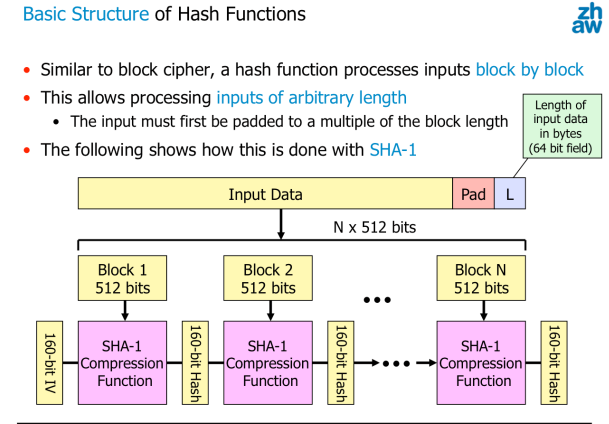
**IS ZA Data**

Important properties:

* Hash can be efficiently comuted, even if the message has a length of several gigabytes
* Mapping should be pseudo-random:
  + If a single bit in the message is changed, about 50 % of all bits in the hash should change
  + Given a ahsh, it is practically impossible to find a message that produces the hash (preimage resistance)
  + Collision resistance



Gebrauch Hash Funktionen:

* Accidental errors during data transmission
* Malicious modifications from an attacker?

A raw hash cannot be used to protect the integrity of a message when assuming an active attacker

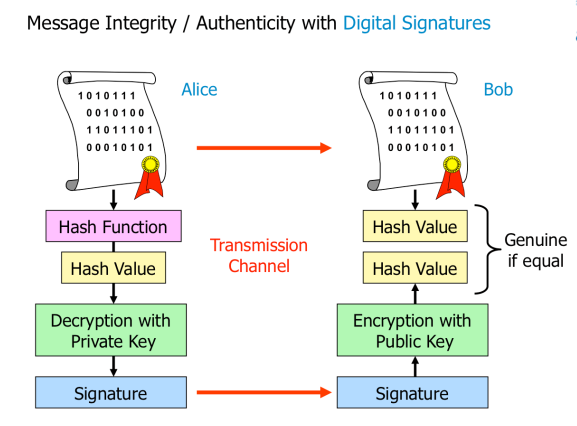
Solution 1: hash function with a secrety key => Message authentication code (MAC)

Solution 2: hash function with digital signatures

Digital signatures :

* Sign a document or a message => private key used
* Verify a signature => public key used

Only a hash of the message and not the message itself is signed



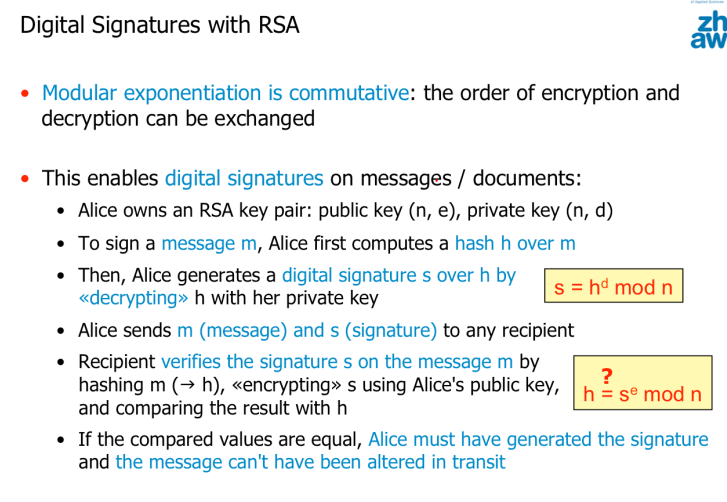
MACs vs digital signatures:

Digital signatures based on secret keys:

* Pro:
  + Fast and efficient
    - Based on simple hashing functions
* Contra:
  + Recipient must know the secret key in order to verify the integrity and authenticity

Digital Signatures based on Public/Private Key Pairs:

* Pro:
  + Public key can be freely distributed
    - Anyone can check the authenticity and integrity
* Contra:
  + Encryption and decryption operations => time-consuming



On average, 2^m-1 trials are required to find a document having the same hash value as a given one => not feasible if m is large => birthday attack

* Wer hat am gleichen Tag Geburtstag wie ich?
* Hash function must be preimage resistant!!

Only 2^m/2 trials are needed to find two documents with the same hash

* Collision resistant
* Welche zwei Leute haben am gleichen Tag Geburstag?

MD5 => collisions detected in one hour, shouldn’t be used anymore

You should combine encryption and integrity protection

ENCRYPT then MAC! => more secure

Otherwise:

* Attacker could modify the ciphertext to exploit possible implementation weaknesses in the decryption mechanism
  + Padding oracle attack
* Curently affect block ciphers used in CBC mode, stream ciphers are not affected

Authenticated Encryption

* GCM (Galois/Counter Mode)
  + Performance:
    - Fast
  + Official NIST standard
  + Integrated solution for combining encryption and a MAC
  + Encrypt then MAC

Authentication with Passwords

Sniffing

* Passwords are transmitted in plaintext over a network => sniff password
* Send them only over protected links (e.g TLS) or using challenge-response protocols

Phishing:

* Fake login => capture password

Online Attacks:

* Einfach ausprobieren beim Login direkt
* Blocking or slowing down the attacker

Offline Attacks:

* Angreifer klaut das Password von der Datenbank
* Passwörter nicht in Plaintext speichern

Password Re-use

* Passwords are re-used for different services => breaking one breaks them all
* Make sure to use dedicated passwords for critical services

Cracking Hashed passwords

* Dictionary attack
* Minor variations
* More random passwords

Adding salt to the password before hashing it

Salt will be in the password file => not possible to brute force without password file

Increasing the computing effort for the attacker is possible by increasing the complexity to compute the password hash

Key stretching => make a weak key (=> passwords) more secure against a brute force attack by increasing the time it takes to test each possible key.

Challenge-response protocols

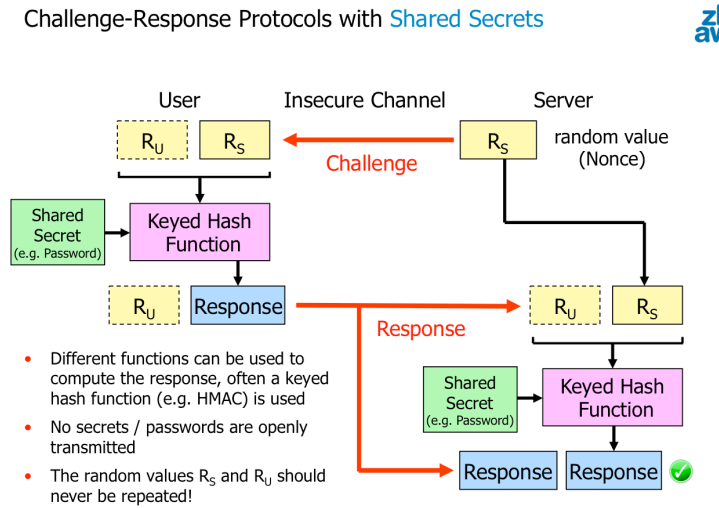
* No passwords are directly sent across the network, which allows using them over insecure channels
* Can also be used with digital signatures

Prinzip:

* Server sends the client a challenge
* Client transforms the challenge in a way that is only possible if the password is known => response
* The response is sent to the server, which checks if the response is correct

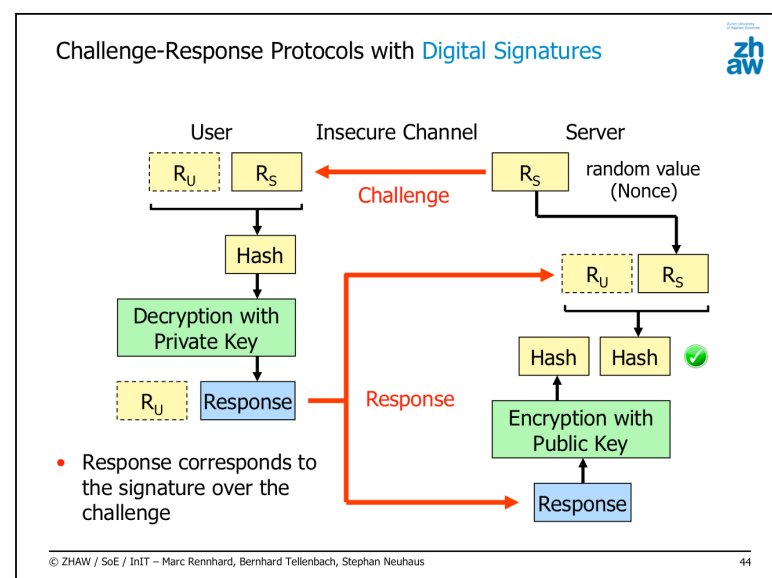
Challenge Response Protocols using Shared Secrets

* Every time a user wants to log into a server
  + Server sends a challenge to the user a nonce
* In order to prevent replay attacks a challenge string should never be used twice
* User creates a response => nonce + keyed hash function initialized with the secret key (or a password)



Challenge Response Protocols using Public Key Digital Signatures

* User forms a hash value out of the random value Rs received from the server as a challenge optionaly a nonce Ru of her own choice
* Decrypting hash value with private key => digital signature => user sends it back as a response to the server



Offline-attack:

* Record the authentication process of a legitimate user to get both challenge and response
* Try different passwords offline until the response matches the recorded response

Key stretching to increase the work factor for the attacker

Pre-compiled attack:

* Condition: user does not include his own nonce
* List of passwords and matching responses for a given challenge
* Attacker makes sure this challenge is sent to the user, e.g by adapting the original challenge sent by the server
* Attacker records the response and compares it with the entries on the list

Passwords have a much smaller entropy than typical cryptographic keys

How to make life of an attacker more difficult => best practices :

* Slow-down online-attacks
* Complicate offline-attacks on password files
  + Salted password hashes
  + Key stretching
* Complicate offline-attacks on password-based challenge-response protocols
  + Include a user challenge in the computation and use key stretching

Cryptographic hash function maps a variable-length message to a fixed-sized hash value

* One way and collision free