Chair for Network Architectures and Services
Department of Informatics
TU München – Prof. Carle

Network Security

Chapter 2 – Cryptography

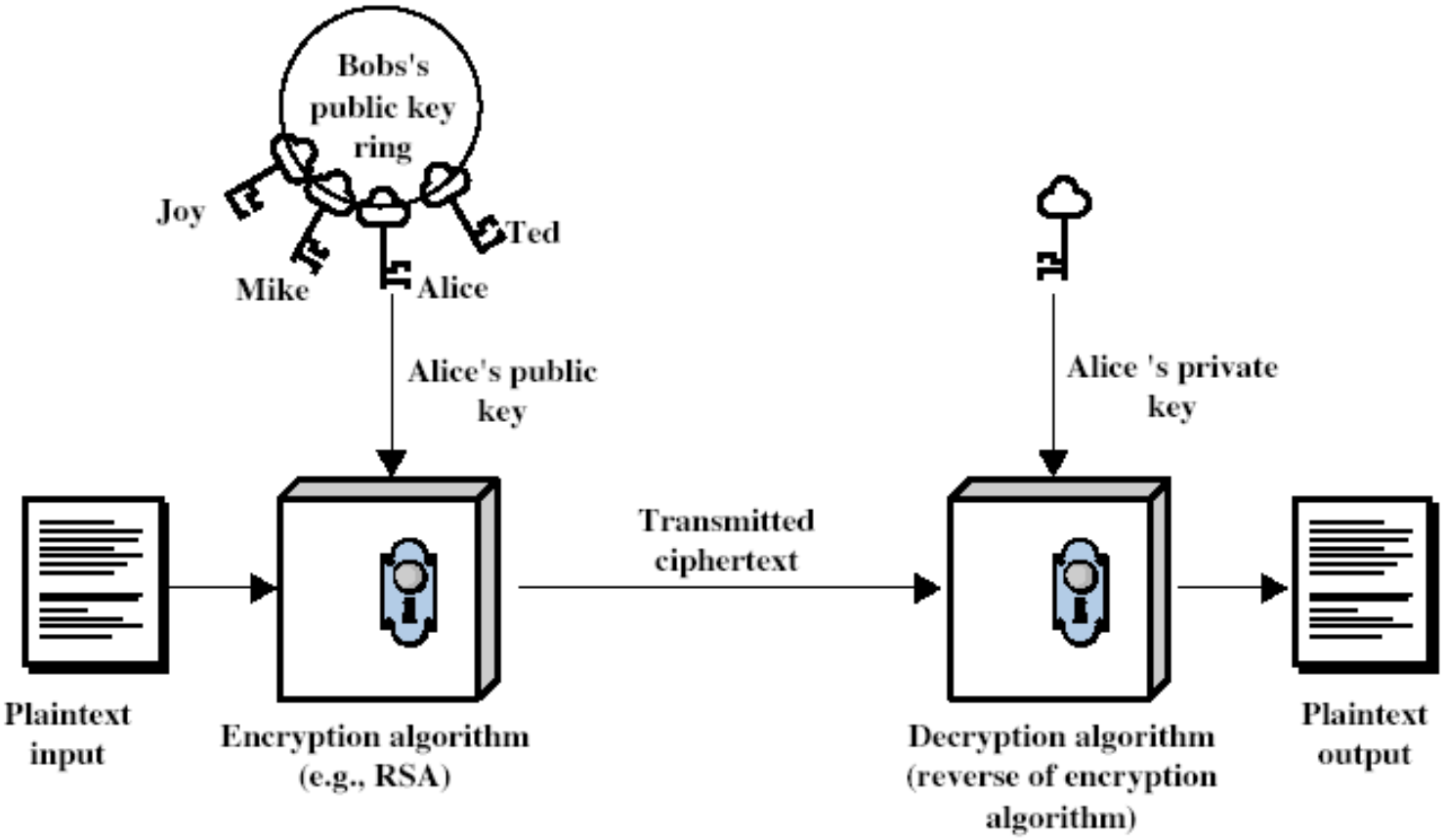
2.4 Public Key Cryptography

Acknowledgments

This course is based to a significant extend on slides provided by Günter Schäfer, author of the **book "Netzsicherheit - Algorithmische Grundlagen und Protokolle"**, available in German from **dpunkt Verlag**. The English version of the book is entitled “Security in Fixed and Wireless Networks: An Introduction to Securing Data Communications” and is published by Wiley is also available. We gratefully acknowledge his support.

The slides by Günter Schäfer have been partially reworked by Heiko Niedermayer, Ali Fessi, Ralph Holz and Georg Carle.

Encryption/Decryption using Public Key Cryptography



General Idea: encrypt with a publicly known key, but decryption only possible with a secret = private key

Public Key Cryptography

- General idea:
  - Use two different keys
    - a private key K_{priv}
    - a public key K_{pub}
  - Given a ciphertext $c = E(K_{pub}, m)$ and K_{pub} it should be *infeasible* to compute the corresponding plaintext without the private key K_{priv}:

    $m = D(K_{priv}, c) = D(K_{priv}, E(K_{pub}, m))$
  - It must also be infeasible to compute K_{priv} when given K_{pub}
  - The key K_{priv} is only known to the owner entity A
    → called A's *private key* $K_{priv\text{-}A}$
  - The key K_{pub} can be publicly known and is called A's *public key* $K_{pub\text{-}A}$

Public Key Cryptography

- Applications:
  - Encryption: If B encrypts a message with A's public key $K_{pub\text{-}A}$, he can be sure that only A can decrypt it using $K_{priv\text{-}A}$
  - Integrity check and digital signatures:
    - If B encrypts a message with his private key $K_{priv\text{-}B}$, everyone knowing B's public key $K_{pub\text{-}B}$ can read the message and know that B has sent it.

- Important:
  - If B wants to communicate with A, he needs to verify that he really knows A's public key and does not accidentally use the key of an adversary
  - Known as the "*binding of a key to an identity*"
  - Not a trivial problem – so-called Public Key Infrastructures are one "solution"
    - X.509
    - GnuPG Web of Trust

Public Key Cryptography

- Ingredients for a public key crypto system:
  - One-way functions: It is believed that there are certain functions that are easy compute, while the inverse function is very *hard* to compute
    - Real-world analogon: phone book
  - When we speak of *easy* and *hard*, we refer to certain complexity classes → more about that in crypto lectures and complexity theorey
  - For us: *Hard* means “infeasible on current hardware”
  - We know candidates, but have no proof for the existence of such functions
    - Existence would imply **P != NP**
- Special variant: Trap door functions
  - Same as one-way functions, but if a *second (“secret”) information* is known, then the inverse is *easy* as well
- Blueprint: use a trap-door function in your crypto system
- Candidates:
  - **Factorization problem**: basis of the RSA algorithm
    - Complexity class unknown, but assumed to be outside **P**
  - **Discrete logarithm problem**: basis of Diffie-Hellman and ElGamal
    - No polynomial algorithms known, assumed to be outside **P**

The Discrete Logarithm: DLog

- In the following, we will discuss another popular one-way / trap-door function: the discrete logarithm
- DLog is used in a number of ways
  - Diffie-Hellman Key Agreement Protocol
    - “Can I agree on a key with someone else if the attacker can read my messages?”
  - ElGamal
  - DLog problems can be transformed to Elliptic Curve Cryptography
    - We’ll discuss this later
- Now: more mathematics

Some Mathematical Background

- Theorem/Definition: *primitive root, generator*
  - Let p be prime. Then $\exists$ g $\in$ {1,2,…,p-1} such that

    $\{g^a \mid 1 \leq a \leq (p-1)\} = \{1,2,\ldots,p-1\}$ if everything is computed MOD p

  i.e. by exponentiating g you can obtain all numbers between 1 and (p -1)
  - For the proof see [Niv80a]
  - g is called a primitive root (or generator) of {1,2,…,p-1}
- Example: Let p = 7. Then 3 is a primitive root of {1,2,…,p-1}

  $1 \equiv 3^6$ MOD 7, $2 \equiv 3^2$ MOD 7, $3 \equiv 3^1$ MOD 7, $4 \equiv 3^4$ MOD 7,

  $5 \equiv 3^5$ MOD 7, $6 \equiv 3^3$ MOD 7

DLog: Some Mathematical Background

- Definition: discrete logarithm
  - Let *p* be prime, *g* be a primitive root of {1,2,…,p-1} and *c* be any element of {1,2,…,p-1}. Then $\exists$ *z* such that: $g^z \equiv c \text{ MOD } p$

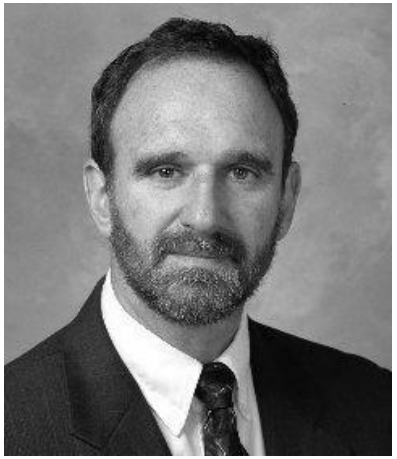
    z is called the discrete logarithm of *c* modulo *p* to the base *g*
  - Example: 6 is the discrete logarithm of 1 modulo 7 to the base 3 as $3^6 \equiv 1 \text{ MOD } 7$
  - The calculation of the discrete logarithm *z* when given *g*, *c*, and *p* is a computationally difficult problem and the asymptotical runtime of the best known algorithms for this problem is exponential in the bit-length of p

Diffie-Hellman Key Exchange (1)

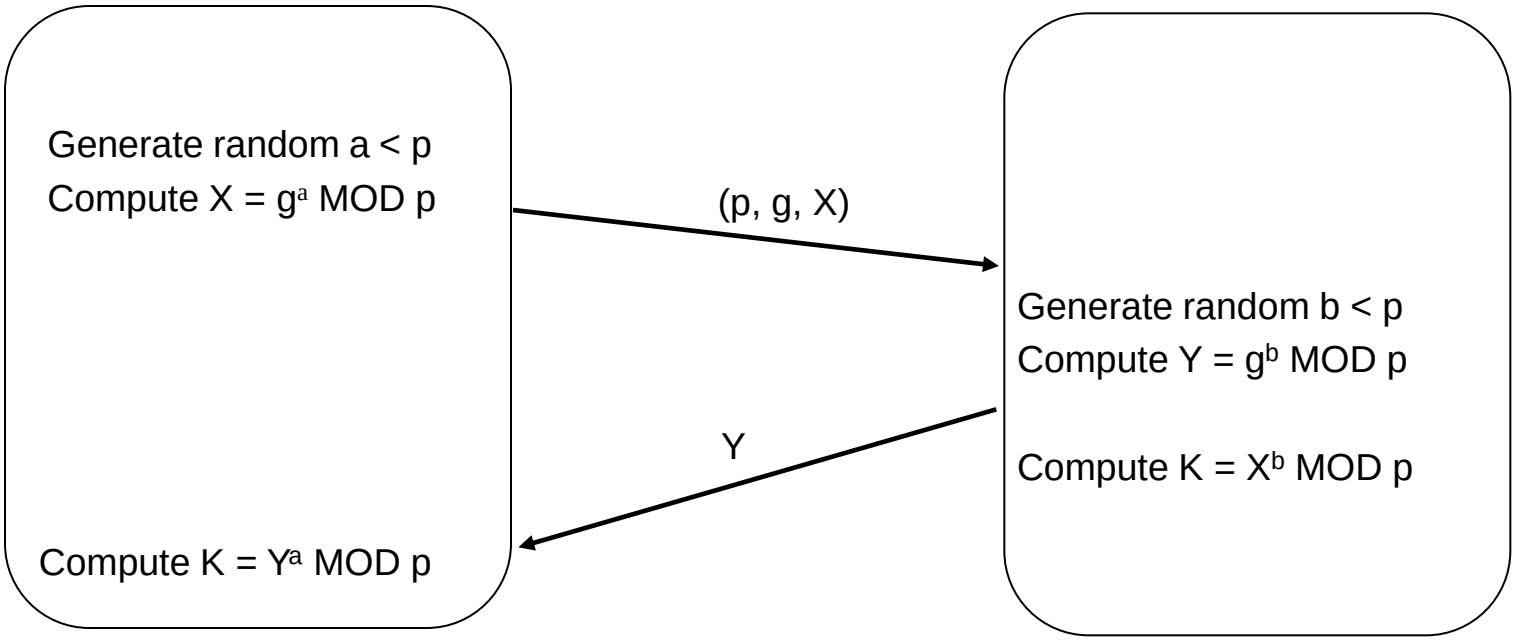
- The Diffie-Hellman key exchange was first published in the landmark paper [DH76], which also introduced the fundamental idea of asymmetric cryptography
- The DH exchange in its basic form enables two parties A and B to agree upon a *shared secret* using a public channel:
  - *Public channel* means, that a potential attacker can read all messages exchanged between A and B
  - It is important that A and B can be sure that the attacker is not able to alter messages as in this case he might launch a *man-in-the-middle attack*
  - The mathematical basis for the DH exchange is the problem of finding *discrete logarithms in finite fields*
  - The DH exchange is *not* an encryption algorithm.

Diffie-Hellman Key Exchange (2)

Whitfield Diffie



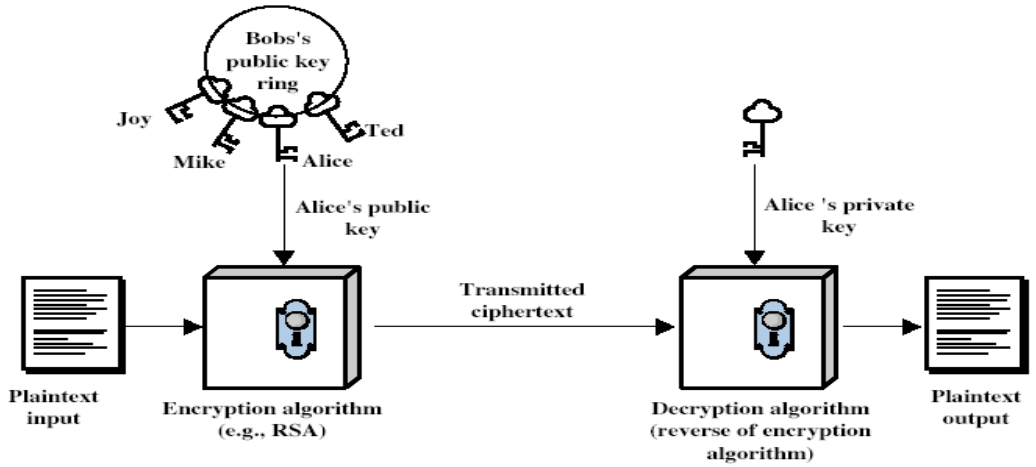
Martin E. Hellman



Diffie-Hellman Key Exchange (3)

- If Alice (*A*) and Bob (*B*) want to agree on a shared secret *K* and their only means of communication is a public channel, they can proceed as follows:
- *A* chooses a prime *p,* a primitive root *g* of {1,2,…,p-1} and a random number *x*
- *A* and *B* can agree upon the values *p* and *g* prior to any communication, or *A* can choose *p* and *g* and send them with his first message
- *A* chooses a random number *a:*
- *A* computes $X = g^a \ MOD \ p$ and sends *X* to *B*
- B chooses a random number *b*
- *B* computes $Y = g^b \ MOD \ p$ and sends *Y* to *A*
- Both sides compute the common secret:
  - *A* computes $K = Y^a \ MOD \ p$
  - *B* computes $K' = X^b \ MOD \ p$
  - As $g^{(a \cdot b)}$ MOD *p* = $g^{(b \cdot a)}$ MOD *p,* it holds: *K* = *K*'
- An attacker Eve who is listening to the public channel can only compute the secret *K*, if she is able to compute either *a* or *b* which are the discrete logarithms of *X* and *Y* modulo *p* to the base *g.*
- In essence, A and B have agreed on a key *without ever sending the key over the channel*
- This does not work anymore if an attacker is on the channel and can replace the values with his own ones

Can we use it for public key encryption and decryption?



- The TLS / SSL protocols support the use of Diffie-Hellman as Public Key algorithm. How does this work?
  - Here, the Diffie-Hellman values are considered to be constant.
  - Public key of Alice: g^a *MOD p*, g, p
  - Private key of Alice: a
  - Public key of Bob: g^b *MOD p*, g, p
  - Private key of Alice: b
  - To send a message: Bob takes Alice's public key, computes k = g^{ab} *MOD p and encrypts the message with k using symmetric key cryptography.*
  - *Alice needs to know Bob's public key and that the message is from Bob. Then she can generate k and decrypt it using symmetric encryption.*

Can we use it for signature?

- **Remember the problem**: If B encrypts a message with his private key $K_{priv\text{-}B}$, everyone knowing B's public key $K_{pub\text{-}B}$ can read the message and know that B has sent it (given that the message makes sense or is known because it was also sent in clear text).
- This means that Bob would have to use his secret DH value b in combination with anyone else's public key. But if it uses Alice's public key only Alice can obtain the shared key and no one else. So, in that way this cannot be done.

→ Diffie-Hellman is key agreement protocol and not a general-purpose public key algorithm.

- **Now, we want to discuss public key algorithms where we**
  - **… do not need symmetric encryption for encryption and decryption.**
  - **… can sign messages, so that all others can verify.**
  - **Therefore, we briefly introduce El Gamal on the next slide and then discuss RSA in more detail.**

El Gamal

- The ElGamal algorithm was invented by an Egyptian cryptographer "*Tahar El Gamal*". It uses the DLog problem like in Diffie-Hellman.

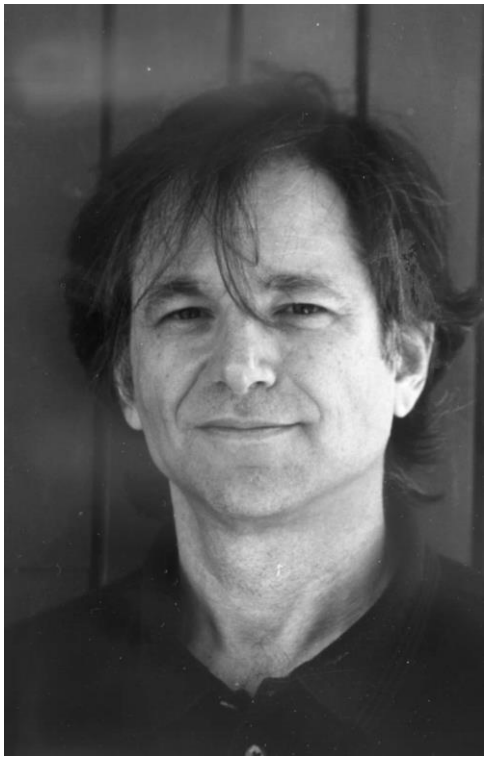
- Again, the public key of Alice: g^a *MOD p*, g, p
- Encyption
  - Bob choses a random z and computes g^z *MOD p*
  - *Message m → ciphertext c*
  - $c = m * g^{az}$ *MOD p*
  - *Bob sends* g^z *MOD p and the ciphertext c to Alice.*
- *Why does this work for signatures also?*
  - *Bob could use his private key b instead of Alice's public key.*
- Real world
  - ElGamal is a default in GnuPG
  - Digital Signature Algorithm (DSA) is based on ElGamal
  - As such, ElGamal/DSA is also part of Digital Signature Standard (NIST)

The RSA Public Key Algorithm

- The RSA algorithm was described in 1977 by R. Rivest, A. Shamir and L. Adleman [RSA78]

Ron Rivest

Adi Shamir



Leonard Adleman

- Note: Clifford Cocks in the UK came up with the same scheme in 1973 – but he worked for the government and it was treated classified and thus remained unknown to the scientific community.

Some Mathematical Background

- Definition: *Euler's Φ Function:*

  Let $\Phi(n)$ denote the number of positive integers $m < n$, such that m is relatively prime to n.

  → "m is relatively prime to n" = the greatest common divisor (gcd) of m and n is one.

- Let p be prime, then {1,2,…,p-1} are relatively prime to p, $\Rightarrow \Phi(p) = p-1$
- Let p and q be distinct prime numbers and $n = p \times q$, *then*

  $\Phi(n) = (p-1) \times (q-1)$

- Euler's Theorem:

  Let n and a be positive and relatively prime integers,

  $\Rightarrow a^{\Phi(n)} \equiv 1 \text{ MOD } n$

  - Proof: see [Niv80a]

The RSA Public Key Algorithm

- RSA Key Generation:
  - Randomly choose *p, q* distinct and large primes
    (really large: hundreds of bits = 100-200 digits each)
  - Compute $n = p \times q$, calculate $\Phi(n) = (p-1) \times (q-1)$ *(Euler's Φ Function)*
  - Pick $e \in Z$ such that $1 < e < \Phi(n)$ and e is relatively prime to $\Phi(n)$,
    i.e. $\gcd(e,\Phi(n)) = 1$
  - Use the extended Euclidean algorithm to compute d such that

    $e \times d \equiv 1 \text{ MOD } \Phi(n)$

  - The public key is (n, e)
  - The private key is d – this is the "trap door information"

The RSA Public Key Algorithm

- Definition: RSA function
  - Let p and q be large primes; let $n = p \times q$.
    Let $e \in \mathbb{N}$ be relatively prime to $\Phi(n)$.
  - Then RSA(e,n) := $x \rightarrow x^e$ MOD n
- Example:
  - Let M be an integer that represents the message to be encrypted, with M positive, smaller than n.
    - Example: Encode with <blank> = 99, A = 10, B = 11, ..., Z = 35
      So “HELLO” would be encoded as 1714212124.
      If necessary, break M into blocks of smaller messages: 17142 12124
  - To encrypt, compute: $C \equiv M^e$ MOD n
- Decryption:
  - To decrypt, compute: $M' \equiv C^d$ MOD n

The RSA Public Key Algorithm

- Why does RSA work:
  - As $d \times e \equiv 1 \text{ MOD } \Phi(n)$

  $\Rightarrow \exists\, k \in Z: \quad (d \times e) = 1 + k \times \Phi(n)$

  We sketch the "proof" for the case where M and n are relatively prime

  $M' \equiv C^d \text{ MOD } n$

  $\equiv (M^e)^d \text{ MOD } n$

  $\equiv M^{(e \times d)} \text{ MOD } n$

  $\equiv M^{(1 + k \times \Phi(n))} \text{ MOD } n$

  $\equiv M \times (M^{\Phi(n)})^k \text{ MOD } n$

  $\equiv M \times 1^k \text{ MOD } n$ (Euler's theorem*)

  $\equiv M \text{ MOD } n = M$

  - In case where M and n are not relatively prime, Euler's theorem can not be applied.
  - See [Niv80a] for the complete proof in that case.

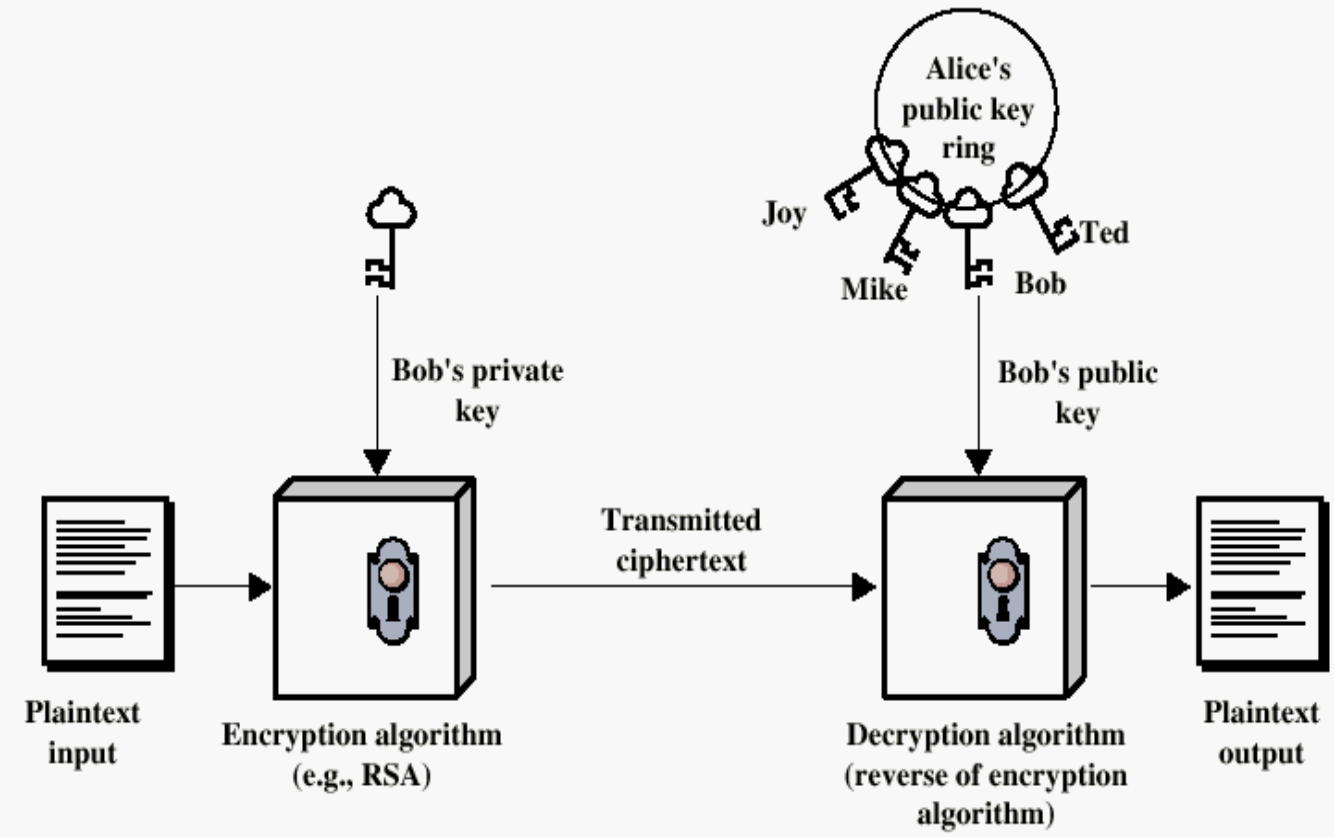
Using RSA

- All public-key crypto systems are much slower and more resource-consuming than symmetric cryptography
- Thus, RSA is usually used in a hybrid way:
  - Encrypt the actual message with symmetric cryptography
  - Encrypt the symmetric key with RSA
- Using RSA requires some precautions
  - Careful with choosing *p* and *q*: there are factorization algorithms for certain values that are very efficient
  - Generally, one also needs a *padding scheme* to prevent certain types of attacks against RSA
  - E.g. attack via Chinese remainder theorem: if the same clear text message is sent to *e* or more recipients in an encrypted way, and the receivers share the same exponent *e*, it is easy to decrypt the original clear text message
  - Padding also works against a Meet-in-the-middle attack
  - OAEP (from PKCS#1) is a well-known padding scheme for RSA

On the Security of RSA

- The security of the RSA algorithm lies in the presumed difficulty of factoring $n = p \times q$
- It is known that computing the private key from the public key is as difficult as the factorization
- It is unknown if the private key is really needed for efficient decryption (there might be a way without, only no-one knows it yet)
- RSA is one of the most widely used – and studied – algorithms
- We need to increase key length regularly, as computers become more powerful
  - 768 bit keys have already been factored
  - There are claims that 1024 bits may be routinely breakable in the not-so-far future
  - Current NIST recommendation is 2048 bit, should be on the safe side
  - More is better, but slower

Digital Signatures



- Signing = adding a proof of who has created a message, and that it has not been altered on the way
  - Who: authenticity
  - Not altered: integrity

Digital Signatures

- A wants to sign a message. General idea:
  - A computes a cryptographic *hash value* of her message: h(m)
    - Hashes are one-way functions, i.e. given h(m) it's infeasible to obtain m
    - We'll discuss hash functions soon
  - A encrypts h(m) with her *private* key $K_{priv\text{-}A}$ → Sig = $E_{K\_priv}(h(m))$
  - Given m, everyone can now
    - compute h(m)
    - Decrypt signature: D(E(h(m))) = h(m) and check if hash values are the same
  - If they match, A must have been the creator as only A knows the private key

Elliptic Curve Cryptography (ECC)

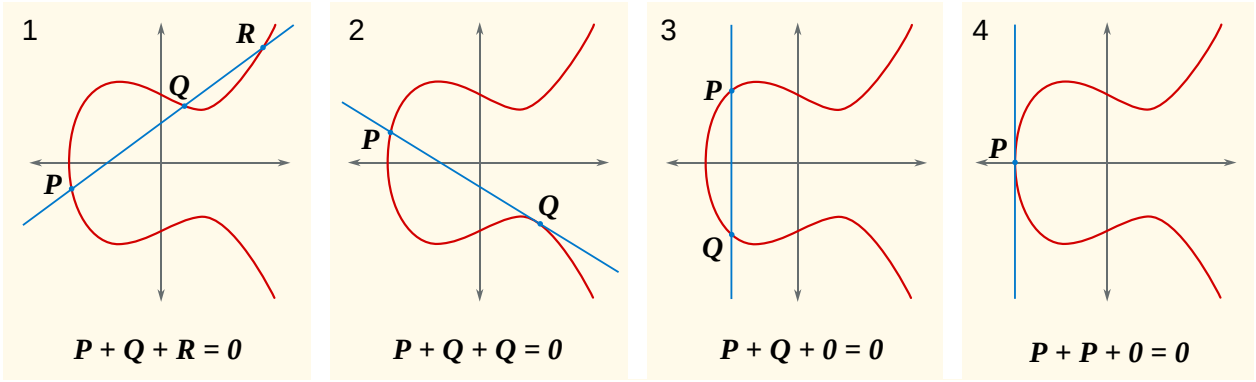
- Motivation: RSA is still probably the most widely implemented algorithm for Public Key Cryptography
  - Does public key cryptography need long keys with 1024-8192 bits?
  - Also, it is good to think of alternatives due to the developments in the area of primality testing, factorization and computation of discrete logarithms

  → Elliptic Curve Cryptocraphy (ECC)
- ECC is based on a finite field of points.
- Points are presented within a 2-dimensional coordinate system: (x,y)
- All points within the elliptic curve satisfy an equation of this type:

$$y^2 = x^3 + ax + b$$

Elliptic Curve Cryptography (ECC)

- Given this set of points an additive operator can be defined



- A multiplication of a point P by a number n is simply the addition of P to itself n times

$$Q = nP = P + P + \ldots + P$$

- The problem of determining n, given P and Q, is called the elliptic curve's discrete logarithm problem (ECDLP)
- The ECDLP is believed to be hard in the general class obtained from the group of points on an elliptic curve over a finite field

Elliptic Curve Cryptography (ECC)

- Any DLog-based algorithm can be turned into an ECC-based algorithm
- ECC problems are generally believed to be “harder”
  (though there is a lack of mathematic proofs)
- Allows us to have shorter key sizes
  → good for storage and transmission over networks
- New ECC curves are being developed and more and more applications use ECC.

Key Length (1)

- It is difficult to give good recommendations for appropriate and secure key lengths
- Hardware is getting faster
- So key lengths that might be considered as secure this year, might become insecure in 2 years
- Adi Shamir published in 2003 [Sham03] a concept for breaking 1024 bits RSA key with a special hardware within a year (hardware costs were estimated at 10 Millions US Dollars)
- Bruce Schneier recommends in [Fer03] a minimal length of 2048 bits for RSA “if you want to protect your data for 20 years”
- He recommends also the use of 4096 and up to 8192 bits RSA keys

Key Length (2)

- Comparison of the security of different cryptographic algorithms with different key lengths
  - Note: this is an informal way of comparing the complexity of breaking an encryption algorithm
  - So please be careful when using this table
  - Note also: a symmetric algorithm is supposed to have no significant better attack that breaks it than a brute-force attack

Symmetric	RSA	ECC
56	622	105
64	777	120
74	1024	139
103	2054	194
128	3214	256
192	7680	384
256	15360	512

Source [Bless05] page 89

Pitfall: Public key cryptography is not “symmetric”

- In contrast to symmetric cryptography, sender and receiver do not form a closed group with shared knowledge.
  - Public Key Cryptography
    - Encrypt with private key → everyone can read
    - Encrypt with a public key → only owner of key (receiver) can read
  - Symmetric Cryptography
    - Encrypt with shared key → only sender and receiver can read

Pitfall: Public key cryptography is not “symmetric”

- How to combine encryption and signature to protect messages m?
  - Case 1: A → B: $E_{kpub\text{-}B}(m, Sig_{kpriv\text{-}A}(m))$
    - Attack if destination B was not included in M: B → C: $E_{kpub\text{-}C}(m, Sig_{kpriv\text{-}A}(m))$
      - This attack is called “Surreptitious forwarding” : receiver B can decrypt, re-encrypt and replace receiver with some entity C and claim message was always for C.
    - Recommendation: always include receiver (and all other relevant entities like sender, etc.) in signature: A → B: $E_{kpubB}(B, m, Sig_{kpriv\text{-}A}(B, m))$
  - Case 2: A → B : $E_{kpub\text{-}B}(m)$, $Sig_{kpriv\text{-}A}(E_{kpub\text{-}B}(m))$ = sign encrypted data only
    - Attack: C → B: $E_{kpub\text{-}B}(m)$, $Sig_{kpriv\text{-}C}(E_{kpub\text{-}B}(m))$
      - Attacker C can just strip signature and replace it with his own – and receiver cannot determine *who* has sent the message. Note that attacker C cannot read plaintext m, yet he can sign it!
    - Recommendation: sign plaintext instead of ciphertexts, e.g.: A → B: $E_{kpub\text{-}B}(m)$,$Sig_{kpriv\text{-}A}(m)$ or include sender, receiver in m.
    - However, for symmetric case, encrypt then sign is best according to theory.

How to apply Public Key Cryptography?

Alice (A)

E(kpriv_A, h(m)),
E(kpub_B, k), E(k,m)

Bob (B)

- Usually used in combination with symmetric cryptography and hash functions
  - Symmetric cipher protects large data
  - Hash function computes fingerprint of m
  - Public Key Encryption (with public key of Bob) protects key
  - Public Key Signature (with private key of Alice) protects fingerprint
- Recent idea: provide API so that only data and public key have to be provided.
  - NaCL (Salt) Network and Cryptography Library
  - http://nacl.cr.yp.to
  - Usually, cryptography libraries require more programmer interaction and choices.

Summary

- Public key cryptography allows to use two different keys for:
  - Encryption / Decryption
  - Digital Signing / Verifying
- Some practical algorithms that are still considered to be secure:
  - RSA, based on the difficulty of factoring
  - Diffie-Hellman (a key agreement protocol)
- As their security is entirely based on the difficulty of certain number theory problems, algorithmic advances constitute their biggest threat
- Practical considerations:
  - Public key cryptographic operations are magnitudes slower than symmetric ones
  - Public cryptography is often just used to exchange a symmetric *session key* securely, which is on turn will be used for to secure the data itself.

Additional References

[Bless05] R. Bless, S. Mink, E.-O. Blaß, M. Conrad, H.-J. Hof, K. Kutzner, M. Schöller: "Sichere Netzwerkkommunikation", Springer, 2005, ISBN: 3-540-21845-9

[Bre88a] D. M. Bressoud. *Factorization and Primality Testing.* Springer, 1988.

[Cor90a] T. H. Cormen, C. E. Leiserson, R. L. Rivest. *Introduction to Algorithms.* The MIT Press, 1990.

[DH76] W. Diffie, M. E. Hellman. *New Directions in Cryptography.* IEEE Transactions on Information Theory, IT-22 , pp. 644-654, 1976.

[DSS] National Institute of Standards and Technology (NIST). FIPS 186--3, DRAFT Digital Signature Standard (DSS), March 2006.

[ElG85a] T. ElGamal. *A Public Key Cryptosystem and a Signature Scheme based on Discrete Logarithms.* IEEE Transactions on Information Theory, Vol.31, Nr.4, pp. 469-472, July 1985.

[Ferg03] Niels Ferguson, B. Schneier: “Practical Cryptography”, Wiley, 1st edition, March 2003

[Kob87a] N. Koblitz. *A Course in Number Theory and Cryptography.* Springer, 1987.

[Men93a] A. J. Menezes. *Elliptic Curve Public Key Cryptosystems.* Kluwer Academic Publishers, 1993.

[Niv80a] I. Niven, H. Zuckerman. *An Introduction to the Theory of Numbers.* John Wiley & Sons, 4th edition, 1980.

[Resc00] Eric Rescorla, „SSL and TLS: Designing and Building Secure Systems“, Addison-Wesley, 2000

[RSA78] R. Rivest, A. Shamir und L. Adleman. *A Method for Obtaining Digital Signatures and Public Key Cryptosystems.* Communications of the ACM, February 1978.

[Sham03] Adi Shamir, Eran Tromer, “On the cost of factoring RSA-1024”, RSA Cryptobytes vol. 6, 2003