$
  - Then $ø(n) =(p-1)(q-1)$
  - Then d such that $e*d = 1 \bmod ø(n)$

Breaking RSA

- Mathematical approach takes 3 forms
  - factor $n=p*q$, hence find p, q, d
  - determine $ø(n)$, directly
  - find d directly
- Currently believed to be as hard as factoring
  - breaking RSA does not need to solve factoring problem
  - quantum computers (if built) are able to factor large numbers
  - known plain text attacks exist

Assignment

- RSA parameters are p=11, q=13, e=7.
  - Use xgcd() to find the secret key *d*?
  - Use modular exponention algorithm to find the result of encryption of message M=9 with key *d*?
  - Try to decrypt your result!