UNIVERSITA' DEGLI STUDI DI PARMA
Dipartimento di Ingegneria dell'Informazione

# Public Key (asymmetric) Cryptography

Luca Veltri
(mail.to: luca.veltri@unipr.it)

Course of Network Security, Spring 2014

http://www.tlc.unipr.it/veltri



## Public-Key Cryptography

- Also referred to as asymmetric cryptography or two-key cryptography
- Probably most significant advance in the 3000 year history of cryptography
  - **public invention due to Whitfield Diffie & Martin Hellman in 1975**
    - at least that's the first published record
    - known earlier in classified community (e.g. NSA?)
- Is asymmetric because
  - **who encrypts messages or verify signatures cannot decrypt messages or create signatures**
  - **more in general, operation performed by two parties use different key values**
- Uses clever application of number theoretic concepts and mathematic functions rather than permutations and substitutions





## Public-Key vs. Secret Cryptography

- All secret key algorithms do the same thing
  - **they take a block and encrypt it in a reversible way**
- All hash algorithms do the same thing
  - **they take a message and perform an irreversible transformation**
- Instead, public key algorithms look very different
  - **in how they perform their function**
  - **in what functions they perform**
- They have in common: a private and a public quantities associated with a principal





## Public-Key vs. Secret Cryptography (cont.)

- Public key cryptography can do anything secret key cryptography can do, but..
- The known public-key cryptographic algorithms are orders of magnitude slower than the best known secret key cryptographic algorithms
  - **are usually used only for things secret key cryptography can't do (or can't do in a suitable way)**
- Complements rather than replaces secret key crypto
  - **often it is mixed with secret key technology**
  - **e.g. public key cryptography might be used in the beginning of communication for authentication and to establish a temporary shared secret key used to encrypt the conversation**

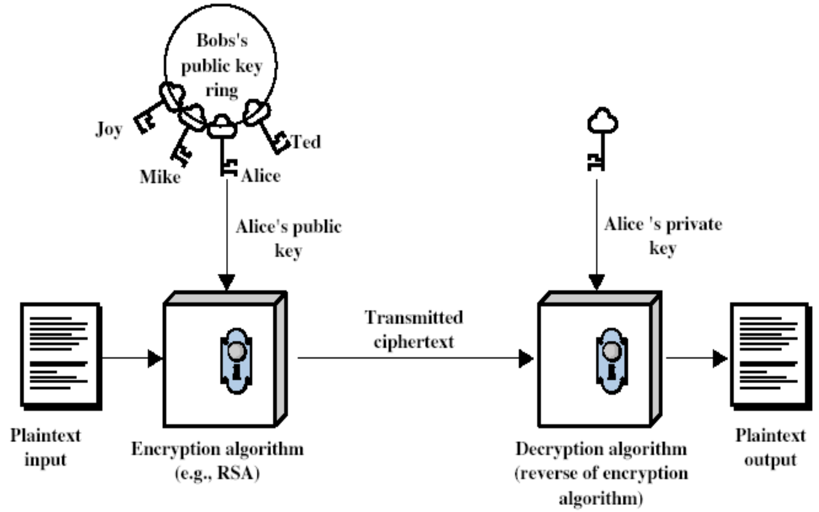
# Public-Key vs. Secret Cryptography (cont.)

- With symmetric/secret-key cryptography
  - **you need a secure method of telling your partner the key**
  - **you need a separate key for everyone you might communicate with**
- Instead, with public-key cryptography, keys are not shared
- Public-key cryptography often uses two keys:
  - **a public-key, which may be known by anybody, and can be used to encrypt messages, or verify signatures**
  - **a private-key, known only to the recipient, used to decrypt messages, or sign (create) signatures**
  - **it is computationally easy to en/decrypt messages when key is known**
  - **it is computationally infeasible to find decryption key knowing only encryption key (and vice-versa)**
- Some asymmetric algorithms don't use keys at all!





# Public-Key Cryptography








# Why Public-Key Cryptography?

- Can be used to:
  - **key distribution – secure communications without having to trust a KDC with your key (key exchange)**
  - **digital signatures –verify a message is come intact from the claimed sender (authentication)**
  - **encryption/decryption - secrecy of the communication (confidentiality)**
- Note:
  - **public-key cryptography simplifies but not eliminates the problem of trusted systems and key management**
  - **some algorithms are suitable for all uses,others are specific to one**
- Example of public key algorithms:
  - **RSA, which does encryption and digital signature**
  - **El Gamal and DSS, which do digital signature but not encryption**
  - **Diffie-Hellman, which allows establishment of a shared secret**
  - **zero knowledge proof systems, which only do authentication**





# Security of Public Key Schemes

- Security of public-key algorithms still relies on key size (as for secret-key algorithms)
- Like private key schemes brute force exhaustive search attack is always theoretically possible
  - **But keys used are much larger (>512bits)**
- A crucial feature is that the private key is difficult to determine from the public key
  - **security relies on a large enough difference in difficulty between easy (en/decrypt) and hard (cryptanalyse) problems**
  - **often the hard problem is known, its just made too hard to do in practise**
    - requires the use of very large numbers
    - hence is slow compared to private key schemes

# Rivest, Shamir, and Adleman (RSA)



## Rivest, Shamir, and Adleman

- by Rivest, Shamir & Adleman of MIT in 1977
- best known & widely used public-key scheme
- Based on exponentiation in a finite (Galois) field over integers modulo n
  - **nb. exponentiation takes $O((\log n)^3)$ operations (easy)**
- uses large integers (eg. 1024 bits)
- security due to cost of factoring large numbers
  - **nb. factorization takes $O(e^{\log n \log \log n})$ operations (hard)**
- The key length is variable
  - **long keys for enhanced security, or a short keys for efficiency**
- The plaintext block size (the chunk to be encrypted) is also variable
  - **The plaintext block size must be smaller than the key length**
  - **The ciphertext block will be the length of the key**
- RSA is much slower to compute than popular secret key algorithms like DES, IDEA, and AES





## RSA Algorithm

- First, you need to generate a public key and a corresponding private key:
  - **choose two large primes p and q (around 512 bits each or more)**
    - p and q will remain secret
  - **multiply them together (result is 1024 bits), and call the result n**
    - it's practically impossible to factor numbers that large for obtaining p and q
  - **choose a number *e* that is relatively prime (that is, it does not share any common factors other than 1) to $\phi(n)$**
    - since you know p and q, you know $\phi(n) = (p-1)(q-1)$
  - **your public key is KU =<e,n>**
  - **find the number *d* that is the multiplicative inverse of e mod $\phi(n)$**
  - **your private key is KR=<d,n> or KR=<d,p,q>**
- To encrypt a message m (< n), someone can use your public key
  - **$c = m^e \bmod n$**
- Only you will be able to decrypt c, using your private key
  - **$m = c^d \bmod n$**





## Why RSA Works

- Because of Euler's Theorem:
  - $a^{k\phi(n)+1} \bmod n = a$
    - where gcd(a,n)=1
- In RSA have:
  - **$n=p \cdot q$**
  - **$\phi(n)=(p-1)(q-1)$**
  - **carefully chosen e & d to be inverses mod $\phi(n)$**
    - hence $e \cdot d=1+k \cdot \phi(n)$ for some k
- Hence:
  $c^d = (m^e)^d = m^{1+k\phi(n)} = m \bmod n$

# RSA Key Setup

- Each user generates a public/private key pair by:
  - **selecting two large primes at random** `p,q`
  - **computing their system modulus** `n = p q`
    - note `ø(n)=(p−1)(q−1)`
  - **selecting at random the encryption key** `e`
    - where 1<`e<ø(n), gcd(e,ø(n))=1`
  - **solve following equation to find decryption key** `d`
    - `e d = 1 mod ø(n) and 0≤d≤n`
- Publish their public encryption key: KU={e,n}
- Keep secret private decryption key: KR={d,p,q}





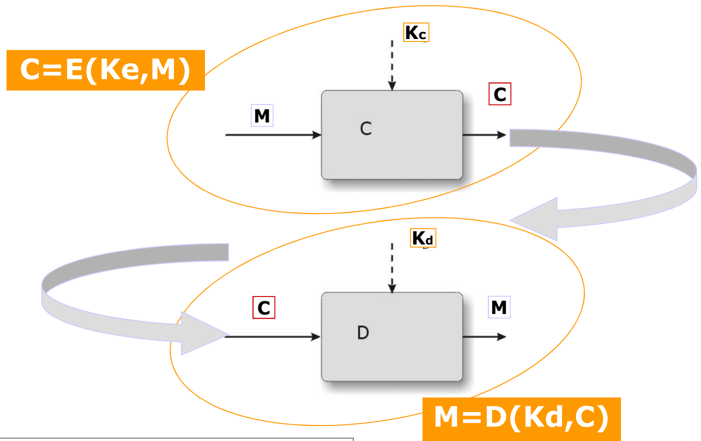
# RSA Use

- To encrypt a message M the sender:
  - **obtains public key of recipient KU=<e,n>**
  - **computes: c=me mod n, where 0≤m<n**
- To decrypt the ciphertext c the owner:
  - **uses their private key KR=<d,n>**
  - **computes: m=$c^d$ mod n**
- Note that the message m must be smaller than the modulus n (block if needed)





# RSA




**M** plaintext
**C** ciphertext
**Ke** encription key (e.g. public key, Ku or K+)
**Kd** decription key (e.g. private key, Kr or K-)





# RSA Example

RSA setup

- select primes: p=17 & q=11
- compute n = pq =17×11=187
- compute ø(n)=(p–1)(q-1)=16×10=160
- select e : gcd(e,160)=1; choose e=7
- determine d: de=1 mod 160 and d < 160 Value is d=23 since 23×7=161= 10×160+1
- publish public key KU={7,187}
- keep secret private key KR={23,187}={23,17,11}

# RSA Example (cont)

RSA encryption/decryption:

- given message M = 88 (nb. 88<187)
- encryption:
  **C = 88$^{7}$ mod 187 = 11**
- decryption:
  **M = 11$^{23}$ mod 187 = 88**





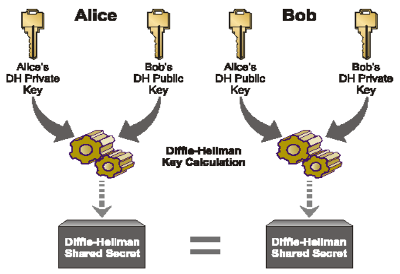
# RSA Security

- three approaches to attacking RSA:
  - **brute force key search (infeasible given size of numbers)**
  - **mathematical attacks (based on difficulty of computing ø(N), by factoring modulus N)**
  - **timing attacks (on running of decryption)**





# Diffie-Hellman








# Diffie-Hellman

- First public-key type scheme proposed
- By Diffie & Hellman in 1976 along with the exposition of public key concepts
  - **now know that James Ellis (UK CESG) secretly proposed the concept in 1970**
    - predates RSA
  - **less general than RSA: it does neither encryption nor signature**
- Is a practical method for public exchange of a secret key
  - **allows two individuals to agree on a shared secret (key)**
  - **It is actually used for key establishment**
- Used in a number of commercial products

# Diffie-Hellman Setup

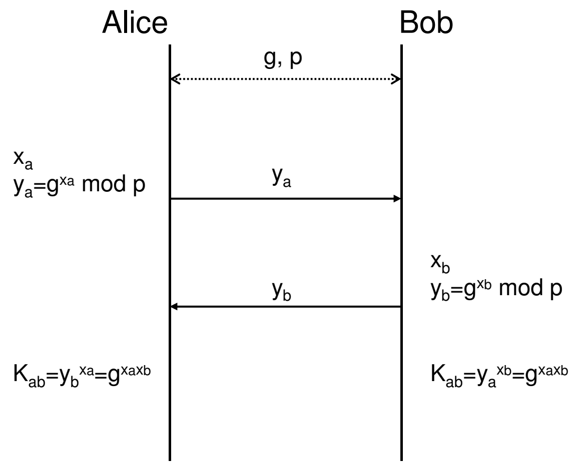
Diffie-Hellman setup:

- all users agree on global parameters:
  - **p = a large prime integer or polynomial**
  - **g = a primitive root mod p**
- each user (eg. A) generates their key
  - **chooses a secret key (number):** $x_A < p$
  - **compute their** public key: $y_A = g^{x_A} \bmod p$
- each user makes public that key $y_A$





# Diffie-Hellman Key Exchange








# Diffie-Hellman Key Exchange

Key exchange:

- Shared key $K_{AB}$ for users A & B can be computed as:

  $K_{AB} = g^{x_A \cdot x_B} \bmod p$

  $= y_A^{x_B} \bmod p$ (which B can compute)

  $= y_B^{x_A} \bmod p$ (which A can compute)
- $K_{AB}$ can be used as session key in secret-key encryption scheme between A and B
- Attacker must solve discrete log





# Diffie-Hellman - Example

- users Alice & Bob who wish to swap keys:
- agree on prime p=353 and g=3
- select random secret keys:
  - **A chooses** $x_A=97$, **B chooses** $x_B=233$
- compute public keys:
  - $y_A=3^{97} \bmod 353 = 40$ **(Alice)**
  - $y_B=3^{233} \bmod 353 = 248$ **(Bob)**
- compute shared session key as:

  $K_{AB}= y_B^{x_A} \bmod 353 = 248^{97} = 160$ **(Alice)**

  $K_{AB}= y_A^{x_B} \bmod 353 = 40^{233} = 160$ **(Bob)**

# Zero Knowledge Proof Systems

- Only do authentication
  - **prove that you know a secret without revealing the secret**
- RSA is a zero knowledge system
- There are zero knowledge systems with much higher performance
- Example (Isomorphic graphs):
  - **Alice defines two large (say 500 vertices) isomorphic graphs $G_A$ , $G_B$**
  - **$G_A$ and $G_B$ become public, but only Alice knows the mapping**
  - **to prove her identity to Bob, Alice find a set of isomorphic graphs $G_1, G_2, .. , G_k$**
  - **Bob divides the set into two subset $T_A$ and $T_B$**
  - **Alice shows to Bob the mapping between each $G_i \in T_A$ and $G_A$, and between each $G_j \in T_B$ and $G_B$**





# Security uses of public key cryptography

- Transmitting over an insecure channel
  - **each party has a <public key, private key> pair (Ku,Kr)**
  - **each party encrypts with the public key of the other party**

**encrypt $m_A$ using $Ku_B$ ⟶ decrypt $m_A$ using $Kr_B$**
**decrypt $m_B$ using $Kr_A$ ⟵ encrypt $m_B$ using $Ku_A$**

- Secure storage on insecure media
  - **encrypt with public key, decrypt with private key**
  - **useful when you can let third party to encrypt data**
- Peer Authentication
  - **authentication by proving the knowledge of the private key**

**encrypt r using $Ku_B$ ⟶ decrypt to r using $Kr_B$**
**⟵ r**





# Security uses of public key cryptography

- Key establishment
  - **e.g. Diffie-Hellman**
- Data authentication (Digital signature)
  - **based on cryptographic checksum**
  - **see later**
- Note
  - **Public key cryptography has specific algorithm for specific function such as**
    - data encryption
    - MAC/digital signature
    - key establishment
    - peer authentication





# Pros and cons of Public key cryptography

- Every users have to keep only one secret (the private key)
- Public keys of other users can verified through a trusted third party infrastructure (e.g. PKI)
- The total number of keys for *N* users is *2N*
  - **instead, with symmetric cryptography n(n-1)/2 keys are needed**

# Digital signature and digital certification



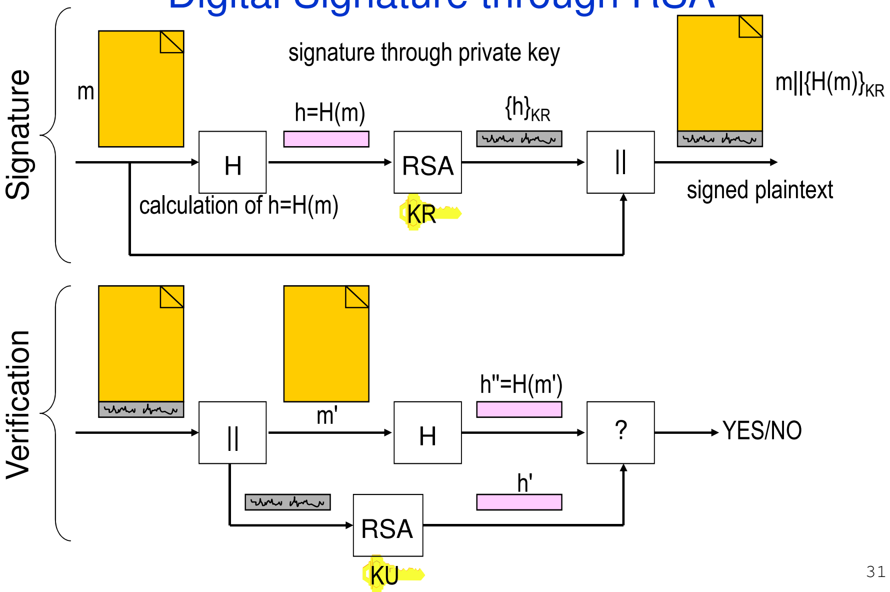
## Digital Signature

- Digital Signature is an application in which a signer, say "Alice," "signs" a message m in such a way that
  - **anyone can "verify" that the message was signed by no one other than Alice, and**
  - **consequently that the message has not been modified since she signed it**
- i.e. the message is a true and correct copy of the original
- The difference between digital signatures and conventional ones is that digital signatures can be mathematically verified
- The typical implementation of digital signature involves a message-digest algorithm and a public-key algorithm for encrypting the message digest (i.e., a message-digest encryption algorithm)





## Digital Signature through RSA

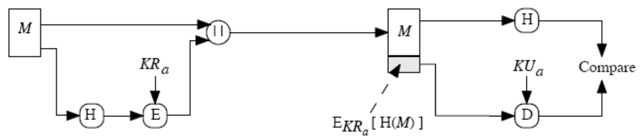




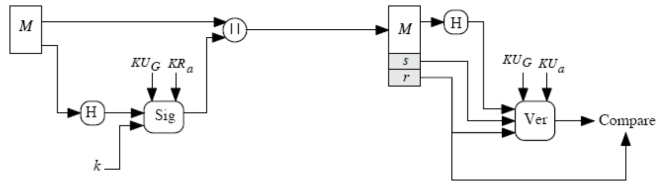



## Two Approaches to Digital Signatures

- RSA approach




- DSS approach



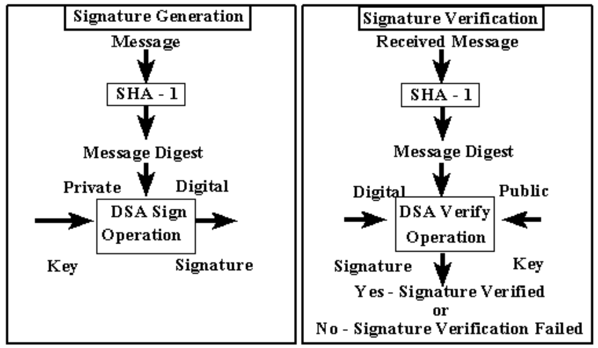
# Digital Signature Standard (DSS)

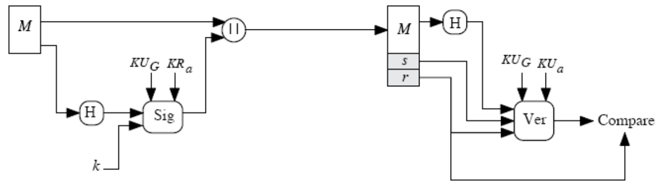
- DSS (Digital Signature Standard)
- Proposed by NIST (U.S. National Institute of Standards and Technology) & NSA in 1991
  - **FIPS 186**
- Based on an algorithm known as DSA (Digital Signature Algorithm)
  - **is a variant of the Elgamal (Taher Elgamal) scheme**
  - **uses 160-bit exponents**
  - **creates a 320 bit signature (160+160) but with 1024 (or more) bit security**
  - **uses SHA/SHS hash algorithm**
- Security depends on difficulty of computing discrete logarithms





# DSS Operations








# DSA Key Generation

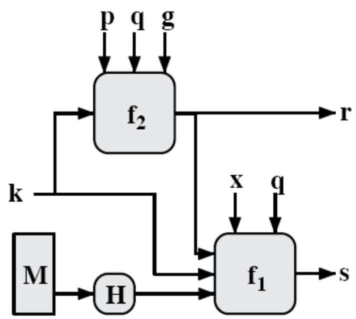
- have shared global public key values (p,q,g)
  - **L and N are respectively the key length and hash length**
    - L = 1024 or more, and multiple of 64 (e.g. 1024, 2048, 3072,..)
    - N = 160 or more (e.g. 160, 256, ..)
  - **take a large (L-bit) prime p**
  - **choose q, a N-bit prime factor of p-1**
    - in practice, you can choose q, and then p such that (p-1) is multiple of q
  - **choose g such that its multiplicative order modulo p is q**
    - in practice, $g=a^{(p-1)/q}$
    - for some arbitrary *a* with 1<*a*<p-1, with $a^{(p-1)/q} \bmod p > 1$
- choose $x<q$
- compute $y = g^x \bmod p$
- public key = (p,q,g,y)
- private key = x





# DSS Signing and Verifying

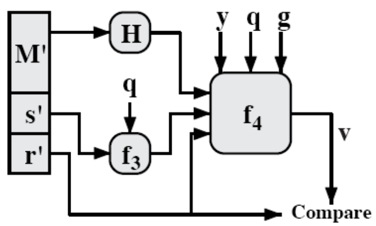
- Signing




s=f1(H(m),k,x,r,q)=($k^{-1}$(H(m)+xr)) mod q

r=f2(k,p,q,g)=($g^k$ mod p) mod q

- Verifying




w=f3(s,q)=$s^{-1}$ mod q

v=f4(p,q,g,y,H(m),w,r)=

=(($g^{H(m)w \bmod q}$ $y^{rw \bmod q}$) mod p) mod q

# DSA Signature Creation

- to sign a message M the sender generates:
  - **a random signature key k, k<q**
    - N.B.: k must be random, be destroyed after use, and never be reused
- computes the message digest:
  **h = SHA(M)**
- then computes signature pair:
  $r = (g^k \bmod p) \bmod q$
  $s = k^{-1}(h+x \cdot r) \bmod q$
- sends signature (r,s) with message M





# DSA Signature Verification

- having received M & signature (r,s)
- to verify a signature, recipient computes:
  $w = s^{-1} \bmod q$
  $v = (g^{hw \bmod q} y^{rw \bmod q} \bmod p) \bmod q$
- if v=r then signature is verified

- proof
  $v = (g^{hw \bmod q} y^{rw \bmod q} \bmod p) \bmod q =$
  $= (g^{w(h+xr) \bmod q} \bmod p) \bmod q =$
  $= (g^k \bmod p) \bmod q =$
  $= r$





# Digital Certification

- Digital certification is an application in which a certification authority "signs" a special message m containing
  - **the name of some user, say "Alice," and**
  - **her public key**

  in such a way that anyone can "verify" that the message was signed by no one other than the certification authority and thereby develop trust in Alice's public key
- The typical implementation of digital certification involves a signature algorithm for signing the special message