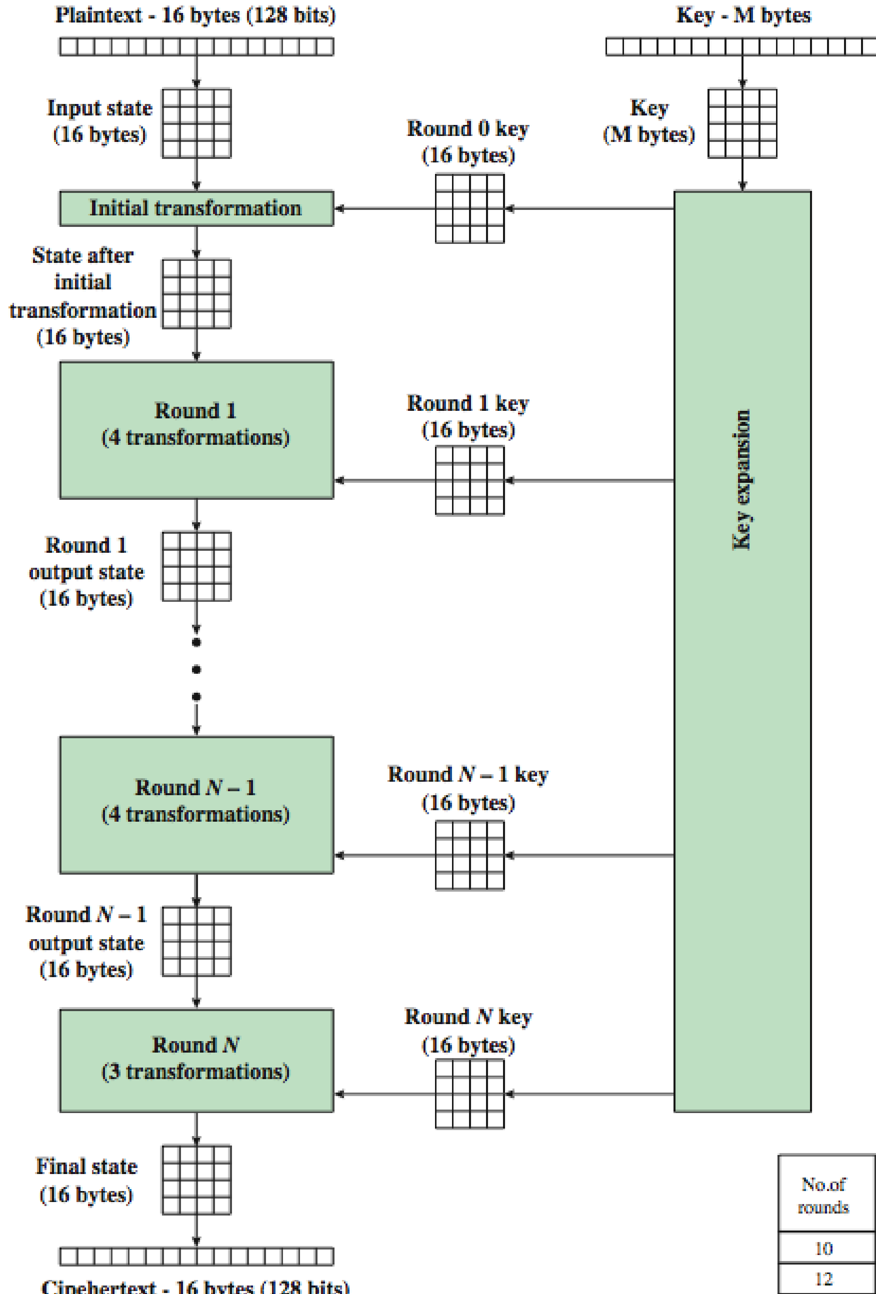
Corso di Sicurezza dei Sistemi
Prof. Salvatore D'Antonio

RSA



No.of rounds	Key Length (bytes)
10	16
12	24
14	32

AES structure

- Data block of 4 columns of 4 bytes is state
- Key is expanded to array of words
- Has 9/11/13 rounds in which state undergoes:
  - byte substitution (1 S-box used on every byte)
  - shift rows (permute bytes between columns)
  - mix columns (subs using matrix multiply of groups)
  - add round key (XOR state with key material)

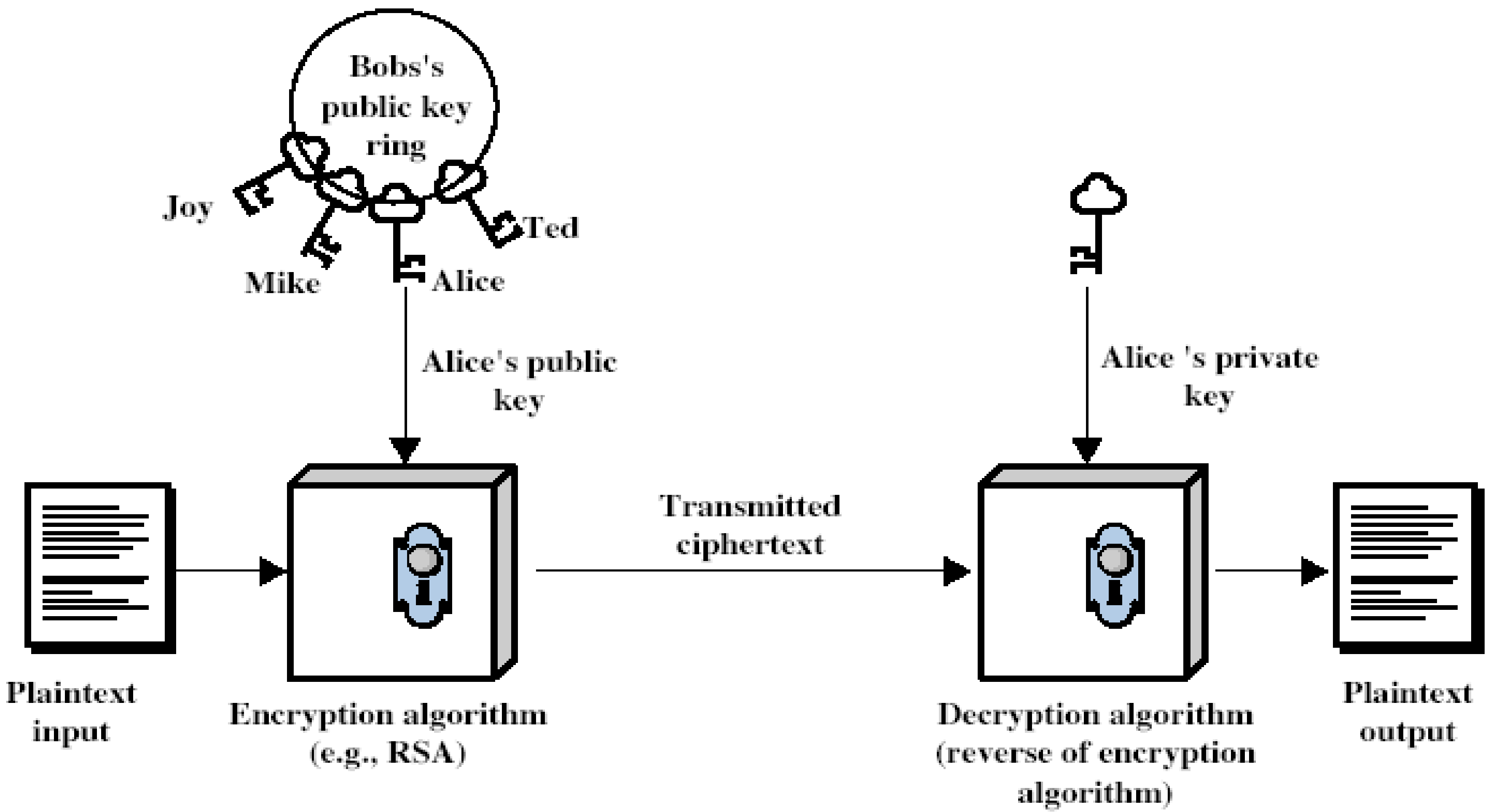
Public key cryptography

- Two keys
  - Private key known only to individual
  - Public key available to anyone
- Idea
  - Confidentiality: encipher using public key, decipher using private key
  - Integrity/authentication: encipher using private key, decipher using the public one

Requirements

- A public key encryption algorithm has to meet the following requirements:
  - It must be computationally easy to encipher or decipher a message given the appropriate key
  - It must be computationally infeasible to derive the private key from the public key
  - It must be computationally infeasible to determine the private key from a chosen plaintext attack

Public-Key Cryptography



Modular Arithmetic

- Public key algorithms are based on modular arithmetic.
  - Modular addition.
  - Modular multiplication.
  - Modular exponentiation.

Modular Addition

- Addition modulo (mod) *K*
  - (d_k+d_m) mod *K*, e.g., if $K=10$ and d_k is the key

+	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	0
2	2	3	4	5	6	7	8	9	0	1
3	3	4	5	6	7	8	9	0	1	2

- Additive inverse: addition mod *K* yields 0.

Modular Multiplication

- Multiplication modulo *K*

*	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9
2	0	2	4	6	8	0	2	4	6	8
3	0	3	6	9	2	5	8	1	4	7

- Multiplicative inverse: multiplication mod *K* yields 1
- Only some numbers have inverse

Modular Multiplication

- Only the numbers *relatively prime* to *n* will have mod *n* multiplicative inverse
- *x*, *m* are relatively prime: no other common factor than 1
  - Eg. 8 and 15 are relatively prime - factors of 8 are 1,2,4,8 and of 15 are 1,3,5,15 and 1 is the only common factor

Totient Function

- 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, p != q
    - $ø(n)=(p-1)(q-1)$
  - E.g.,
    - `ø(37) = 36`
    - `ø(21) = (3-1)×(7-1) = 2×6 = 12`

Modular Exponentiation

- Modular exponentiation calculates the remainder when an integer *b* (the base) raised to the *e*th power (the exponent), b^e, is divided by a positive integer *m* (the modulus)
- $c = b^e \bmod m$
- From the definition of c, it follows that $0 \le c < m$

Modular Exponentiation

x^y	0	1	2	3	4	5	6	7	8	9
0		0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1
2	1	2	4	8	6	2	4	8	6	2
3	1	3	9	7	1	3	9	7	1	3
4	1	4	6	4	6	4	6	4	6	4
5	1	5	5	5	5	5	5	5	5	5
6	1	6	6	6	6	6	6	6	6	6
7	1	7	9	3	1	7	9	3	1	7
8	1	8	4	2	6	8	4	2	6	8
9	1	9	1	9	1	9	1	9	1	9

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 512 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 (>= 256 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 = $<e,n>$
  - For private key, find *d* that is the multiplicative inverse of *e* mod $ø(n)$, i.e., $e*d = 1 \bmod ø(n)$, let priv = $<d,p,q>$

RSA Example

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

- *gcd = greatest common divisor

How Does RSA Work?

- Given pub = <e, n> and priv = <d, n>
  - encryption: $c = m^e \bmod n$, $m < n$
  - decryption: $m = c^d \bmod n$
- given message `M = 88` (nb. `88<187`)
- encryption:

  `C =` 88^7 `mod 187 = 11`
- decryption:

  `M =` 11^{23} `mod 187 = 88`

Why Does RSA Work?

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

Lab exercise - Input

- Message
  - ��N,���MWNf3z�Q�q�O2����3'5��'tc
- Public key
  - File my.pub
- To get public key info
  - Launch openssl
  - rsa -inform PEM -text -noout -pubin -in my.pub
  - You will get the information that this is a 256 bit key. You will also get the modulus (just remove the colons) and the exponent e.

Lab exercise

- The structure of the RSA private key is
- RSAPrivateKey ::= SEQUENCE {

```
version           Version,
modulus           INTEGER,  -- n
publicExponent    INTEGER,  -- e
privateExponent   INTEGER,  -- d
prime1            INTEGER,  -- p
prime2            INTEGER,  -- q
exponent1         INTEGER,  -- d mod (p-1)
exponent2         INTEGER,  -- d mod (q-1)
coefficient       INTEGER,  -- (inverse of q) mod p
otherPrimeInfos   OtherPrimeInfos OPTIONAL
}
```

Lab exercise

- Factorization of the public key modulus to get p and q
- Use the online factorization tool available at https://www.alpertron.com.ar/ECM.HTM
- Modulus is 0x00b59956b45ff72a0e0f86f9c33f379a97db05a22e20b7d4f9e3e67dd13f578b59
- p and q are
  - p=INTEGER:283916467959484617011451501441821706737
  - q=INTEGER:289308866787136623255147447840735195113

Lab exercise

- Once we get p and q, we need to calculate:
  - *ø(n)= (p-1)*(q-1)*
  - *d = modinv(e, ø)*
  - *dp = d mod(p-1)*
  - *dq = d mod(q-1)*
  - *qi = modinv(q,p)*

Lab exercise

- for the modinv you can just use a script similar to this python-script:

```
def egcd(a,b):
    if a == 0:
        return (b,0,1)
    else:
        g,y,x=egcd(b % a, a)
        return (g,x - (b // a) * y, y)

def modinv(a,m):
    gcd, x, y = egcd(a,m)
    if gcd != 1:
        return None
    else:
        return x % m
```

Keyfile generation

- You will use a config-file and the openssl asn1parse generator
- The config-file needs to have the following structure:

- asn1=SEQUENCE:rsa_key

- [rsa_key]
- version=INTEGER:0
- modulus=INTEGER:*xxxx*
- pubExp=INTEGER:*xxxx*
- privExp=INTEGER:*xxxx*
- p=INTEGER:*xxxx*
- q=INTEGER:*xxxx*
- e1=INTEGER:*xxxx*
- e2=INTEGER:*xxxx*
- coeff=INTEGER:*xxxx*

Keyfile generation

- To generate a DER encoded key
  - Launch openssl
  - asn1parse -genconf conf.cnf -out newkey.der
- You can check whether you have generated a RSA key with the right values using the following command
  - rsa -in newkey.der -inform der -text –check

Decryption

- Now that you have your private key you can use it to decrypt the message
  - cat message | openssl rsautl -decrypt -keyform DER -inkey newkey.der
- Alternatively you can also generate a private key in PEM form from the DER encoded key using the following command in openssl
  - rsa -inform der -outform pem -in newkey.der -out new.key
- To decrypt the message
  - cat message | openssl rsautl -decrypt -inkey new.key
  - {AREyouKIDDINGme}