Chapter 2: Security Techniques Background

- Secret Key Cryptography
- Public Key Cryptography
- Hash Functions
- Authentication

Chapter 3: Security on Network and Transport Layer

Chapter 4: Security on the Application Layer

Chapter 5: Security Concepts for Networks

2.3: Hash Functions
- Message Digest (MD)
- MD2, MD4, MD5
A cryptographic hash function (also called message digest) is a one-way transformation: \( h: \text{message } m \text{ of arbitrary length} \rightarrow \text{fixed length value } h(m) \)

Properties:
- For any message \( m \), it is easy to compute \( h(m) \)
- Given \( h(m) \), there is no way (cheaper than brute force) to find a \( m \) that hashes to \( h(m) \)
- It is computationally impossible to find two different \( m \) and \( m' \) which hash to the same value \( h(m) \)

It is necessary for the transformation that the output must not be predictable:
- If 1000 inputs are selected at random, any particular bit in the 1000 resulting outputs should be “1” about half the time
- Each output should have about 50% of “1” bits (with high probability)
- If two inputs differ only by one bit, the outputs should look like independently chosen random numbers

Many messages map to the same hash value
Message Digest

The output should not be predictable

- But: It is still possible that two outputs have the same value although the inputs were different

- Birthday Problem:
  - For a single person P1, there are $n = 365$ possible birthdays → The probability $p_1$ of having birthday at one of the days is $n/n$
  - For the second person P2, there are 364 (i.e. $n-1$) possibilities to have birthday at a day different from P1; the probability for this is $p_2 = \frac{n \cdot (n-1)}{n}$
  - Generalizing - the probability of having birthday on different days for $r$ people:
    $$p_r = \frac{n!}{(n-r)!n^r}$$
  - The probability of a match is $1 - p_r$
  - On the average, a match will occur after $\approx \sqrt{n}$ steps

\[ p_r \leq \frac{1}{2} \text{ for } r \geq 23 \]
(for the birthday problem)
Message Digest

• If the message digest has \( k \) bits (i.e. \( 2^k \) different message digests exist), it would take \( 2^{k/2} \) messages, chosen at random, to create two outputs with identical values
  \[ \rightarrow k \geq 128 \text{ because it was considered infeasible to test } 2^{64} \text{ messages with brute force} \]

• If somebody is able (or maybe by pure luck) to create two different messages with the same 128 bit MD, the whole algorithm for MD construction is considered null and void!

History

• With RSA it is possible to digitally sign a message (signature \( \equiv \) encrypt a message with the private key)
• But: computing a signature for a long message with RSA is slow
• Idea: sign message digest rather than the original message
• Message digest algorithms started with public key cryptography (after the invention of RSA)

Note: MD is even used when the message is transmitted in plain text just to ensure integrity
Application of a Message Digest: Authentication

Authentication using a message digest:

- Alice and Bob share a secret $K_{AB}$
- Alice wants to know, if Bob is „still alive“: Alice sends a challenge $r_A$ (a random number)
- Bob concatenates the secret $K_{AB}$ with $r_A$ and calculates a message digest $MD(K_{AB} \mid r_A)$
- Bob sends $MD(K_{AB} \mid r_A)$ to Alice, and Alice checks the result (apply the same procedure)
- Same procedure is applied in the other direction with a challenge $r_B$
Application of a Message Digest: Message Integrity Code

Use Message Digest to generate a MIC:
- Only the appropriate sender is able to compute the appropriate MIC for a message $m$
- Obviously, $MD(m)$ is not a MIC for $m$, since anyone can compute $MD(m)$

→ Compute a MIC considering a secret key $K_{AB}$ (same trick as for authentication):

The MIC can only be computed if $K_{AB}$ is known, i.e. can only be computed and checked for correctness by Alice and Bob
Application of a Message Digest: Encryption

Use Message Digest for encryption:
- Problem: Message digest algorithms are not reversible
- Idea: Generate (pseudo) random numbers using Message Digest and use Vernam Cipher (XOR message and random bit stream)
- Alice and Bob need a shared secret $K_{AB}$:

\[
\begin{align*}
    b_1 &= MD(K_{AB} \mid IV) \\
    b_2 &= MD(K_{AB} \mid b_1) \\
    b_3 &= MD(K_{AB} \mid b_2) \\
    \vdots \\
    b_n &= MD(K_{AB} \mid b_{n-1}) \\
    c_1 &= m_1 \oplus b_1 \\
    c_2 &= m_2 \oplus b_2 \\
    c_3 &= m_3 \oplus b_3 \\
    \vdots \\
    c_n &= m_n \oplus b_n
\end{align*}
\]

- Alice and Bob can compute $b_i$ in advance and need a different IV for further encryption, since it is not secure to use the same bit stream twice
- Variant: use $c_i$ in computing the MD rather than $b_i$

Necessary: partition of the message into „chunks“ $m_1, m_2, \ldots$ whose length is identical to the MD length.
How to compute a Message Digest?

- First idea: convert a secret key algorithm into a message digest algorithm for arbitrary messages.
- Used e.g. to store hashes of UNIX passwords instead of the passwords themselves.
- Given: A secret key algorithm with key bits and message block length $b$ bits (e.g. DES: $k=56$ and $b=64$).

**Algorithm:**
- Split message $m$ into $k$-bit chunks: $m_1$, $m_2$, ...
- Use $m_1$ as a key to encrypt a “constant”
- Use $m_2$ to encrypt the previous result
- ...
- Use the final $b$-bit result as message digest.

- Problem: 64 bit message digest is too short (see birthday problem)
  - Generate a second 64-bit quantity using the chunks $m_1$, $m_2$, ... e.g. in reverse order.
Message Digest 2 (MD2)

Need for a cryptographically secure message digest function
- Ron Rivest developed MD2 (RFC 1319), MD4 (RFC 1320), MD5 (RFC 1321)
- Later: SHS (Secure Hash Standard)
- MD was proprietary and was never published, MD3 was superseeded by MD4

MD2 overview:
- The Input to MD is a message with an arbitrary number of bytes
- The message is padded to be a multiple of 16 bytes
- A 16-byte quantity called checksum is appended
- The MD is computed in a final pass
MD2 Padding

• There must always be padding (even if the length of the original message is a multiple of 16 bytes)
• If the length of message is a multiple of 16 bytes then add 16 bytes of padding
• Else add the necessary number of bytes to make the message a multiple of 16 bytes
• Each padding byte contains the number of padding bytes:

```
padding

original message | r Bytes (1 ≤ r ≤ 16) each containing r

multiple of 16 bytes

... 5 5 5 5 5

43 bytes

End of the message, begin of padding
```

This trick allows to detect the end of the message
MD2 Checksum Computation

- The checksum is similar to a message digest, but not cryptographically secure
- It is a 16-byte value \( C = C_0C_1…C_{15} \), computed as follows:
  - \( C \) is set to 0
  - Process message one byte \( M_n \) a time
  - Let \( m = n \mod 16 \). Byte \( C_m \) of the checksum depends on byte \( M_n \) of the message, byte \( C_{m-1} \) of the checksum and the previous value of byte \( C_m \):
    \[
    C_m := C_m \oplus \pi(C_{m-1} \oplus M_n)
    \]

- The MD2 \( \pi \) substitution is specified in a certain table
MD2 Final Pass

Input: the message $M$ with 16-byte checksum

Algorithm:

- Initialize a 48-byte block $X = X_0, X_1, X_2, \ldots, X_{47}$
- Initialize the first 16 bytes of $X$ to 0
- Process $M$ in 16-byte chunks:
  - Set the second 16 bytes in $X$ to the current message chunk
  - Set the last 16 byte in $X$ to the XOR of the former both parts.
  - Use a compression function to compute an intermediate state for $X$: perform 18 passes, in each pass modifying all 48 byte
- Repeat for each 16-byte chunk in $M$ with the result for $X_0...X_{15}$ from the previous round

Output after processing all chunks in $M$: $\text{MD} = X_0...X_{15}$
**MD2 Compression Function**

For each message chunk:

- **Initial value**: 0

- **Byte “-1”**:
  - Initial value: 0
  - After pass $i$: Byte “-1” = Byte 47 + $i \mod 256$

- **Pass number** (0-17)

- **π substitution**

- **For $n = 0$ to 47**: $C_n := C_n \oplus \pi(C_{n-1})$

- **MD intermediate**

16-byte message chunk

Padded message with appended 16-byte checksum

For next message block

Discarded

Final MD2 after 18 passes for each message chunk

---

**Chapter 2.3: Hash Functions**
MD4

- Works on 32-bit-words (faster processing on modern CPUs than MD2)
- The MD has a length of 128 bit
- Message has to be a multiple of 512 bit (sixteen 32-bit-words):
  - The original message is padded by adding a '1' bit, followed by '0' bits
  - A 64-bit length value for the unpadded message, mod $2^{64}$, is appended

The original message is padded by adding a '1' bit, followed by '0' bits, and a 64-bit length value for the unpadded message, mod $2^{64}$, is appended.

- The message is processed in 512-bit chunks (sixteen 32-bit words)
- The message digest is initialized with a defined intermediate value
- Each step in the message digest computation takes the current intermediate digest value and modifies it using the next chunk of the message
- Each step consists of three passes
MD4 Passes

**Pass 1: Selection function**
- Calculation bases on function $F(x,y,z) = (x \land y) \lor (\neg x \land z)$
- $\neg x$ is the bitwise complement of $x$
- $F$ operates on three parameters of 32 bit, thus one chunk is processed in 16 parts

**Pass 2: Majority function**
- Output of pass 1 is used as input for pass 2
- Calculation bases on function $G(x,y,z) = (x \land y) \lor (x \land z) \lor (y \land z)$

**Pass 3: Final function**
- Output of pass 2 is used as input of pass 3
- Calculation bases on function $H(x,y,z) = x \oplus y \oplus z$

The output of pass 3 is used as input for pass 1 for processing the next message chunk
MD5

- Somebody found a weakness in MD4 if only two passes were used.
- Thus: develop stronger algorithm: MD5.
- MD5 is very similar to MD4, but was designed to be less performant regarding speed and more concerned with security.
- Padding in MD5 is identical to the padding in MD4.
- The major differences are:
  - MD4 consists of three passes for each 16-byte chunk of the message. MD5 makes four passes for each 16-byte chunk.
  - The functions are slightly different, e.g. number of bits in shift operations.
  - MD4 has two constants, one in pass 2, and another in pass 3. MD5 uses a different constant $T_i$ for each message word on each pass. Since there are 4 passes, each of which covers 16 messages, there are 64 32-bit constants used in MD5:
    \[ T_i = \left\lfloor 2^{32} \left| \sin(i) \right| \right\rfloor \text{ for } i = 1 \ldots 64 \]
MD5 Passes

Pass 1: Selection function
- Calculation bases on function \( F(x,y,z) = (x \land y) \lor (\neg x \land z) \)

Pass 2: Second selection function
- Calculation bases on function \( G(x,y,z) = (x \land z) \lor (y \land \neg z) \)

Pass 3: First modification function
- Calculation bases on function \( H(x,y,z) = x \oplus y \oplus z \)

Pass 4: Second modification function
- Calculation bases on function \( I(x,y,z) = y \oplus (x \lor \neg z) \)

MD5 is seen as more secure than MD4, but: why exactly 4 passes were chosen?
Intermediate Conclusion

Several basic security techniques are well researched:

*Secret Key Cryptography*
- Encryption of messages based on a symmetric key
- Standard methods: DES, 3DES, AES

*Public Key Cryptography*
- Usage of an asymmetric key
- E.g. RSA
- Slower than secret key cryptography, thus not useful in encryption
- Useful for secret key exchange, digital signatures, authentication

*Hash functions*
- Computation of a message digest, e.g. for message integrity, authentication
- E.g. MD5

But: how to construct authentication systems which can make use of these techniques?