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

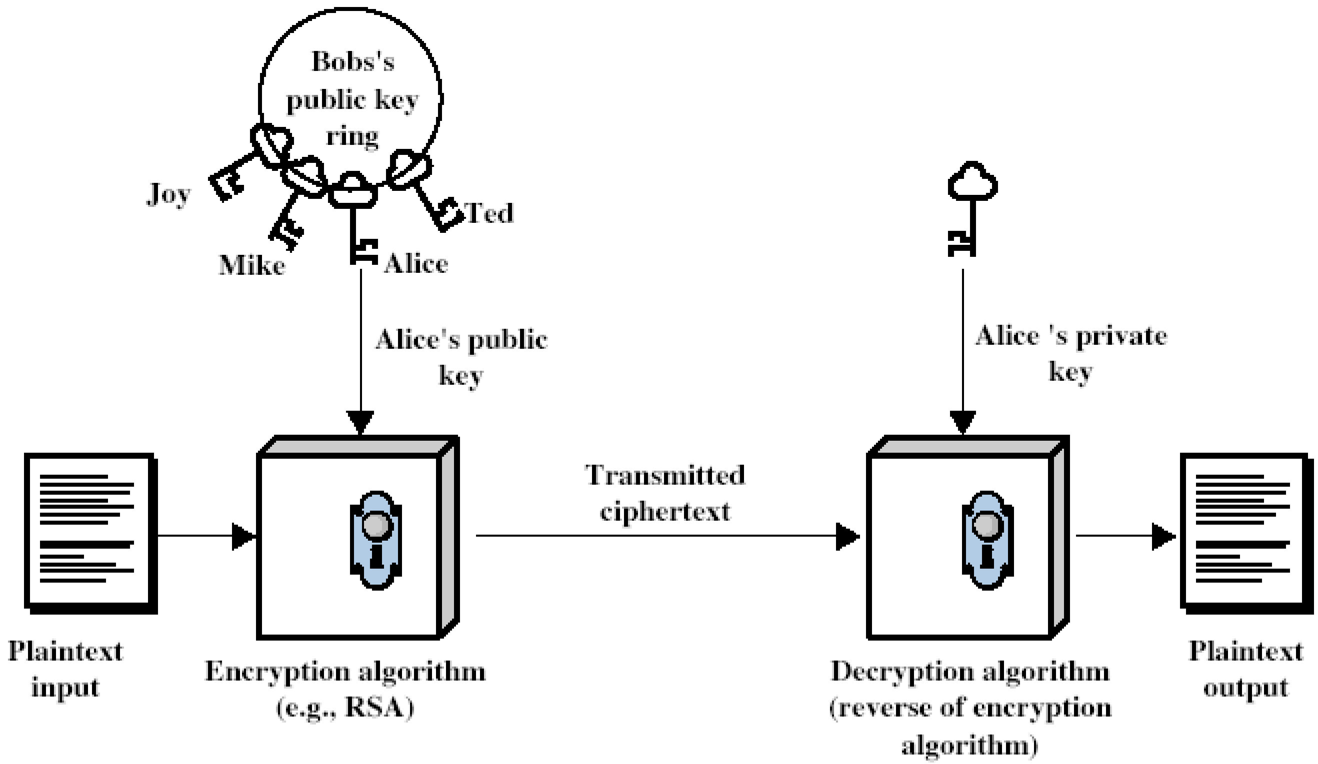
Network Security

Chapter 4
Public Key Cryptography

“However, prior exposure to discrete mathematics will help the reader to appreciate the concepts presented here.”

E. Amoroso in another context [Amo94]

Encryption/Decryption using Public Key Cryptography



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:

    $m = D(K_{priv}, c) = D(K_{priv}, E(K_{pub}, m))$
    - This implies that it should be infeasible to compute K_{priv} when given K_{pub}
  - The key K_{priv} is only known to one entity A and is called A's *private key* $K_{priv\text{-}A}$
  - The key K_{pub} can be publicly announced and is called A's *public key* $K_{pub\text{-}A}$

Public Key Cryptography (4)

- 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 $K_{priv\text{-}A}$
  - Signing:
    - If A encrypts a message with his own private key $K_{priv\text{-}A}$, everyone can verify this signature by decrypting it with A's public key $K_{pub\text{-}A}$
  - Attention: It is essential, that if B wants to communicate with A, it needs to verify that he really knows A's public key and not the key of an adversary!

Public Key Cryptography (5)

- Design of asymmetric cryptosystems:
  - Difficulty: Find an algorithm and a method to construct two keys K_{priv} , K_{pub} such that it is not possible to decipher $E(K_{pub} , m)$ with the knowledge of K_{pub}
  - Constraints:
    - The key length should be “manageable”
    - Encrypted messages should not be arbitrarily longer than unencrypted messages (we would tolerate a small constant factor)
    - Encryption and decryption should not consume too much resources (time, memory)

Public Key Cryptography (6)

- Basic idea:
  Take a problem in the area of mathematics or computer science that is *hard* to solve when knowing only K_{pub} ,
  but *easy* to solve when knowing K_{priv}

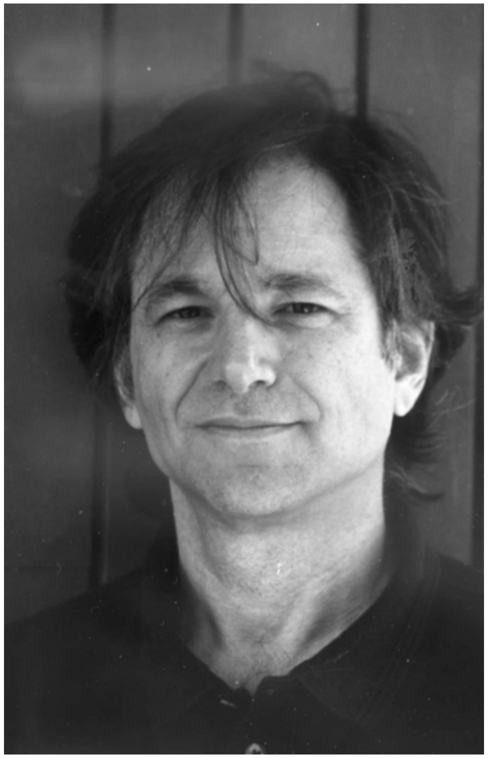
  - Knapsack problems: basis of first working algorithms, which were unfortunately almost all proven to be insecure
  - **Factorization problem**: basis of the RSA algorithm
  - **Discrete logarithm problem**: basis of Diffie-Hellman and ElGamal

The RSA Public Key Algorithm (1)

- The RSA algorithm was invented in 1977 by R. Rivest, A. Shamir and L. Adleman [RSA78] and is based on Euler's Theorem.

Ron Rivest

Adi Shamir



Leonard Adelaide

Some Mathematical Background

- Definition: *Euler's Φ Function:*

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

  "*m* is relatively prime to *n*" i.e. the greatest common divisor (gcd) between *m* and *n* is one.

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

- RSA Key Generation:
  - Randomly choose *p, q* distinct large primes (e.g. both *p* and *q* have 100 to 200 digits each)
  - Calculate $n = p \times q$, calculate $\Phi(n) = (p-1) \times (q-1)$ *(Euler's Φ Function)*
  - Pick $e \in Z$ such as $1 < e < \Phi(n)$ and *e* is relatively prime to $\Phi(n)$, i.e. *e* and $\Phi(n)$ do not have a greater common divisor greater than 1
  - Using the extended Euclidean algorithm to compute *d,* such that:

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

    i.e., there exists $u \in Z$ such as $e \times d = 1 + u \times \Phi(n)$

  - The public key is (n, e)
  - The private key is *d*

The RSA Public Key Algorithm (3)

- Encryption:
  - 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: $C \equiv M^e \text{ MOD } n$

- Decryption:
  - To decrypt, compute: $M' \equiv C^d \text{ MOD } n$

The RSA Public Key Algorithm (4)

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

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

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

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

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

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

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

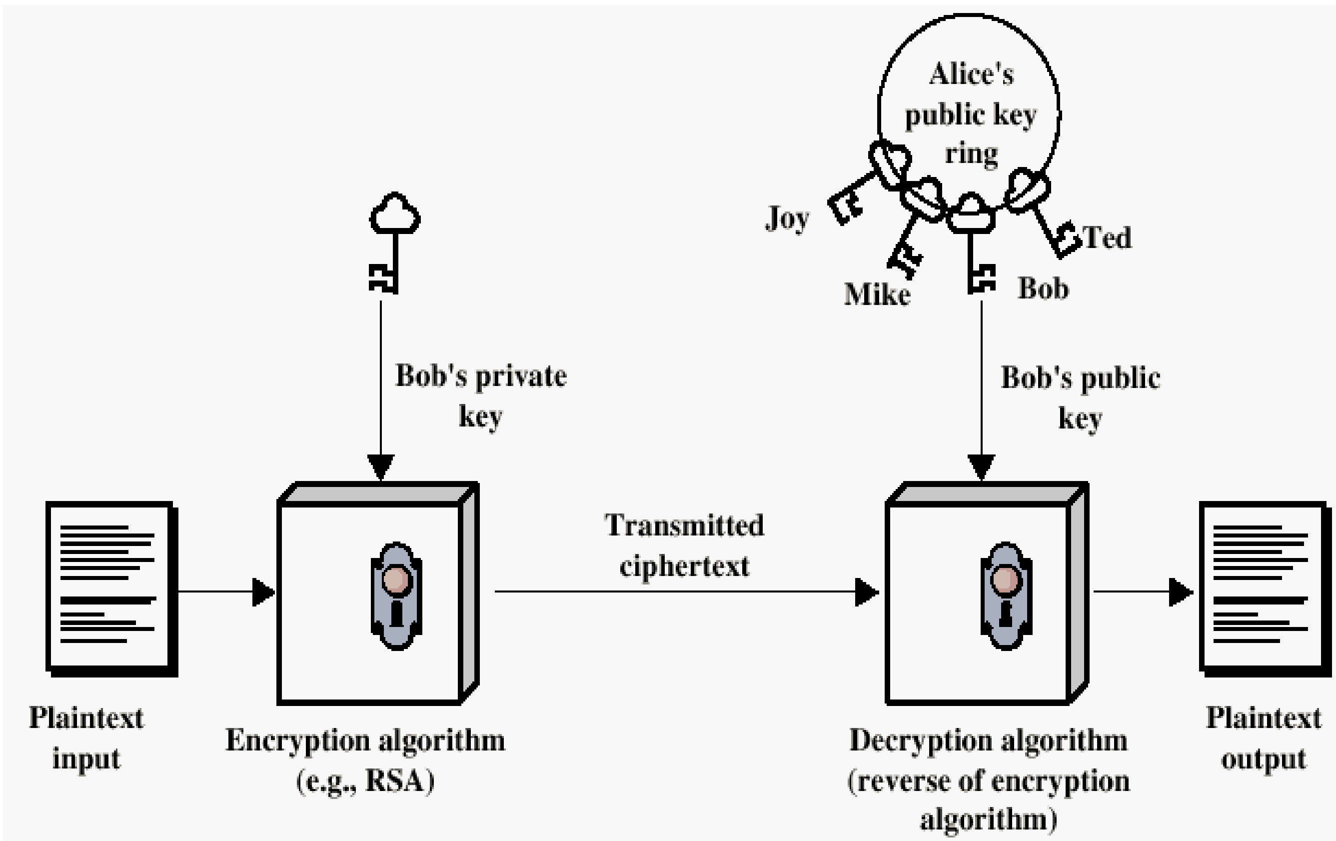
$\equiv M \times 1^k \text{ MOD } n$ (Euler's Theorem)

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

RSA Security

- The security of the RSA algorithm lies in the difficulty of factoring $n = p \times q$, while it is easy to compute $\Phi(n)$ and then d, when p and q are known.
- This class will not teach why it is difficult to factor large n's, as this would require to dive deep into mathematics
- Please be aware that if you choose p and q in an “unfortunate” way, there might be algorithms that can factor more efficiently and your RSA encryption is not at all secure.
- Moral: If you are to implement RSA by yourself, ask a cryptographer to check your design
- Even better: If you have the choice, you should not implement RSA by yourself and instead a published open source implementation that is validated and well understood.

Digital Signatures using RSA



Digital Signatures using RSA

- As $(d \times e) = (e \times d)$, the operation also works in the opposite direction, i.e. it is possible to encrypt with *d* and decrypt with *e*

- This property allows to use the same keys *d* and *e* for:
  - Encryption:
    When *B* encrypts a message using *e*, which is public, only *A* can decrypt it using *d*.
  - Digital Signatures:
    When *A* encrypts a message using *d*, which is private, *B* can decrypt it using *e*.
    In this case, *B* can be sure that it is *A* who sent the message,
    since it is assumed that only *A* possesses the private key *d.*

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 E (E stands for eavesdropper) 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.

Some Mathematical Background (1)

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

    $\{g^a \mid 1 \leq a \leq (p\text{-}1)\} = \{1,2,\ldots,p\text{-}1\}$

  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}

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

  $5 \equiv 3^5 \text{ MOD } 7,\ 6 \equiv 3^3 \text{ MOD } 7$

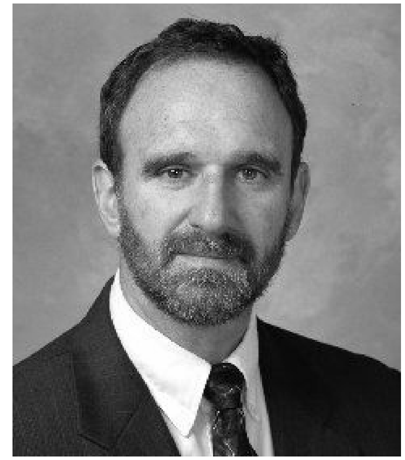
Some Mathematical Background (2)

- 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: $g^z \equiv c$ 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 $3^6 \equiv 1$ 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 (2)

Whitfield Diffie



Martin E. Hellman

Generate random $a < p$

Compute $X = g^a$ MOD p

(p, g, X)

Generate random $b < p$

Compute $Y = g^b$ MOD p

Compute $K = X^b$ MOD p

Y

Compute $K = Y^a$ MOD p

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} (how to find a primitive root g is not treated here), 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.*

The El Gamal Algorithm

- The ElGamal algorithm was invented by an Egyptian cryptographer "*Tahar El Gamal*"

- The ElGamal algorithm can be used for both, encryption and digital signatures (see also [ElG85a])
- Like the DH exchange it is based on the difficulty of computing discrete logarithms in finite fields

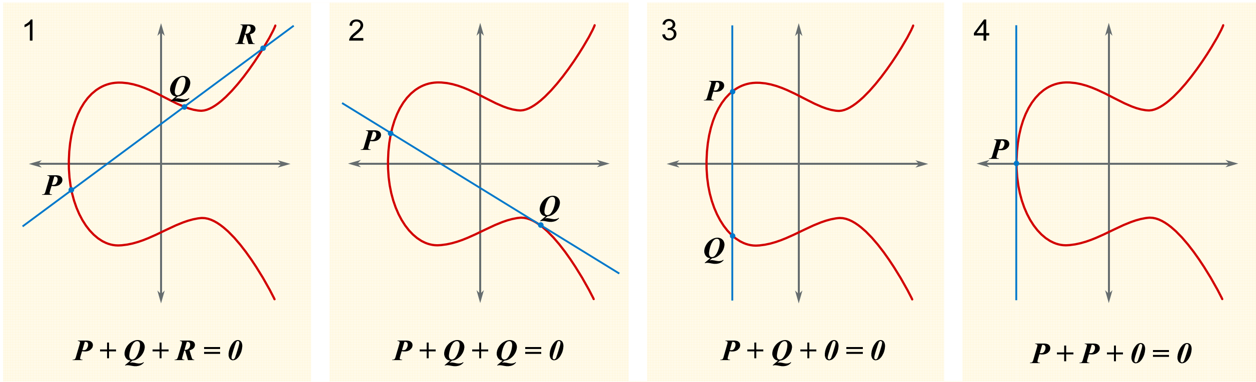
Elliptic Curve Cryptography (ECC)

- Motivation: we assume that RSA is currently the most widely implemented algorithm for Public Key Cryptography.
- However, an alternative is required due to the developments in the area of primality testing, factorization and computation of discrete logarithms that led to techniques that allow to solve these problems in a more efficient way
- 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
- The use of ECC is getting more and more widespread
  - e.g. the implementation of the SSL/TLS protocol „OpenSSL“
  - One of the advantages compared to RSA and El Gamal is the compact key length

Digital Signature Standard (DSS)

- Recall that the NIST has standardized algorithms for symmetric encryption, it has also standardized algorithms for digital signature generation
- that can be used for the protection of messages, and for the verification and validation of those digital signatures.
- Three techniques are allowed:
  - Digital Signature Algorithm (DSA)
    - Security is based on the difficulty of the discrete logarithm problem
    - Builds on the El Gamal digital algorithm
  - RSA
  - Elliptic Curve Digital Signatue Algorithm (ECDSA)
- Furthermore, a cryptographic hash function (SHA-1) is used for generating a hash value of the message to be signed.
- E.g. Digital signature of message M using RSA:

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

Key Length (1)

- It is difficult to give good recommendations for appropriate and secure key lengths
- Hardware is getting faster (Remember Moore's law)
- 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: symmetric algorithms are supposed to have no better attack that breaks it other than brute-force

Symmetric	RSA/El Gamal	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

RSA vs. DSA Performance

Algorithm	Keysize	Signs/s	Verify/s
RSA	512	342	3287
DSA	512	331	273
RSA	1024	62	1078
DSA	1024	112	94
RSA	2048	10	320
DSA	2048	34	27

Digital signature performance (Pentium II 400/OpenSSL)
Source [Resc00] page: 182

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 (not an asymmetric algorithm, but a key agreement protocol)
  - ElGamal, like DH based on the difficulty of computing discrete logarithms
- As their security is entirely based on the difficulty of certain number theory problems, algorithmic advances constitute their biggest threat
- One of the reasons why considerable effort and progress has been seen in ECC
- Practical considerations:
  - Public key cryptographic operations are about 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, 4^{th} 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