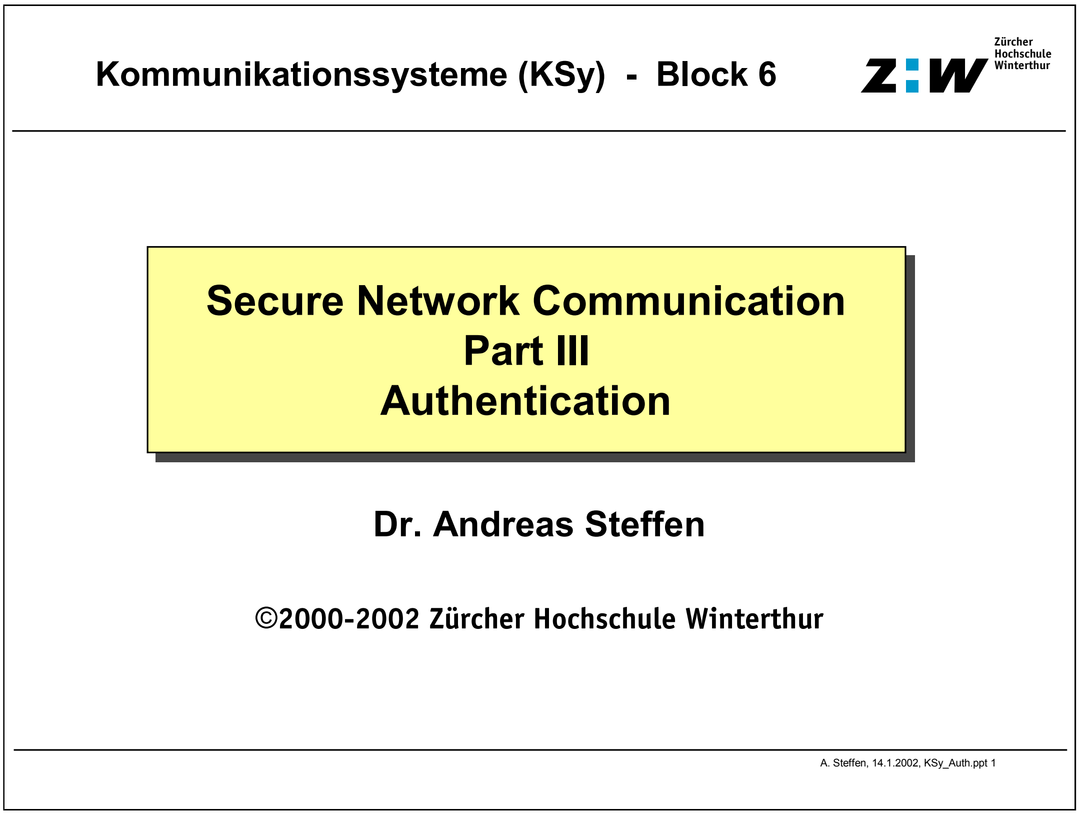


Kommunikationssysteme (KSy) - Block 6

Secure Network Communication
Part III
Authentication

Dr. Andreas Steffen

Data Integrity

- Message digests base on one-way hash functions
- Popular hash functions: MD5 and SHA
- Basic structure of the MD5 / SHA one-way hash functions
- Message authentication codes (MAC)
- Basic structure of a keyed one-way hash function
- Digital signatures based on public key cryptosystems
- Forging documents
- The birthday paradox
- Birthday attacks against hash functions

Authentication Schemes

- Insecure Authentication based on passwords
- Secure Authentication based on challenge/response protocols
- Challenge/response protocols based on MACs
- Challenge/response protocols based on digital signatures

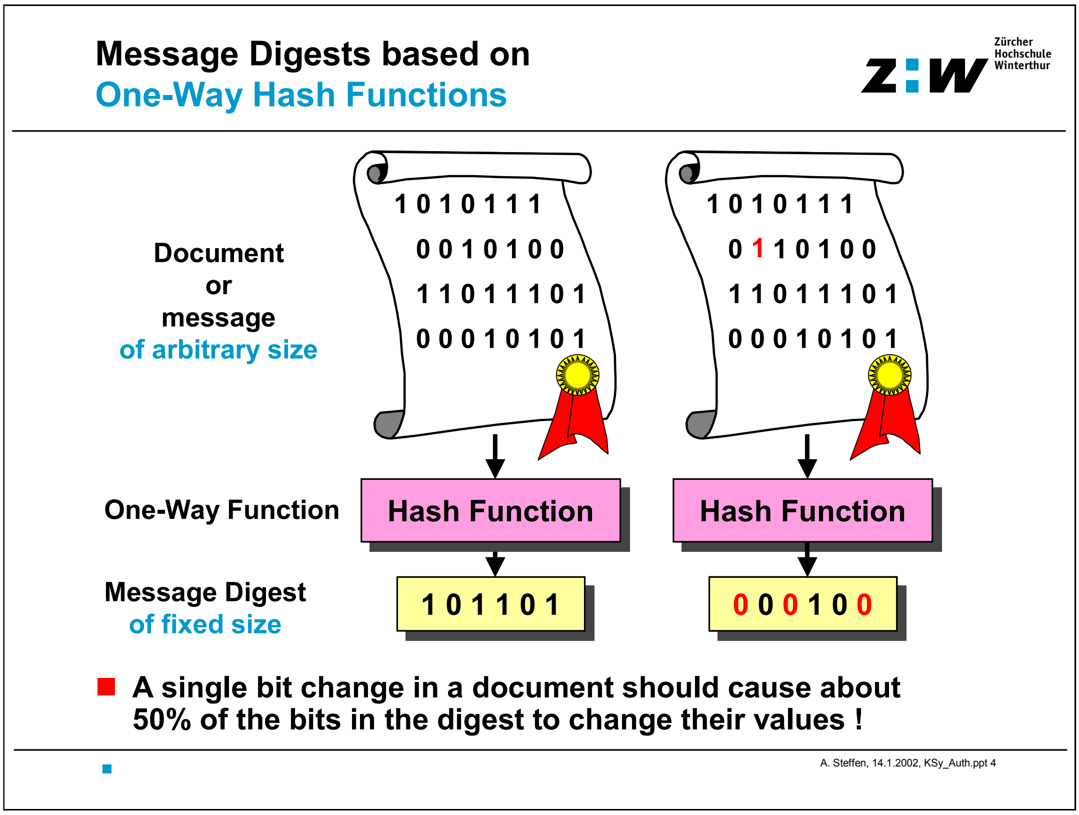
Certificates

- Trust models
- PGP web of trust
- Trust hierarchy based on certification authorities
- X.509 certificates
- Data structure and encoding of X.509 certificates
- X.509 certificate handling by the Netscape 4.7x Browsers
- X.509 certificate handling by the Microsoft Internet Explorer 5.x
- Public Key Infrastructure (PKI)

Data Integrity

Integrity of Documents and Messages

- **Detection of corrupted documents and messages**
  - **Detection of bit errors caused by unreliable transmission links or faulty storage media.**
  - **Solution: Message Digest acting as a unique fingerprint for the document (similar function as CRC).**
- **Protection against unauthorized modification**
  - **Without protection a forger could create both an alternative document and its corresponding correct message digest.**
  - **Symmetric Key Solution: Message Authentication Code (MAC) formed by using a keyed message digest function.**
  - **Asymmetric Key Solution: Digital Signature formed by encrypting the message digest with the document author‘s private key.**



Message Digests based on One-Way Hash Functions

Document or message of arbitrary size

1 0 1 0 1 1 1	1 0 1 0 1 1 1
0 0 1 0 1 0 0	0 1 1 0 1 0 0
1 1 0 1 1 1 0 1	1 1 0 1 1 1 0 1
0 0 0 1 0 1 0 1	0 0 0 1 0 1 0 1

One-Way Function: Hash Function | Hash Function

Message Digest of fixed size: 1 0 1 1 0 1 | 0 0 0 1 0 0

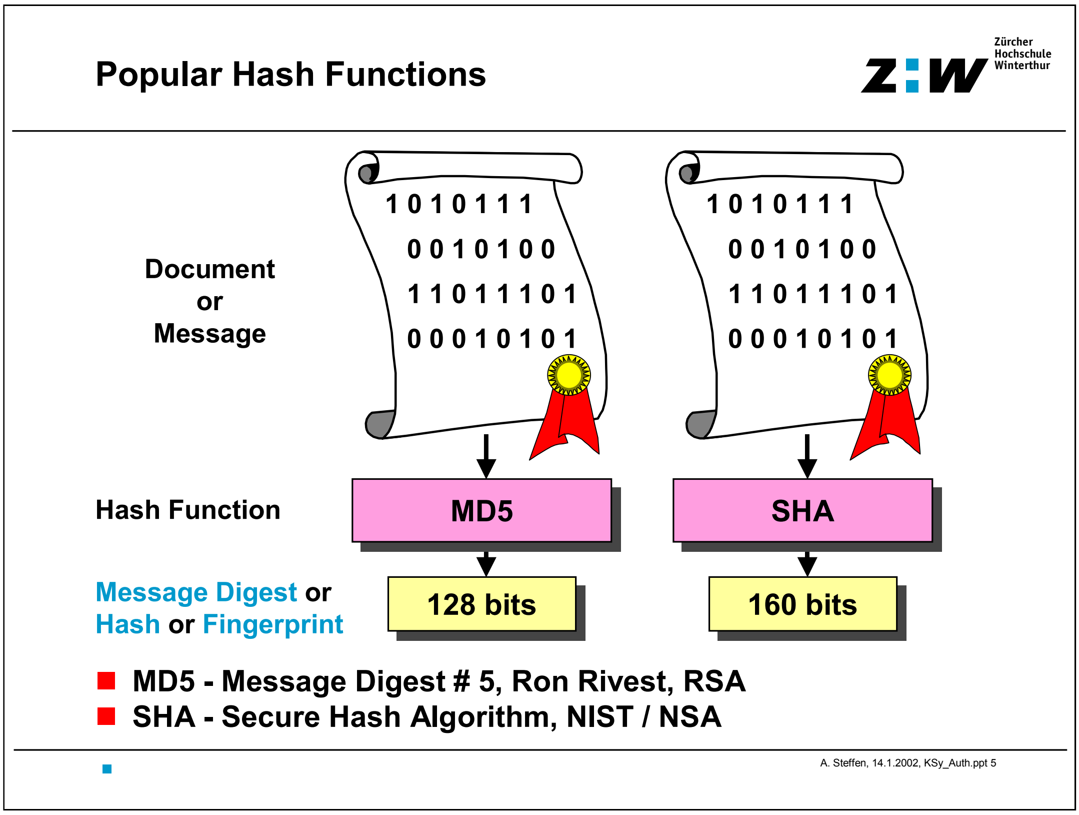
- A single bit change in a document should cause about 50% of the bits in the digest to change their values !

Message Digests

- A message digest of a fixed size acts as a **unique fingerprint** for an arbitrary-sized message, document or packed software distribution file. With a common digest size of 128 .. 256 bits, about 10^{38} .. 10^{77} different fingerprint values can be represented. If on every day of the 21th century 10 billion people wrote 100 letters each, this would amount to $3.65 \cdot 10^{16}$ documents, only. So if each of these letters had its individual fingerprint, only a tiny percentage of all possible values would be used.

One-Way Hash Functions

- For the computation of message digests special **one-way hash functions** are used. A good hash function should have the following properties:
- The computation of message digests should be fast and efficient, allowing the hashing of messages several gigabytes in size. Since a document is usually much larger than its hash value, the mapping is a many-to-one function. For each specific hash value there potentially exist many documents possessing this fingerprint.
- It should be practically infeasible to find a document that produces a given fingerprint. This is why a good hash function is called **one-way**.
- The message digest value should depend on every bit of the corresponding message. If a single bit of the original message changes its value, or one bit is added or deleted, then about 50% of the digest bits should change their values in a random fashion. A good hash function achieves a pseudo-random message-to-digest mapping, causing two nearly identical messages to have totally different hash values.
- Due to the pseudo-random nature of a good hash function and the enormous number space of possible hash values, it also becomes quite impossible that two distinct messages will ever produce the same digest value. So for all of today‘s practical applications we can regard the output of a good hash function as a **quasi-unique fingerprint** of the hashed message.



Popular Hash Functions

Document or Message

1 0 1 0 1 1 1
0 0 1 0 1 0 0
1 1 0 1 1 1 0 1
0 0 0 1 0 1 0 1

Hash Function

MD5 → 128 bits

SHA → 160 bits

Message Digest or Hash or Fingerprint

- MD5 - Message Digest # 5, Ron Rivest, RSA
- SHA - Secure Hash Algorithm, NIST / NSA

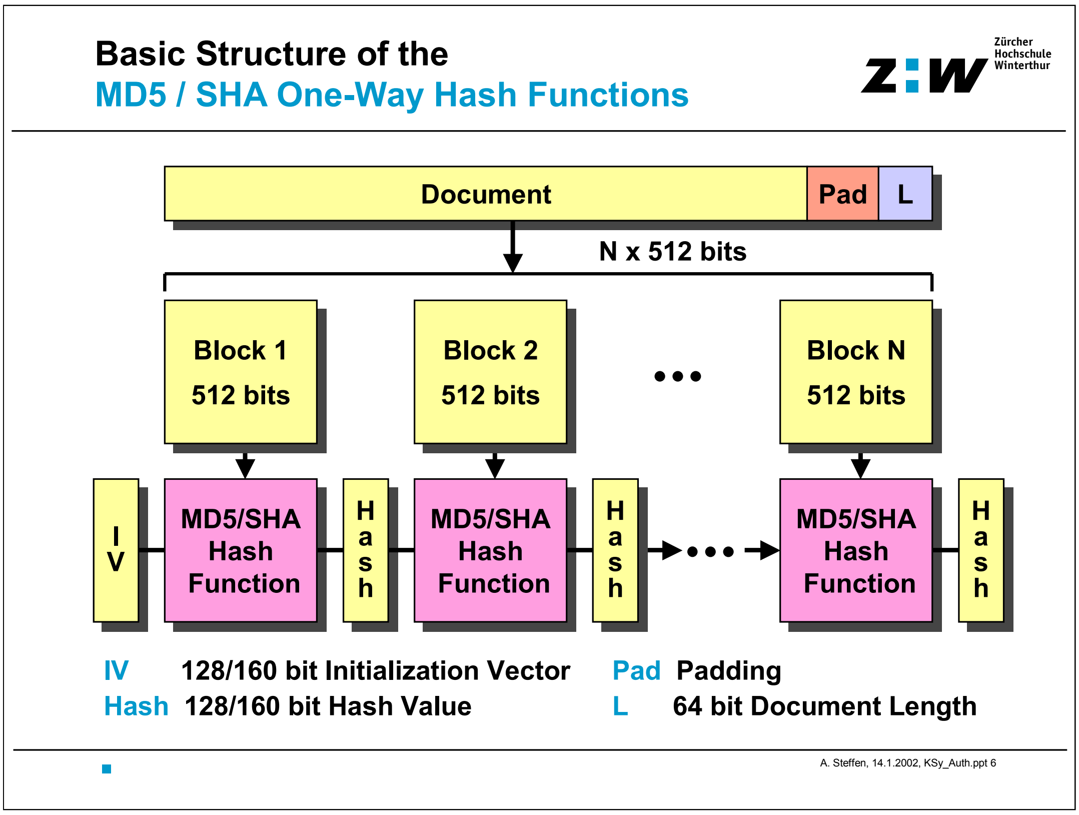
A. Steffen, 14.1.2002, KSy_Auth.ppt 5

MD5 – Message Digest #5

- Invented by Ron Rivest (the R in RSA) of RSA Security Inc.
- MD5 computes a hash value of 128 bits (16 bytes) out of an arbitrary-sized binary document.

SHA – Secure Hash Algorithm

- Developped by the US National Institute of Standards and Technology (NIST) with the assistance of the National Security Agency (NSA).
- SHA-1 computes a hash value of 160 bits (20 bytes) out of an arbitrary-sized binary document. The algorithm is similar to MD5.
- SHA-1 is more secure than MD5 due to its increased hash size.
- An improved SHA-2 algorithm with hash sizes of 256 bits (32 bytes), 384 bits (48 bytes) and 512 bits (64 bytes) was published by NIST in October 2000 to keep up with the increased key sizes of the Advanced Encryption Standard (AES).



Basic Structure of the MD5 / SHA One-Way Hash Functions

IV 128/160 bit Initialization Vector

Hash 128/160 bit Hash Value

Pad Padding

L 64 bit Document Length

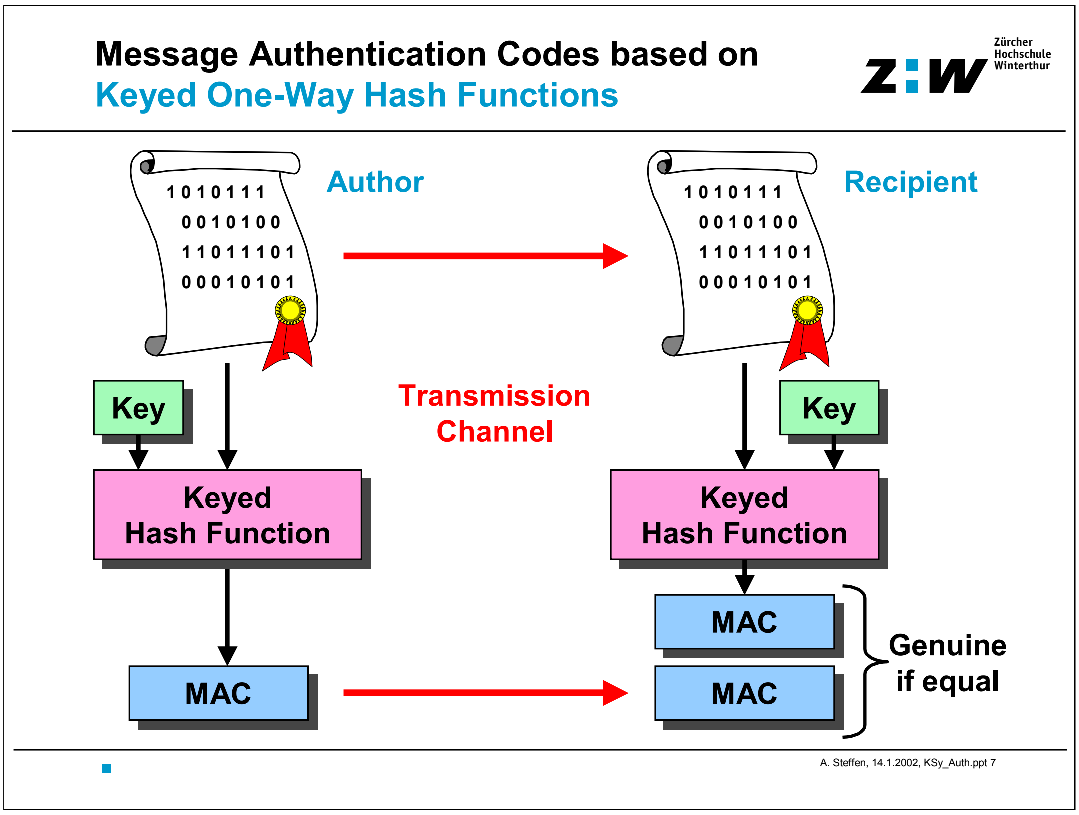
A. Steffen, 14.1.2002, KSy_Auth.ppt 6

Block Algorithms

- Both MD5 and SHA hash functions work on input data blocks of exactly 512 bits. A document to be hashed must first be partitioned into an integer number of data blocks of this size. This is done by first appending a 64 bit document length L to the end of the document and then inserting 0 .. 511 padding bits in front of the document length field in order to fill the last block up to 512 bits.
- This block-by-block processing allows the hashing of arbitrarily large documents in a serial fashion.

Initialization Vector / Hash Value

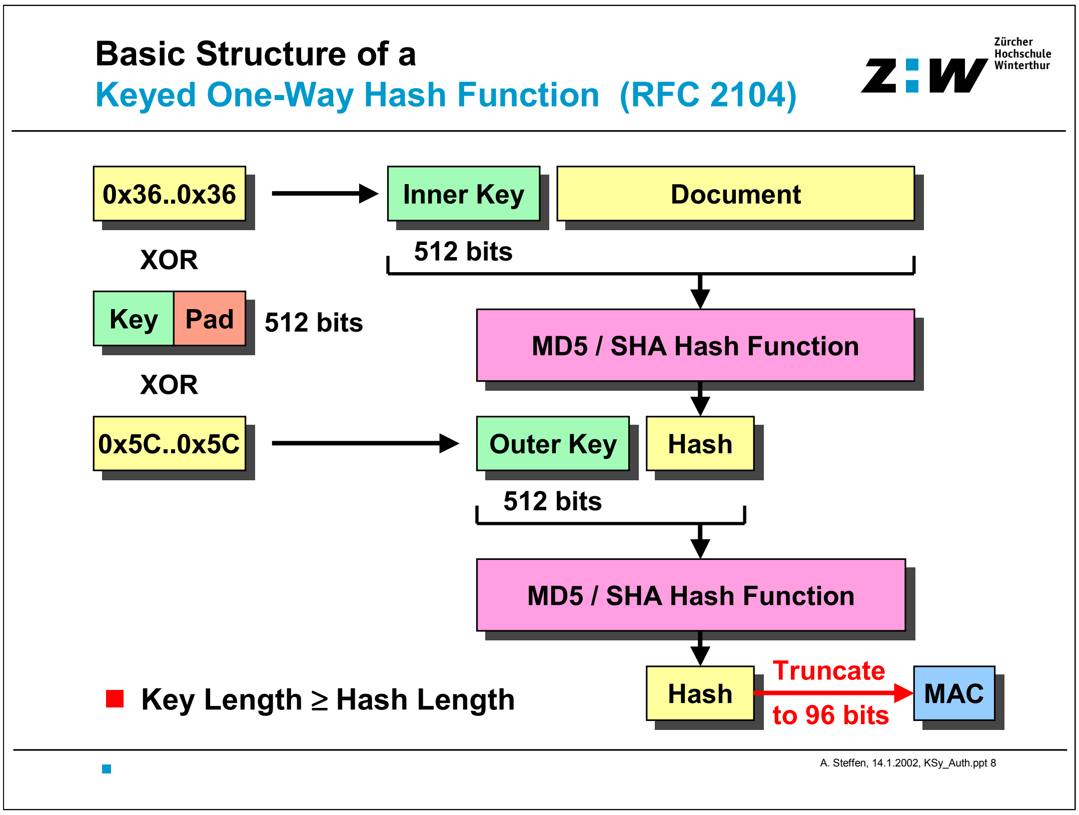
- Besides the 512 bit input data block the hash function is going to process at a time, it requires an initialization vector (IV) of a size that corresponds to the hash value to be computed (128 bits or 160 bits for MD5 or SHA, respectively).
- During the first round the IV takes on a predefined value published in the MD5 and SHA specifications, respectively. Based on the first block of 512 input bits a hash value is computed. If the document consists of a second data block then the hash value of the first round is taken as the IV of the second round. In this chain-like fashion an arbitrary number of N blocks can be hashed, with the hash value of the previous round serving as initialization vector of the next round. After the last block has been processed, the final hash value is returned as a fingerprint representing the whole document.



Message Authentication Codes based on Keyed One-Way Hash Functions

Message Authentication Codes

- A digital message digest in itself does not offer any protection against unauthorized modifications of a message or document. After any change to a document, a new valid MD5 or SHA hash value could be computed on the new content, since the hash algorithms in use have been published and are well documented.
- Only by introducing a secret key into the fingerprint computation a document can be secured against unauthorized modifications. Only the owner(s) of the secret key can produce a valid message digest which is now called a **Message Authentication Code** (**MAC**). Of course the recepient of the secured document must possess the secret key in order to verify the validity of a message by also computing the MAC value and comparing it to the MAC transmitted or stored together with the corresponding document.
- The question now is how to construct efficient **Keyed One-Way Hash Functions** based on the hash algorithms we already know!



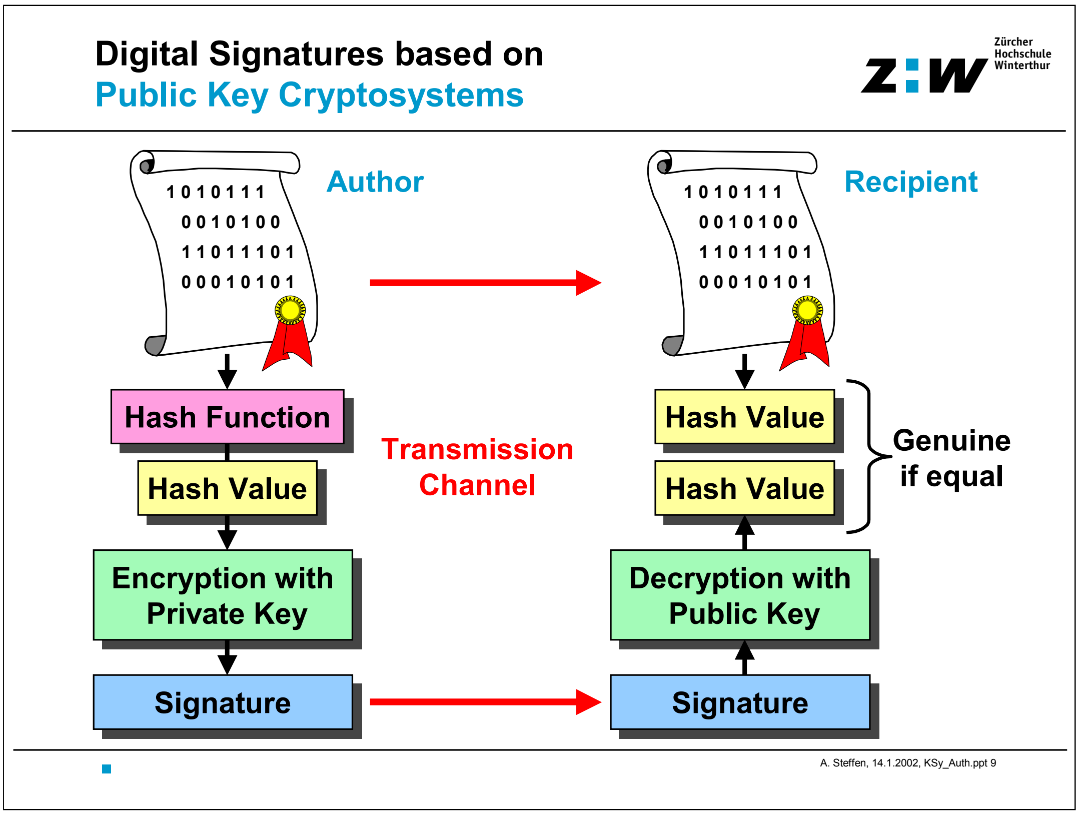
Basic Structure of a Keyed One-Way Hash Function (RFC 2104)

- Key Length ≥ Hash Length

A. Steffen, 14.1.2002, KSy_Auth.ppt 8

Keyed One-Way Hash Functions

- RFC 2104 proposes a method how a keyed one-way hash function can be constructed on the basis of any block-oriented hash function like MD5 or SHA.
- In front of the document to be authenticated, an additional 512 bit inner key block is prepended. This inner key block is formed be padding the secret key up to the full block size of 512 bits and then XOR-ing this first block with a repetition of the value 0x36. In order to achieve maximum security, the length of the secret key should be at least the size of the hash value, i.e. 128 bits for MD5 and 160 bits for SHA.
- This augmented document is now fed into the chosen hash algorithm. Since the hash value of the previous block always serves as an initialization vector for the next block, the hash function operating on the inner key block generates an intialization vector for the hashing of the actual document that depends on the secret key only. As long as the secret key remains the same, all messages can be signed using the same secret initialization vector.
- The same is true for the outer key block, which is formed by XOR-ing the padded key with a different repeated byte value of 0x5C and which is always prepended to the hash value coming out of the first hashing round. The outer key block can be used to compute a second key-dependent initialization vector to hash the hash value coming out of the first round a second time.
- Often the final MAC value is formed by truncating the computed hash value of 128 bits or 160 bits obtained by MD5 or SHA, respectively. A commonly used MAC size is 96 bits. Although discarding part of the hash bits reduces the number of combinations a brute force attack would have to try by a significant factor, it also hides part of the internal state of the hash algorithm, making it more difficult for an attacker to work himself backwards from the output of the second hash round towards the intermediate result of the first hash round.



Digital Signatures based on Public Key Cryptosystems

Author

1010111
0010100
11011101
00010101

Hash Function

Hash Value

Encryption with Private Key

Signature

Transmission Channel

Recipient

1010111
0010100
11011101
00010101

Hash Value

Hash Value

Decryption with Public Key

Signature

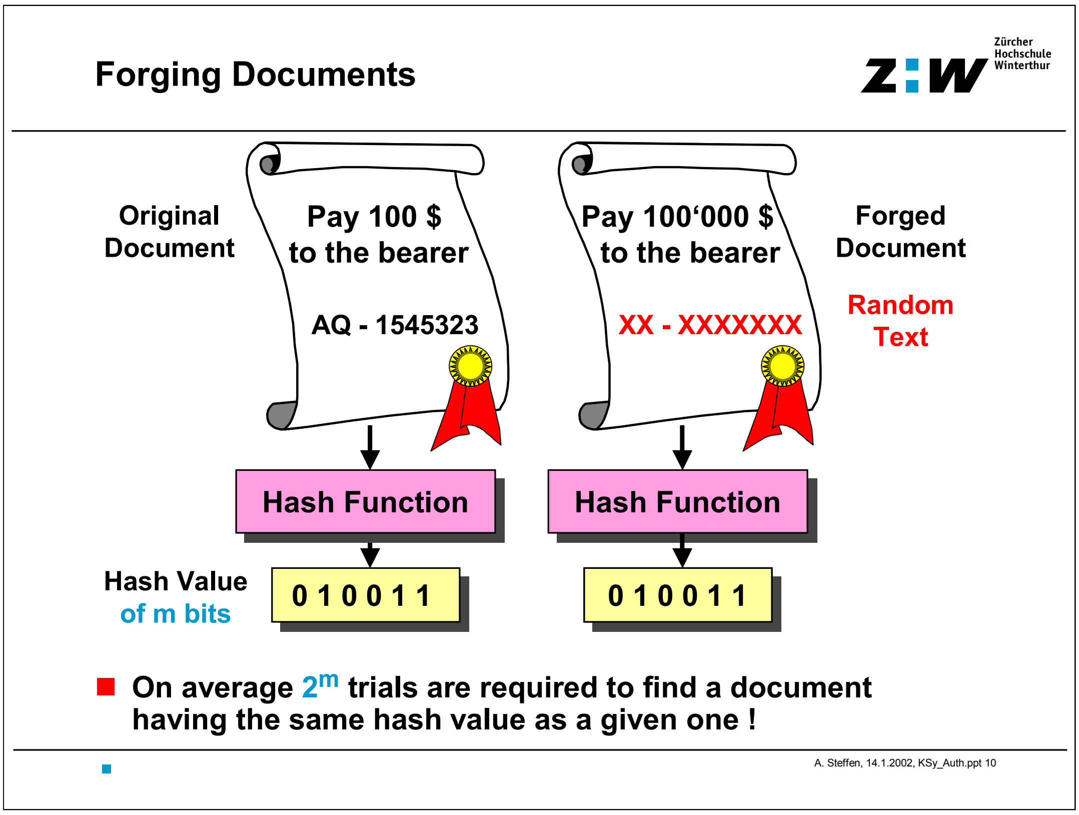
Genuine if equal

Digital Signatures based on Public Key Cryptosystems

- The MD5 or SHA hash value of a message or document is encrypted by the author using his private key, thereby forming a digital signature that is transmitted or stored together with the corresponding message or document.
- The recipient computes the hash over the received document and decrypts the received signature, using the public key of the document‘s author. If the decrypted value equals the computed hash value then the document must be authentic, since only the author possesses the correct private key used to encrypt the signature.

Comparison:

- **Digital Signatures based on Secret Keys**
  - **Pro**: Very fast and efficient, since MAC generation is based on simple hashing functions. Often used for authentication of bulk data at high data rates (e.g. applied to IPsec datagrams).
  - **Contra**: Recipient must know the secret key in order to verify the authenticity of a message. This often leads to a secure key distribution problem.
- **Digital Signatures based on Public/Private Key Pairs**
  - **Pro:** Public keys can be freely and openly distributed, so anyone can check the authenticity of a message.
  - **Contra**: Encryption and decryption operations using private and public keys, respectively, are extremely time-consuming. Used for user or host authentication at the beginning of a remote-login session or for signing low-volume e-mail messages.



Forging Documents

Zürcher Hochschule Winterthur

Original Document

Pay 100 $ to the bearer

AQ - 1545323

Hash Function

Hash Value of m bits

0 1 0 0 1 1

Pay 100'000 $ to the bearer

XX - XXXXXXX

Forged Document

Random Text

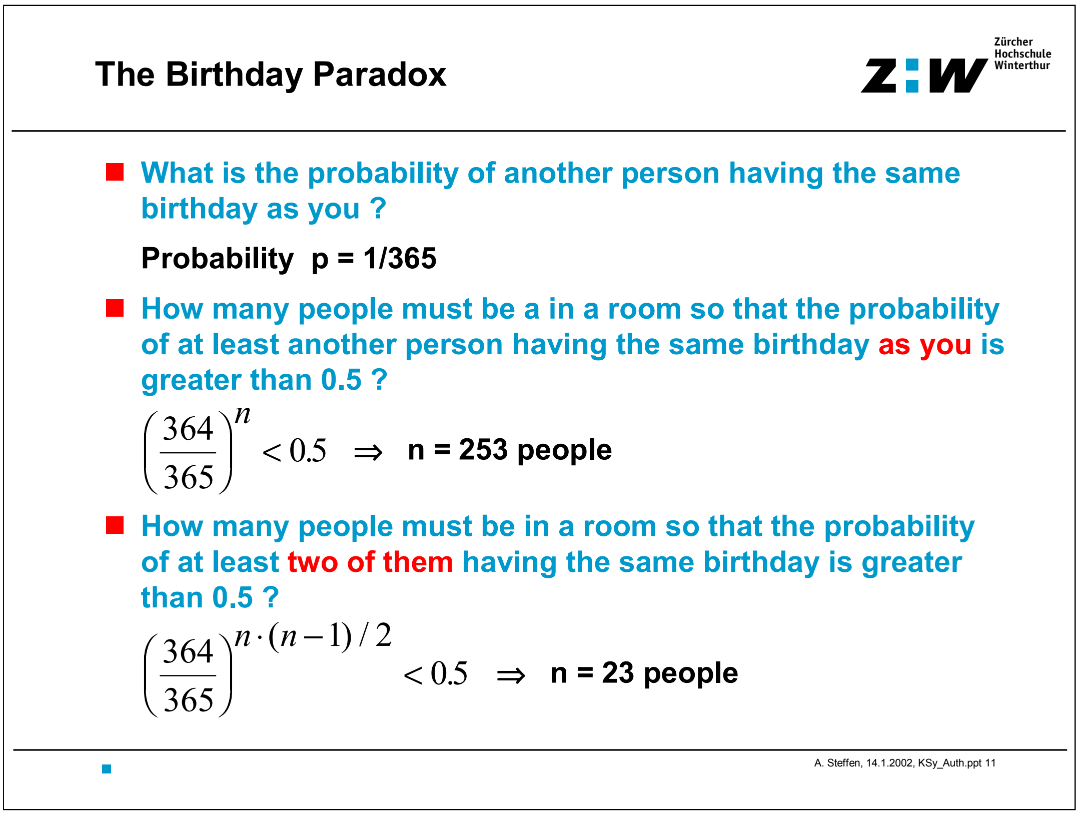
Hash Function

0 1 0 0 1 1

- On average 2^m trials are required to find a document having the same hash value as a given one !

Forging a document with a given hash value

- If a message digest is protected either by using a keyed message authentication code or a digital signature based on a public key cryptosystem, a document can only be successfully forged by creating a second document that has the same hash value as the original document. The forged document can contain a completely different text, but it must offer the possibility to add a certain amount of random text that is either hidden (e.g. by using combinations of <space> and <backspace> characters) or as in our example poses as an arbitrary serial number.
- The random text part of the fake document is now repeatedly changed until the computed hash value matches the fingerprint of the original message. If the hash value has a size of m bits then it can be shown that on the average 2^m trials are required until a document with a matching hash is found. For MD5 this translates into about 2^{128} trials and for SHA even into 2^{160} trials, i.e. hopelessly too many to be able to find a matching document within a reasonable timespan, even when using the most powerful computers or special hardware equipment.



The Birthday Paradox

Zürcher Hochschule Winterthur

- What is the probability of another person having the same birthday as you ?

  Probability p = 1/365

- How many people must be a in a room so that the probability of at least another person having the same birthday as you is greater than 0.5 ?

  $\left(\frac{364}{365}\right)^{n} < 0.5 \quad \Rightarrow$ **n = 253 people**

- How many people must be in a room so that the probability of at least two of them having the same birthday is greater than 0.5 ?

  $\left(\frac{364}{365}\right)^{n \cdot (n-1)/2} < 0.5 \quad \Rightarrow$ **n = 23 people**

- Probability that another person does not have the same birthday as you:

$$p = \frac{364}{365}$$

- Probability that two other persons do not have the same birtday as you:

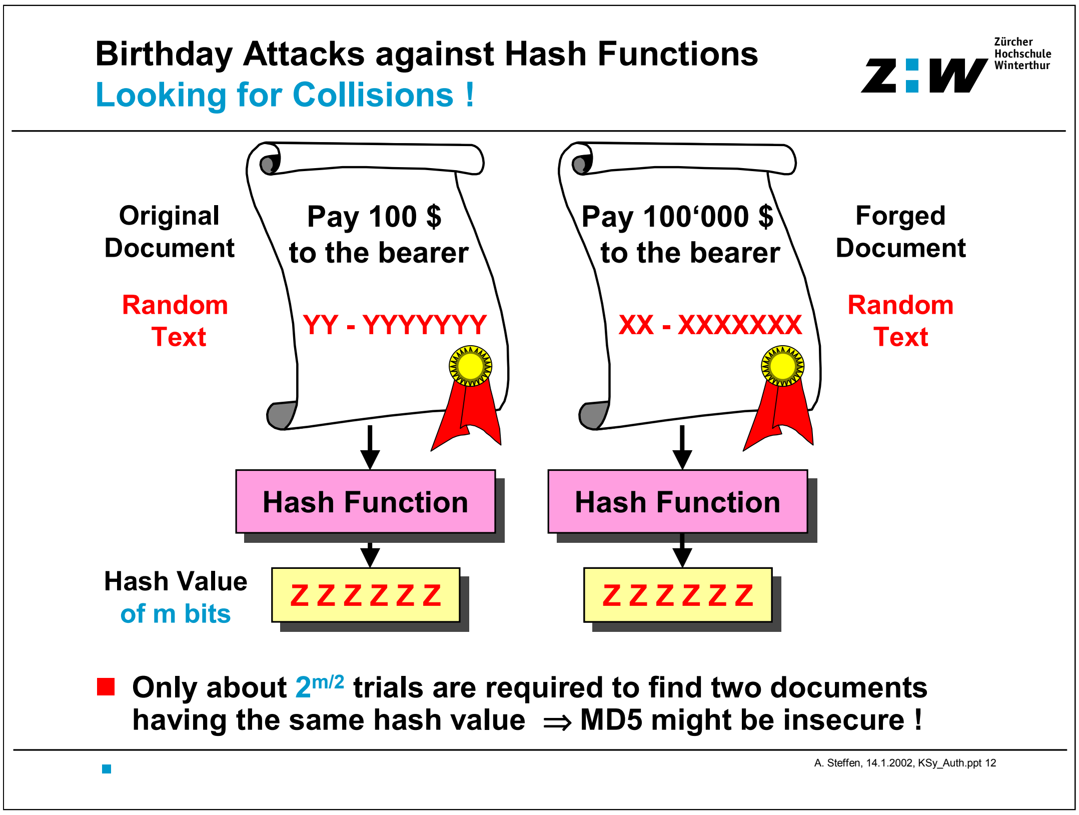
$$p = \frac{364}{365} \cdot \frac{364}{365} = \left(\frac{364}{365}\right)^{2}$$

- Probability that n other persons do not have the same birtday as you:

$$p = \left(\frac{364}{365}\right)^{n}$$

- Probability that among n people no one shares his birthday with a second person:

$$p \approx \left(\frac{364}{365}\right)^{n-1} \cdot \left(\frac{364}{365}\right)^{n-2} \cdot \;\cdots\; \cdot \left(\frac{364}{365}\right)^{2} \cdot \frac{364}{365} \;=\; \left(\frac{364}{365}\right)^{n \cdot (n-1)/2}$$



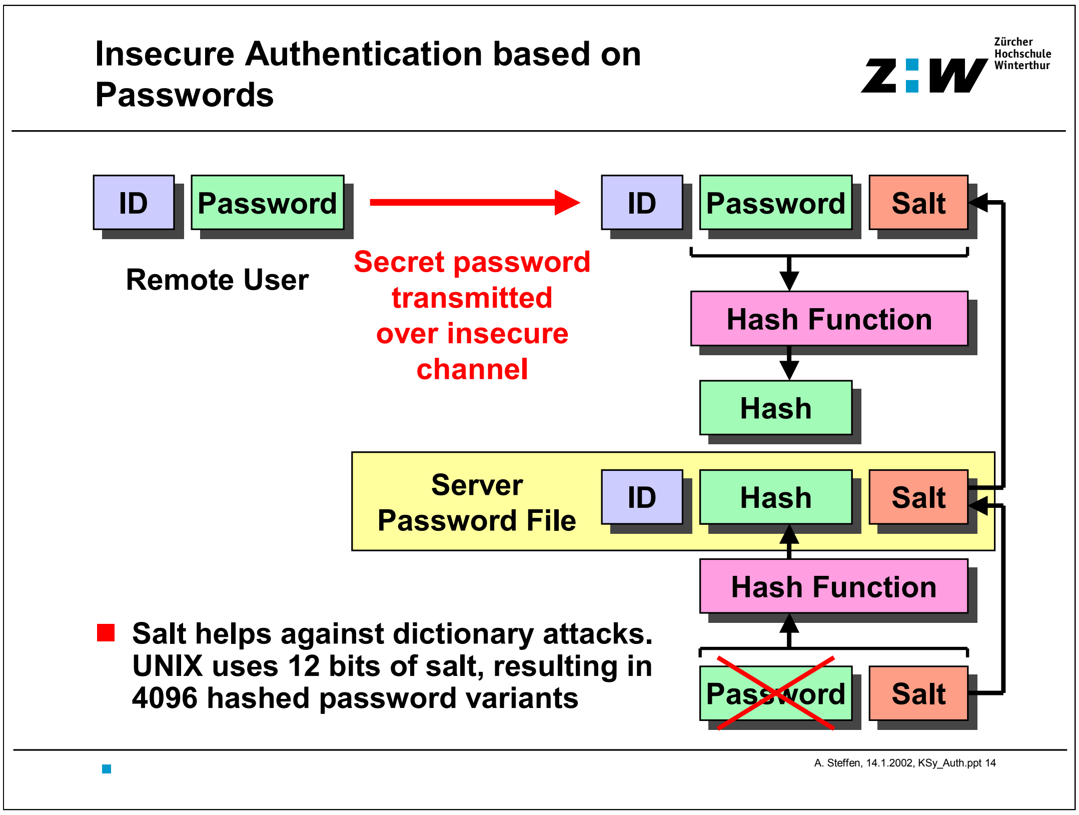
Birthday Attacks against Hash Functions
Looking for Collisions !

- Only about $2^{m/2}$ trials are required to find two documents having the same hash value ⇒ MD5 might be insecure !

A Perfect Crime

- Imagine that you are allowed to create an electronic cheque that someone who owns you money is going to sign digitally. You generate two versions: One cheque over 100 $ and a second one over 100‘000 $. The actual hash values of the two cheques are not important, the only condition that must be fulfilled is that they be identical.
- You now present the first check to your debitor who signs it by encrypting the hash value with his private key. The hash value is now secured and cannot be changed anymore. This does not worry you, since the second cheque has exactly the same fingerprint.
- You go now to the bank and present the forged cheque together with the digital signature of the first cheque. The cashier decrypts the signature using the debitor‘s public key and compares the decrypted value with the hash of the forged cheque. Everything is o.k., you get paid 100‘000 $ and live merrily ever after.
- The sad thing about this story is that it can be done, if the size *m* of the message digest is not large enough. Since you must only find two documents having the same but otherwise arbitrary hash value, the birthday paradox applies. Instead of 2^m trials to find a matching second document, only about $2^{m/2}$ trials are needed if both documents can be freely chosen.
- For the MD5 message digest on the average 2^{64} different documents have to be generated until at least one matching pair of hash values is found. This search requires an enormous amount of computation and storage space but has been shown to be feasible. Therefore MD5 is not regarded as secure enough any more when the authenticity of a document must be guaranteed over a long period of time.
- For highly sensitive applications SHA-1 should be used. A birthday attack would require 2^{80} trials which at least for the next couple of years is still beyond reach of a brute force search attack. But in order to be on the safe side, message digests are going to be extended to 256 bits in the near future. SHA-2 is a likely candidate.

Authentication Schemes



Insecure Authentication based on Passwords

ID Password

Remote User

Secret password transmitted over insecure channel

ID Password Salt

Hash Function

Hash

Server Password File

ID Hash Salt

Hash Function

~~Password~~ Salt

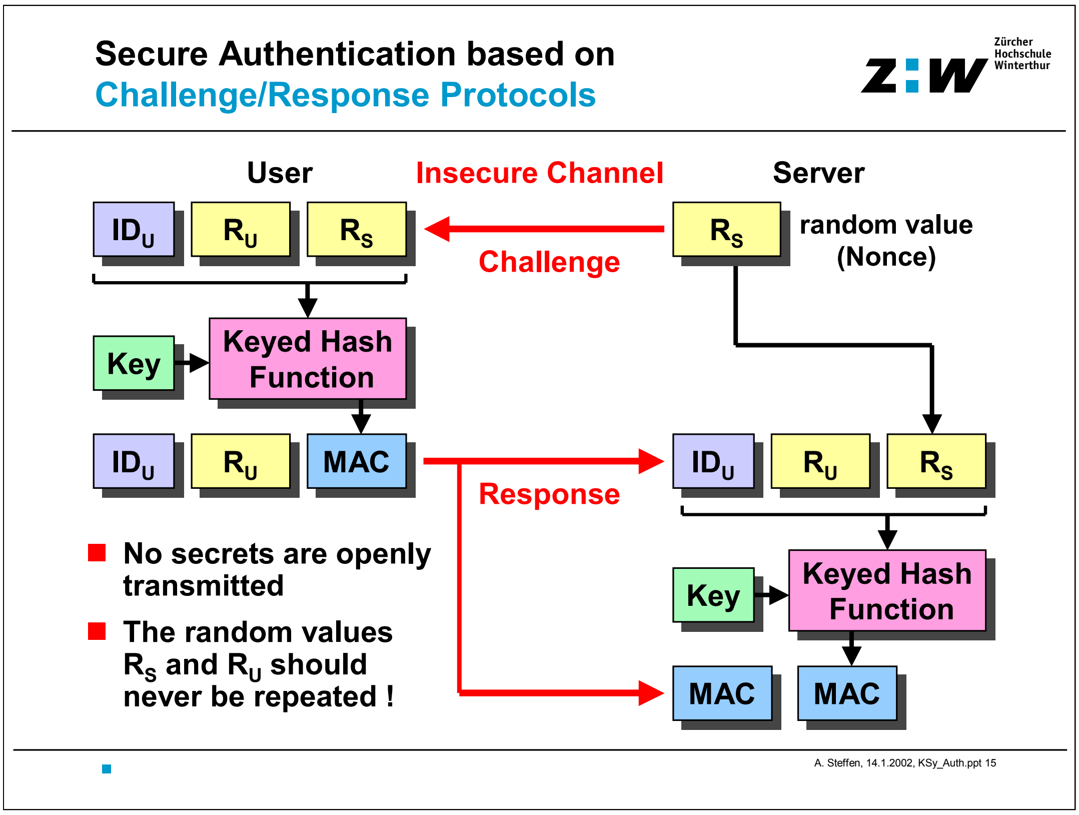
- Salt helps against dictionary attacks. UNIX uses 12 bits of salt, resulting in 4096 hashed password variants

Vulnerability #1: Secret password openly transmitted over insecure channel

- Remote login using the still popular **telnet** protocol requires the transmission in the clear of the remote user‘s identity and password to the host computer. As long as this happens over a direct dial-in telephone line this procedure is relatively secure since it is not so trivial to tap a phone line. But as soon as the login happens from one host computer over the internet to another host, it becomes highly dangerous to send a secret password over such an insecure channel. Cryptographically safe mechanisms like e.g. the **secure shell** (**ssh**) should be used.

Vulnerability #2: Secret password residing on a server storage medium

- Another problem is the secure storage of user passwords on login servers. Since servers usually operate without human intervention, secrets residing on a server cannot be protected by encrypting them with a passphrase. Making the secrets root-readable only offers a certain protection, but as soon as an intruder gains root status all secrecy is lost.
- UNIX systems do not store the user passwords themselves but a DES hash of them. Since UNIX passwords are restricted to 8 ASCII characters, a dictionary attack could be used by precomputing the hash values of several millions of the most popular passwords. By adding a 12 bit random salt value to the password prior to the hashing process, 4096 different hash variants of the same password are created, multiplying the storage requirements for a precomputed dictionary attack by this salt factor. With today‘s hard disk capacities of several gigabytes, salting does not pose a serious obstacle for a cracker anymore.



Secure Authentication based on Challenge/Response Protocols

Zürcher Hochschule Winterthur

User

Insecure Channel

Server

ID_U R_U R_S

Challenge

R_S random value (Nonce)

Key

Keyed Hash Function

ID_U R_U MAC

Response

ID_U R_U R_S

Key

Keyed Hash Function

MAC MAC

- No secrets are openly transmitted
- The random values R_S and R_U should never be repeated !

Why Challenge/Response Protocols?

- The proper way of doing an authentication over an insecure channel is to use a challenge / response protocol which checks if a user requesting access to a server is in possession of a personal secret authenticating her, without actually exchanging any secret information over the open channel.

Challenge/Response Protocols using Message Authentication Codes (MACs)

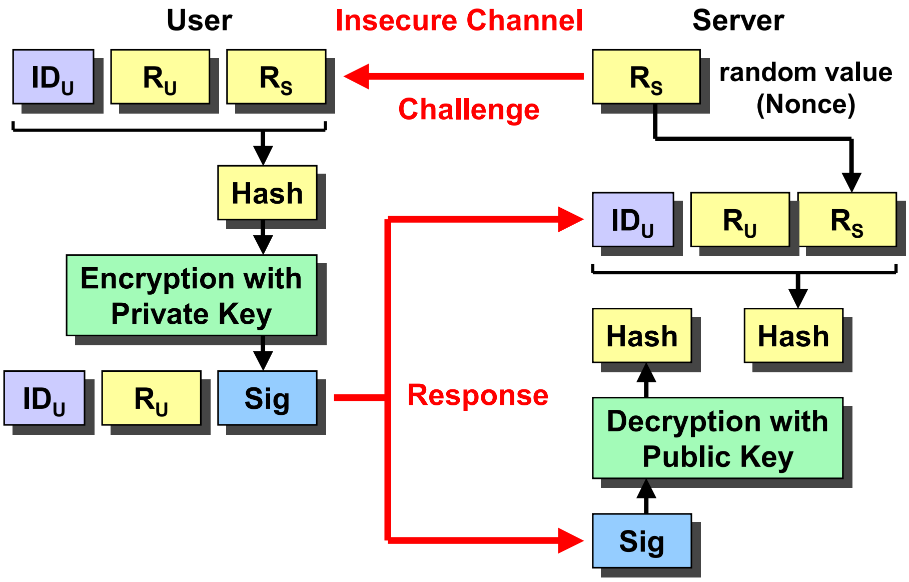
- This method uses a common secret key shared between the user and the server. Every time a user wants to log into a server, the server sends a **challenge** to the user in the form of a **random value** or „nonce“.

  In order to prevent replay attacks a challenge string should never be used twice !

- The user creates a response by concatenating the random value R_S received from the server with her user ID and then feeding this string into a keyed hash function initialized with the secret key. In order to thwart any attempts of an attacker to gain only the slightest information about the secret key by sending controlled sequences of challenge strings, the user adds a random number R_U of his own into the hashing process.
- The user sends her ID string together with the random number she chose and the generated MAC to the server. Under the assumption that the MAC algorithm cannot be inverted, the transmitted data does not disclose any information about the secret key to potential attackers listening in to the channel.
- Since the server has now exactly the same information as the user, i.e. user ID, server nonce, user nonce and secret key, it computes the MAC value as well and compares the result to the MAC transmitted by the user. The user is successfully authenticated if the two values match.

Challenge/Response Protocol based on Digital Signatures

Zürcher Hochschule Winterthur



User

Insecure Channel

Server

ID_U R_U R_S

Challenge

R_S random value (Nonce)

Hash

Encryption with Private Key

ID_U R_U Sig

Response

ID_U R_U R_S

Hash

Hash

Decryption with Public Key

Sig

Challenge/Response Protocols using Public Key Digital Signatures

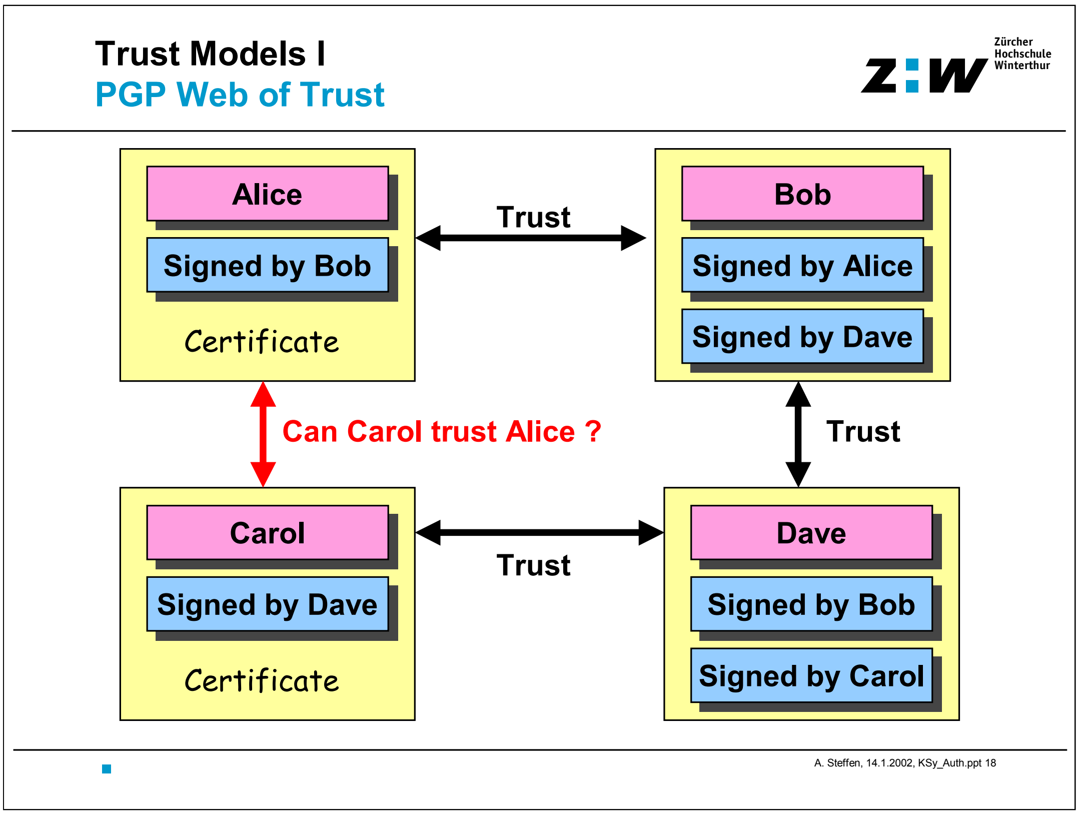
- Challenge/response protocols can also be based on digital signatures using public key algorithms. The user is the sole possessor of the private key, whereas any potential server has a copy of the corresponding public key.
- The user forms a hash value out of her user ID, the random value R_S received from the server as a challenge and a random value R_U of her own choice. By encrypting the hash value with her private key the user forms a digital signature that is sent back as a response to the server.
- It is of utmost importance that the user contributes some random value of her own to the hashing process. Otherwise the challenge string could be abused to trick the user e.g. into blindly signing a document or even to make her decrypt a secret message that has been encrypted using the user‘s public key. If RSA is used for the signatures, certain attacks become possible that can even reveal the private key.

Can public keys be trusted?

- Public key cryptosystems have become popular because the public key does not have to be kept secret and can therefore be published and distributed freely. So when a server authenticates a user by verifying the user signature on the basis of the user‘s public key, the question remains whether the public/private key pair used in the authentication process really belongs to this user.
- If a public key used in an authentication process is fetched from a public directory using an open communications channel, a „man-in-the-middle“ attack can easily replace the original key with a public key generated by the attacker.
- This is where certificates come into play. Certificates establish a reliable and trusted link between a user identity and a public key. The trust mechanisms involved will be presented on the following slides.

Secure Network Communication – Part III

Certificates



Trust Models I

PGP Web of Trust

Zürcher Hochschule Winterthur

Alice

Signed by Bob

Certificate

Trust

Bob

Signed by Alice

Signed by Dave

Can Carol trust Alice ?

Trust

Carol

Signed by Dave

Certificate

Trust

Dave

Signed by Bob

Signed by Carol

Web of Trust

- One possible method of establishing trust in a user‘s public key is the **web of trust** approach employed by the popular **Pretty Good Privacy** (PGP) mail encryption and authentication package. It is typical for closely-knit communities like e.g. people working on a large software project or special interest groups that a personal link can be established between any two people in the community using a relatively small number of „hops“ on the basis of common friends.

Can Carol trust Alice ?

- Can Carol trust Alice if she has never met her before? Probably yes, because Alice has a good friend called Bob, who works together with Dave, who in turn is a friend of Carol‘s. So using three „hops“ a link between Carol and Alice can be established.

How is the „Web of Trust“ principle applied to public key certificates?

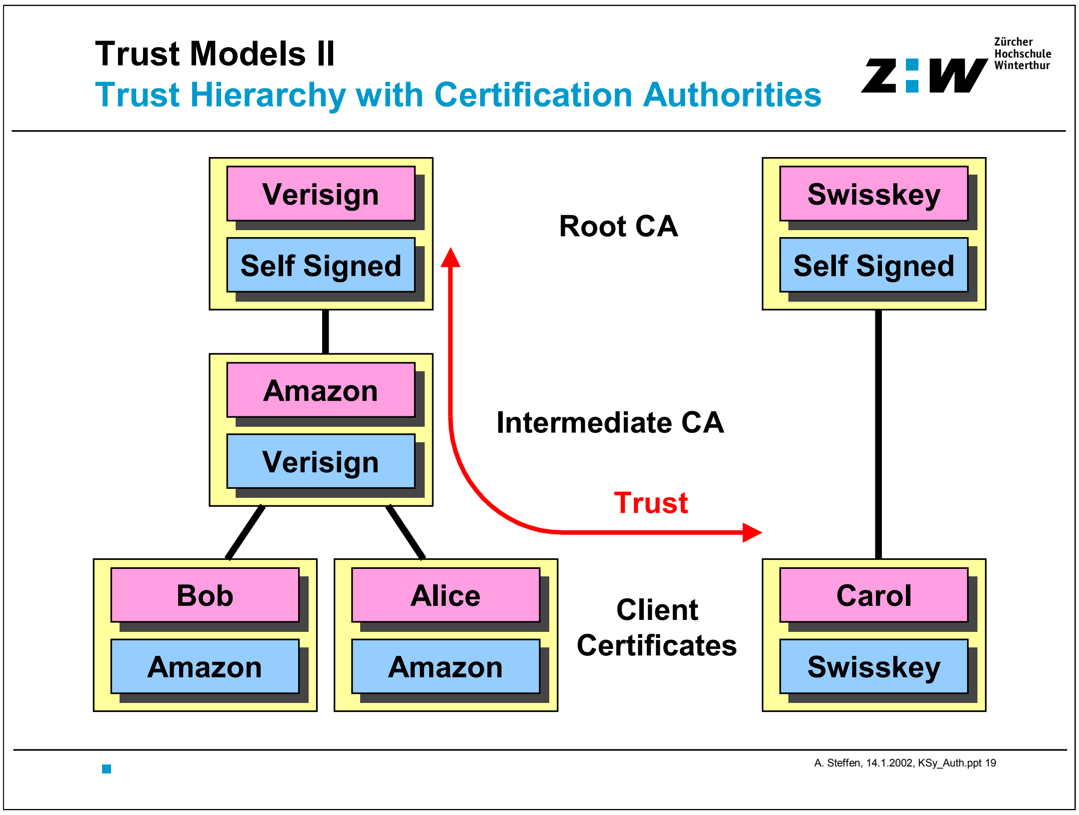
- In a web of trust every participant asks her friends to sign the hash of her user certificate containing the user ID (e.g. an e-mail address) and the public key.
- So when Carol receives an e-mail signed by Alice, she gets Alice‘s certificate from a public directory and sees that it has been signed by Bob. Next she fetches Bob‘s certificate and sees that it has been signed by Dave. Again she asks for Dave‘s certificate and sees that she herself has signed Dave‘s certificate. The chain has now been completed and trust has been established.

Scalability of the Web of Trust

- As mentioned above, a web of trust approach is appropriate for relatively compact user communities. The method does not scale very well when millions of people must be authenticated. The average number of hops rises and with them the number of certificate look-up. Also the longer a trust chain gets, the less trustworthy it becomes.

Don‘t trust authorities!

- The big advantage of a web of trust among peers is that no central authority is required that could become corrupted,.



Trust Models II

Trust Hierarchy with Certification Authorities

Verisign
Self Signed

Root CA

Swisskey
Self Signed

Amazon
Verisign

Intermediate CA

Trust

Bob
Amazon

Alice
Amazon

Client
Certificates

Carol
Swisskey

Hierarchical Trust Chains

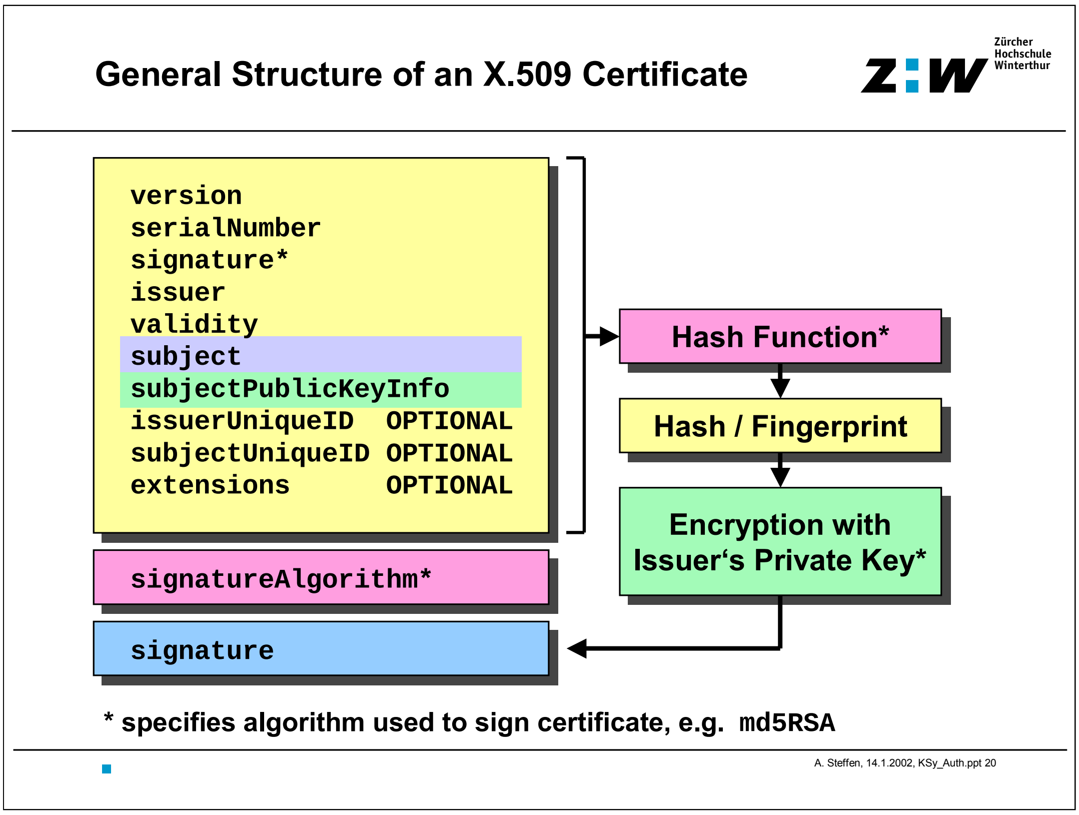
- At the moment it looks as if a second trust model based on certification authorities and hierarchical trust chains is going to establish itself for large scale certificate usage and deployment.

Root Certification Authorities - Root CA

- At the top of the hierachical trust chain are a few Root Certification Authorities which are well known and which intrinsically must be trusted. Each Root CA publishes a **Certificate Practice Statement** (CPS) defining on what policies user or server certificates are issued, how they are managed and how they can be revoked.
- Examples of well-known Root CAs are Verisign, RSA, Baltimore, Entrust, Thawte, Deutsche Telekom and Swisskey.

Intermediate Certification Authorities - Intermediate CA

- Root CAs can directly issue user certificates. This is usually done in the case of private individuals who apply directly for a certificate.
- For large or medium organisations it is much more flexible to set up a certification authority of their own, so that they can issue and revoke certificates for their staff themselves. The certificate of this so called **Intermediate Certification Authority** which is used to sign staff certificates, is in turn issued and signed by a generally trusted Root CA.
- In principle an arbitrary number of hierarchy levels could be implemented, but usually there are not more than two or three hops from a user certificate to root CA certificate at the top of the trust chain.



General Structure of an X.509 Certificate

```
version
serialNumber
signature*
issuer
validity
subject
subjectPublicKeyInfo
issuerUniqueID  OPTIONAL
subjectUniqueID OPTIONAL
extensions      OPTIONAL
```

```
signatureAlgorithm*
```

```
signature
```

Hash Function*

Hash / Fingerprint

Encryption with Issuer‘s Private Key*

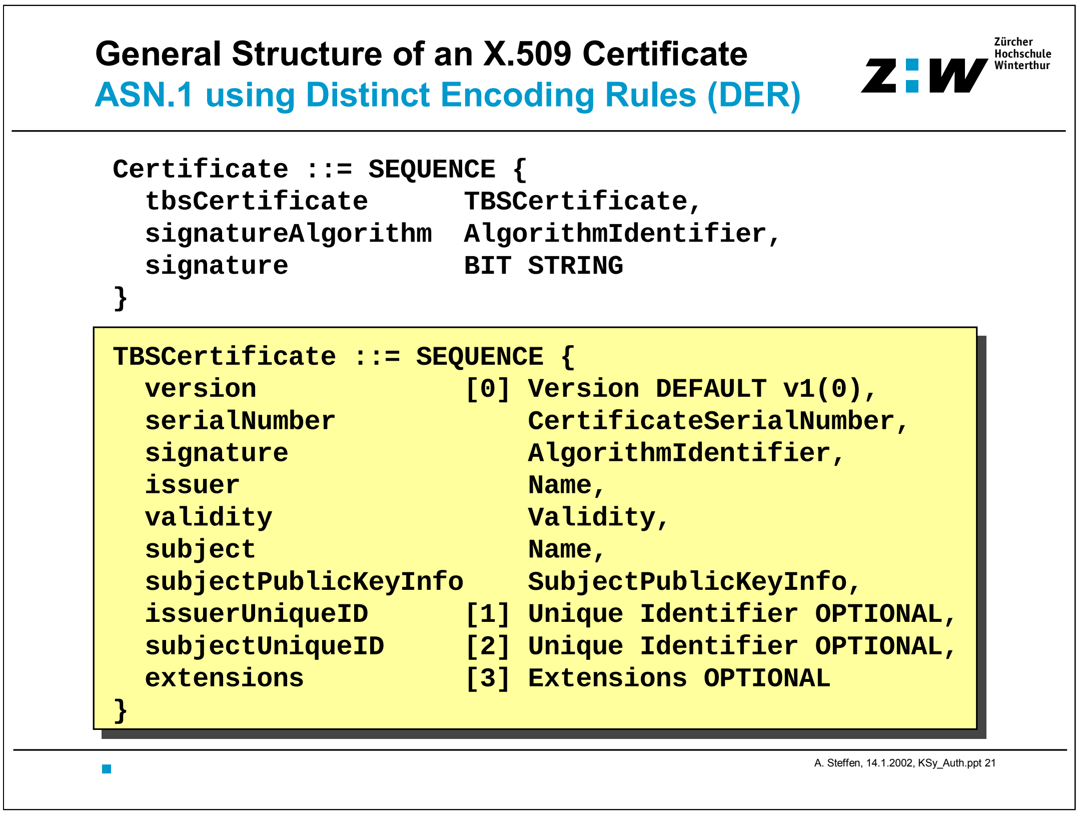
* specifies algorithm used to sign certificate, e.g. `md5RSA`

Certificate Types

- Webs of trust usually use OpenPGP Certificates (RFC 2440)
- Hierarchical trust chains usually use ITU-T X.509 certificates (RFC 2459)

Structure of an X.509 Certificate

- An X.509v3 certificate consists of three parts:
- A certificate body containing
  - a version number (currently v3, v2 and v1 are also possible)
  - a unique serial number assigned by the responsible CA
  - a declaration of the signature algorithm to be used to sign the certificate
  - the ID of the CA that issued and signed the certificate
  - the validity period (not valid before / not valid after)
  - the subject (user) ID
  - the public key of the subject (user)
  - any number of optional v2 or v3 extensions, some of them being very important
- A definition of the signature algorithm used by the CA to sign the certificate
- The signature guaranteeing the authenticity of the certificate, consisting of the hashed certificate body encrypted by the CA‘s private key.



General Structure of an X.509 Certificate
ASN.1 using Distinct Encoding Rules (DER)

```
Certificate ::= SEQUENCE {
  tbsCertificate      TBSCertificate,
  signatureAlgorithm  AlgorithmIdentifier,
  signature           BIT STRING
}
```

```
TBSCertificate ::= SEQUENCE {
  version             [0] Version DEFAULT v1(0),
  serialNumber            CertificateSerialNumber,
  signature               AlgorithmIdentifier,
  issuer                  Name,
  validity                Validity,
  subject                 Name,
  subjectPublicKeyInfo    SubjectPublicKeyInfo,
  issuerUniqueID      [1] Unique Identifier OPTIONAL,
  subjectUniqueID     [2] Unique Identifier OPTIONAL,
  extensions          [3] Extensions OPTIONAL
}
```

Data Structure Definitions

- The data structures used to define the fields of an X.509 certificate are specified in the ITU-T X.509v3 recommendation. The syntax and the data type primitives are based on the generic ITU-T X.680 Abstract Syntax Notation No. 1.

Encoding of X.509 Certificates

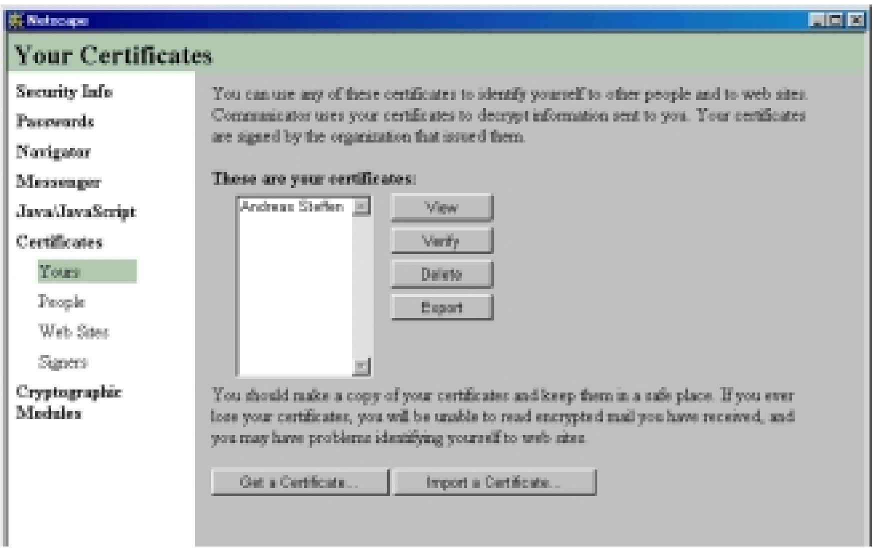
- X.509 certificates are binary encoded using the Distinct Encoding Rules (DER). For details of the encoding, refer to the ITU-T X.690 recommendation.
- The size of a DER encoded X.509v3 certificate containing a 1024 bit RSA public key is usually between 900...1500 bytes, depending on the length of the subject and issuer IDs and the number of included extensions.

Interpreting X.509 Certificates

- The use of the ASN.1 notation and the binary DER encoding make X.509 certificates very difficult to read. Special ASN.1 parsers must be used to convert them into human readable form. The OpenSSL tool kit can be used for this purpose. X.509 certificates can also be analyzed in detail by importing them into Microsoft‘s Internet Explorer.

X.509 Certificate Handling

Netscape 4.7 Browser



Your Certificates

You can use any of these certificates to identify yourself to other people and to web sites. Communicator uses your certificates to decrypt information sent to you. Your certificates are signed by the organization that issued them.

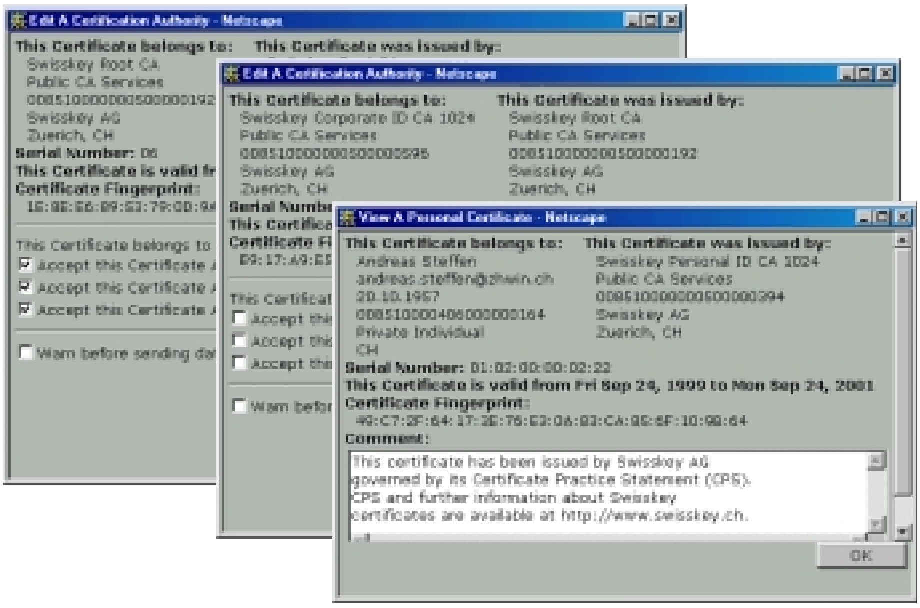
These are your certificates:

You should make a copy of your certificates and keep them in a safe place. If you ever lose your certificates, you will be unable to read encrypted mail you have received, and you may have problems identifying yourself to web sites.

Netscape Menu: `Communicator / Tools / Security Info`

X.509 Certificate Handling - Netscape

Certification Path



Edit A Certification Authority - Netscape

This Certificate belongs to:
Swisskey Root CA
Public CA Services
00851000000005000000192
Swisskey AG
Zuerich, CH

This Certificate was issued by:

Serial Number: 06
This Certificate is valid fr
Certificate Fingerprint:
16:9E:E6:89:53:79:0D:94

This Certificate belongs to
☑ Accept this Certificate A
☑ Accept this Certificate A
☑ Accept this Certificate A

☐ Warn before sending dat

Edit A Certification Authority - Netscape

This Certificate belongs to:
Swisskey Corporate ID CA 1024
Public CA Services
00851000000005000000596
Swisskey AG
Zuerich, CH

This Certificate was issued by:
Swisskey Root CA
Public CA Services
00851000000005000000192
Swisskey AG
Zuerich, CH

Serial Numbe
This Certifica
Certificate Fi
E9:17:A9:E5

This Certificat
☐ Accept thi
☐ Accept thi
☐ Accept thi

☐ Warn befor

View A Personal Certificate - Netscape

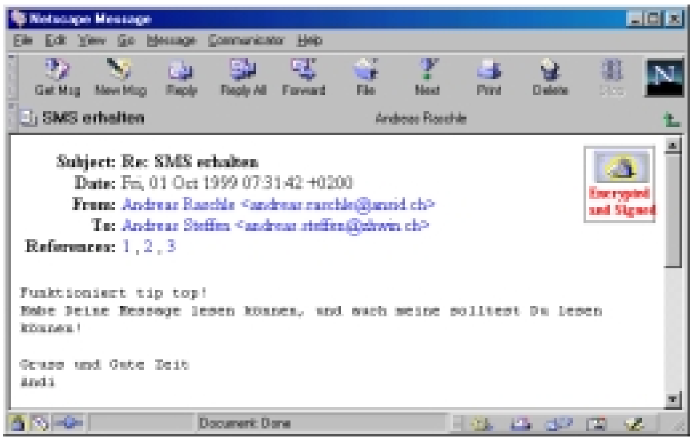
This Certificate belongs to:
Andreas Steffen
andreas.steffen@zhwin.ch
20.10.1957
00851000040600000164
Private Individual
CH

This Certificate was issued by:
Swisskey Personal ID CA 1024
Public CA Services
00851000000005000000294
Swisskey AG
Zuerich, CH

Serial Number: 01:02:00:00:02:22
This Certificate is valid from Fri Sep 24, 1999 to Mon Sep 24, 2001
Certificate Fingerprint:
49:C7:2F:64:17:26:76:E3:0A:83:CA:95:6F:10:98:64
Comment:
This certificate has been issued by Swisskey AG governed by its Certificate Practice Statement (CPS). CPS and further information about Swisskey certificates are available at http://www.swisskey.ch.

X.509 Certificate Handling - Netscape

Encrypted and Signed E-Mail (S/MIME)



SMS erhalten   Andreas Raschle

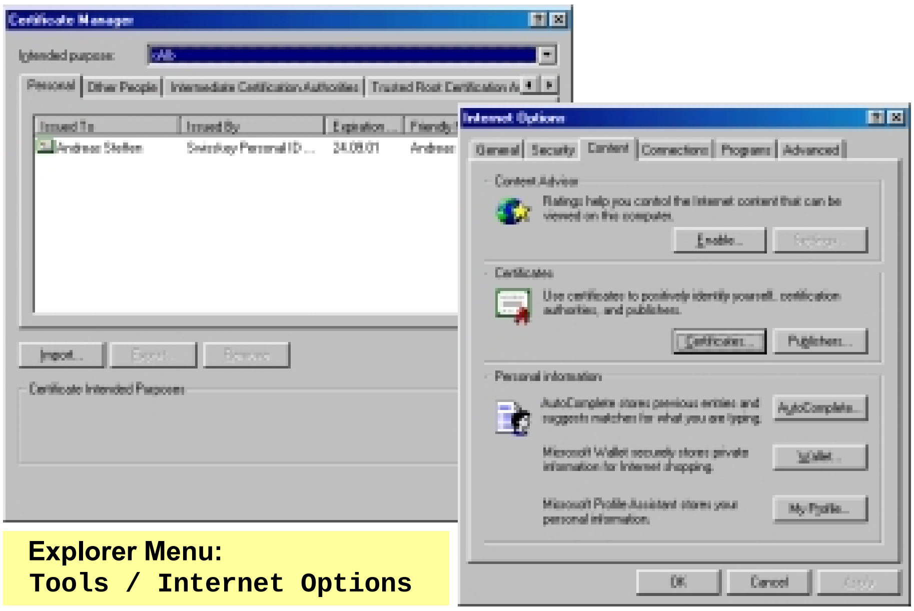
Subject: Re: SMS erhalten
Date: Fri, 01 Oct 1999 07:31:42 +0200
From: Andreas Raschle <andreas.raschle@ascid.ch>
To: Andreas Steffen <andreas.steffen@zhwin.ch>
References: 1 , 2 , 3

```
Funktioniert tip top!
Habe Deine Message lesen können, und auch meine solltest Du lesen
können!

Gruss und Gute Zeit
Andi
```

X.509 Certificate Handling

Microsoft Internet Explorer 5.0



Intended purpose: <All>

Issued To	Issued By	Expiration...	Friendly
Andreas Steffen	Swisskey Personal ID ...	24.08.01	Andreas

Ratings help you control the Internet content that can be viewed on this computer.

Use certificates to positively identify yourself, certification authorities, and publishers.

AutoComplete stores previous entries and suggests matches for what you are typing.

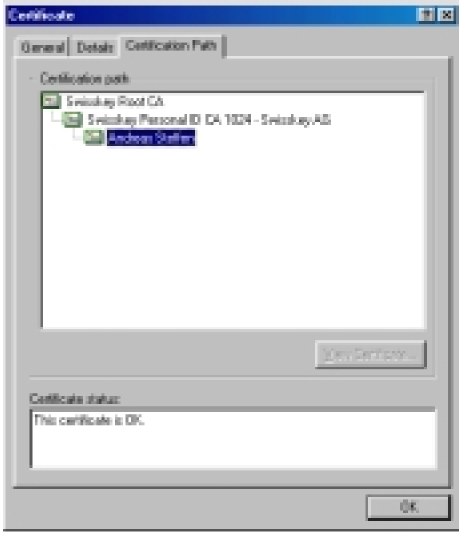
Microsoft Wallet securely stores private information for Internet shopping.

Microsoft Profile Assistant stores your personal information.

Explorer Menu:
`Tools / Internet Options`

X.509 Certificate Handling – Internet Explorer

Certification Path

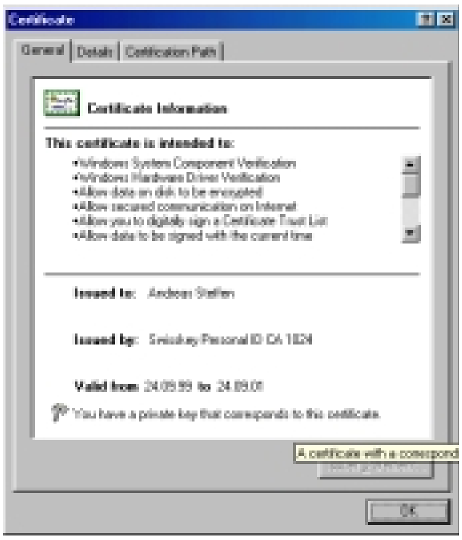


Certification path

- Swisskey Root CA
  - Swisskey Personal ID CA 1024 - Swisskey AG
    - Andreas Steffen

Certificate status:

This certificate is OK.



Certificate Information

This certificate is intended to:

- Windows System Component Verification
- Windows Hardware Driver Verification
- Allow data on disk to be encrypted
- Allow secured communication on Internet
- Allow you to digitally sign a Certificate Trust List
- Allow data to be signed with the current time

Issued to: Andreas Steffen

Issued by: Swisskey Personal ID CA 1024

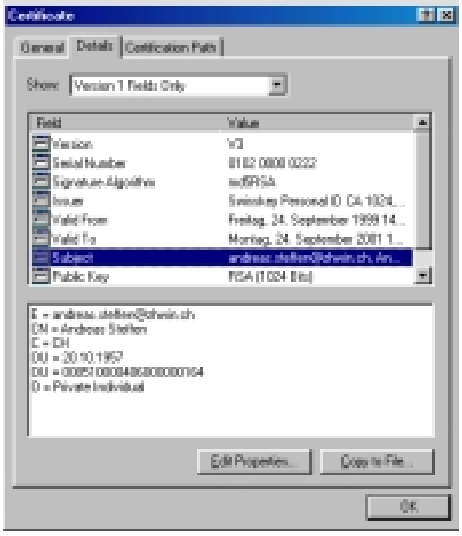
Valid from 24.09.99 **to** 24.09.01

You have a private key that corresponds to this certificate.

A certificate with a correspond

X.509 Certificate Structure
V1 Fields and V3 Extensions

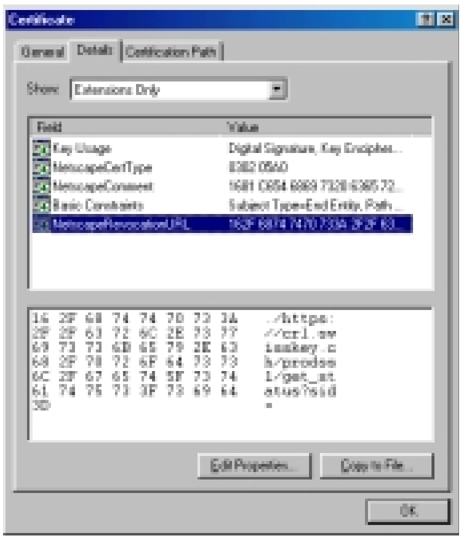
Zürcher Hochschule Winterthur



Show: Version 1 Fields Only

Field	Value
Version	V3
Serial Number	0102 0808 0222
Signature Algorithm	md5RSA
Issuer	Swisskey Personal ID CA 1024...
Valid From	Freitag, 24. September 1999 14...
Valid To	Montag, 24. September 2001 1...
Subject	andreas.steffen@zhwin.ch, An...
Public Key	RSA (1024 Bits)

```
E = andreas.steffen@zhwin.ch
CN = Andreas Steffen
C = CH
OU = 20.10.1957
OU = 00051 0808480808080164
O = Private Individual
```



Show: Extensions Only

Field	Value
Key Usage	Digital Signature, Key Encipher...
NetscapeCertType	0302 05A0
NetscapeComment	1681 C654 6869 7320 6365 72...
Basic Constraints	Subject Type=End Entity, Path...
NetscapeRevocationURL	162F 6874 7470 733A 2F2F 63...

```
16 2F 68 74 74 70 73 3A   ./https:
2F 2F 63 72 6C 2E 73 77   //crl.sw
69 73 73 6B 65 79 2E 63   isskey.c
68 2F 70 72 6F 64 73 73   h/prodss
6C 2F 67 65 74 5F 73 74   l/get_st
61 74 75 73 3F 73 69 64   atus?sid
3D                        =
```

Public Key Infrastructure (PKI)

- **Certification Authority**
  - Governed by a Certificate Practice Statement (CPS)
  - Issues and signs Client and Server Certificates
  - Maintains a Certificate Revocation List (CRL)
  - Offers LDAP / WWW based Directory Services
- **Private Key Management**
  - Secure Generation and/or Distribution of Private Keys
    - Browser or Java Applet generated Keys
    - Hardware generated Keys (Intel 810/820 Chipset, Smart Cards)
  - Secure Storage of Private Keys
    - Smart Cards, USB Modules, SIM Cards (Sonera)
  - Key Recovery of lost private keys