One-Way Hash Functions

Stanislav Palúch

Fakulta riadenia a informatiky, Žilinská univerzita

7. decembra 2017

Securing of Messages against Errors during Transmission

There are several ways how to prevent changing of messages during transmission by random errors.
A part composed of check characters is added to a message in order to find out wheather the message was changed during transmission.
Some such methods are:

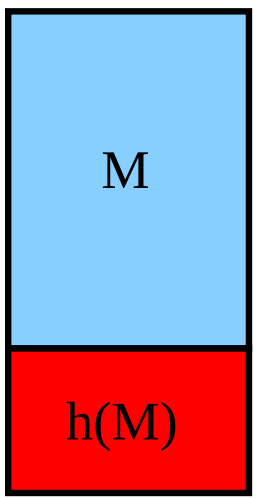
1. Parity check of binary codes
2. Checking modulo 10 of decadic codes
3. Checking modulo 11 of decadic codes
4. Linear (n, k)-codes
5. Check sum – e.g. the sum of all numbers of a message modulo 2^{32}
6. CRC – cyclic redundancy check
7. ...

Coding theory proposes even so called error correction codes capable to repair a small number of errors.
Just mentioned ways are efective methods of securing against random errors but they can not prevent evil minded forgeries of messages.
The attacker can easily change the message in such a way that check characters of forgered message are the same as the ones of the original message.

One Way Hash Functions

Cryptography uses similar approach – it adds additional (reduntant) part called MAC(M) – Message Atentification Code, MD(M) – Message Digest or Fingerprint to sended message M.

Those parts are calculated using one way hash functions.

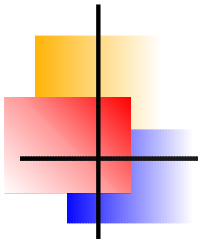


A hash function $h(M)$ has to have the following properties

1. For every M, it is easy to calculate $h(M)$
2. For every h, it is hard to find a message M with $h = h(M)$
3. For every M, it is hard to find such M', $M \neq M'$ such that $h(M) = h(M')$
4. It is hard to find two random messages $M \neq M'$ such that $h(M) = h(M')$

A couple of messages M, M' with property $h(M) = h(M')$ is called **a collision**.

Property 4. is called a **collision resistence**.



Birthday paradox.

Let us have a group of n randomly cosen people and let us investigate the probability of the event that at least two of them have birthday in the same day of the year.

When the number of people reaches 367 then the probability is 1 i.e. 100% (since there are only 366 possible birthdays, including February 29).

However, 99.9% probability is reached with just 70 people, and 50% probability with only 23 people.

These conclusions are based on the assumption that each day of the year is equally probable for a birthday.

Birthday paradox.

Let us have n possible values of $h(M)$ and let us generate at random k messages $M_1, M_2, \ldots, M_k$.

Probability of no collision having only one message M_1 is ..... $p = 1$

Probablility of no collisoin between M_1 and M_2 ..... $p = (1 - \frac{1}{n})$

Probability of no collision after adding message M_3 provided that no collision between M_1 and M_2 occured is ..... $p = (1 - \frac{2}{n})$

.....

.....

Probability of no collision after adding message M_k provided that no collision among $M_1, M_2, \ldots, M_{k-1}$ occured is ..... $p = (1 - \frac{k-1}{n})$

Probability of a Collision

Probability of the event that no collision appeared among k messages is

$$\left(1-\frac{1}{n}\right)\cdot\left(1-\frac{2}{n}\right)\cdot\ \ldots\ \cdot\left(1-\frac{k-1}{n}\right)=\prod_{i=1}^{k-1}\underbrace{\left(1-\frac{i}{n}\right)}_{\approx e^{-\frac{i}{n}}}.$$

$$e^{-x}=1-x+\underbrace{\frac{x^2}{2!}-\frac{x^3}{3!}+\frac{x^4}{4!}-\frac{x^5}{5!}+\ldots}_{\text{this sum is negligible for small } x}$$

$$\prod_{i=1}^{k-1}e^{-\frac{i}{n}}=e^{\sum_{i=1}^{k-1}-\frac{i}{n}}=e^{-\frac{k(k-1)}{2n}}$$

Probability that at least one collision occured is

$$1-e^{-\frac{k(k-1)}{2n}}.$$

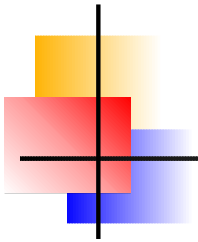
When the Probability of Collision is $> \varepsilon$?

$$
\begin{aligned}
1 - e^{-\frac{k(k-1)}{2n}} &\geq \varepsilon \\
1 - \varepsilon &\geq e^{-\frac{k(k-1)}{2n}} \\
\ln(1-\varepsilon) &\geq -\frac{k(k-1)}{2n} \\
2n\ln(1-\varepsilon) &\geq -(k^2-k) \\
2n\ln(\frac{1}{1-\varepsilon}) &\leq (k^2-k)
\end{aligned}
$$

$$k^2 - k - 2n\ln(\frac{1}{1-\varepsilon}) = 0$$

$$k_{1,2} = \frac{+1 \pm \sqrt{1 + 4.1.2n\ln(\frac{1}{1-\varepsilon})}}{2} =$$

$$= \frac{1}{2} \pm \sqrt{\frac{1}{4} + 2n\ln(\frac{1}{1-\varepsilon})} \approx \pm\sqrt{2n\ln(\frac{1}{1-\varepsilon})}$$



Size of Hash Values

If $k \geq \sqrt{2n \ln(\frac{1}{1-\varepsilon})}$ then the probability of collision is greater than ε.

There are $n = 365$ possibilities for the birthday in one year.
Set $\varepsilon = 0.5$. Then

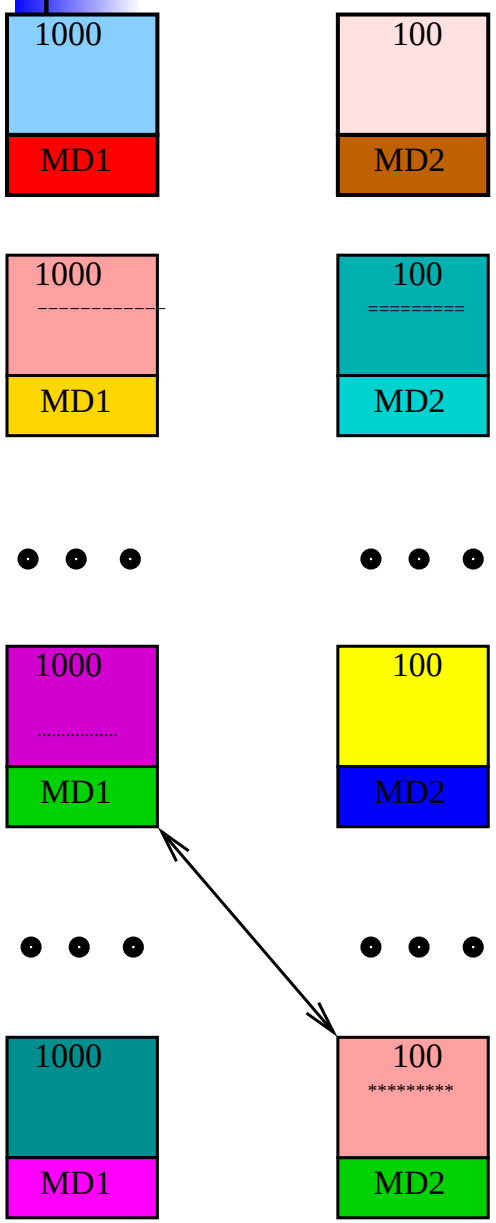
$$k \geq \sqrt{2n \ln(\frac{1}{1-\varepsilon})} = \sqrt{730 \ln(\frac{1}{1-1/2})} == \sqrt{730 \ln(2))} = 22,4944$$

Especially for n and $\varepsilon = 0.5$

$$k \approx \sqrt{n.2.\ln(\frac{1}{2})} \approx 1,17.\sqrt{n}.$$

If a fingerprint of messages was 64 bits long, i.e. $n = 2^{64}$, it suffices to create $1,17 * 2^{32} \approx 5 * 10^9$ random message in order to create a collision with probability at least $1/2$.
Therefore we use hash values 128, 160, 256 and even 512 bits long.

Birthday Attack



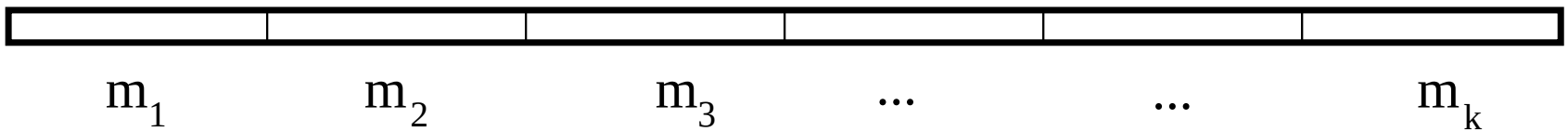
1. Attacker creates two bills of exchange – one to 100 euro, another to 1000 euro.
2. He changes both bills by insignificant changes (adding spaces, adding empty rows etc.) and he such creates another and another variants of both bills until he finds a collision – a couple of a 100-euro bill and a 1000-euro bill with the same hash value h.

   If used hash function has n possible values, then it suffices to create only $1.17\sqrt{n}$ couples of variants of bills in order to find a collision with probability $> \frac{1}{2}$.
3. He gives to the borrower to sign a 100-euro variant with fingerprint h.
4. After some time he reclaims from the borrower 1000 euro on the basis of the fact that the borrower has signed fingerprint h which correspond also to 1000-euro variant.

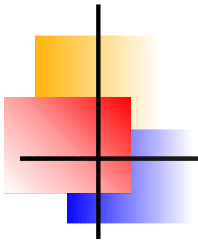
Moral: Allways make a small insignificant change in document before signing it (e.g. add to the day of signing also hour and minute).

General Procedure of Creating Hash Value of a Message

1. Message M is divided into equally large blocks $m_1, m_2, \ldots$.



2. Hash Algorithm has fixed initial vector IV.
   Set $h_0 = IV$.
3. Calculate recursively $h_i = f(m_i, h_{i-1})$.
4. The resulting hash value of all message is $h(M) = h_k$.



Hash Using a Cryptosystem

$$
\begin{aligned}
h_0 &= IV \\
h_i &= f(m_i, h_{i-1})
\end{aligned}
$$

$$
\begin{aligned}
h_i &= E_{h_{i-1}}(m_i) \oplus m_i \\
h_i &= E_{h_{i-1}}(m_i) \oplus m_i \oplus h_{i-1} \\
h_i &= E_{h_{i-1}}(m_i \oplus h_{i-1}) \oplus m_i \\
h_i &= E_{h_{i-1}}(m_i \oplus h_{i-1}) \oplus m_i \oplus h_{i-1}
\end{aligned}
$$

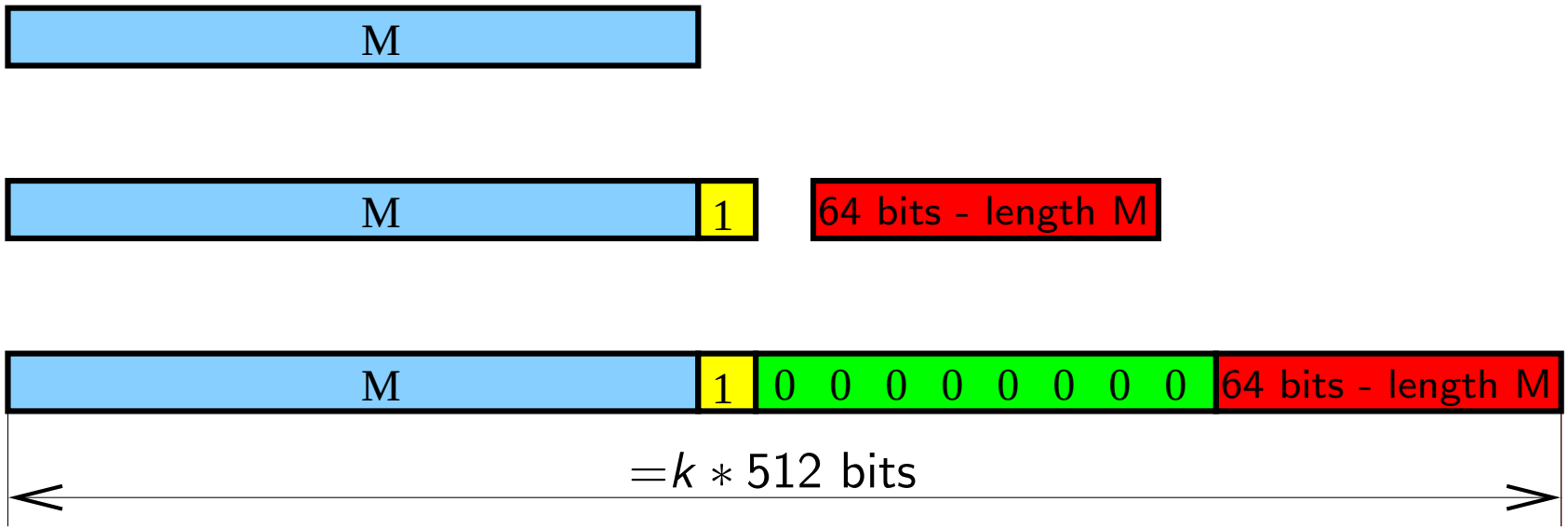
$$
\begin{aligned}
h_i &= E_{m_i}(h_{i-1}) \oplus h_{i-1} \\
h_i &= E_{m_i}(m_i \oplus h_{i-1}) \oplus h_{i-1} \\
h_i &= E_{m_i}(m_i \oplus h_{i-1}) \oplus m_i \oplus h_{i-1} \\
h_i &= E_{m_i}(h_{i-1}) \oplus m_i \oplus h_{i-1}
\end{aligned}
$$

$$
\begin{aligned}
h_i &= E_{m_i \oplus h_{i-1}}(m_i) \oplus m_i \\
h_i &= E_{m_i \oplus h_{i-1}}(h_{i-1}) \oplus h_{i-1} \\
h_i &= E_{m_i \oplus h_{i-1}}(m_i) \oplus h_{i-1} \\
h_i &= E_{m_i \oplus h_{i-1}}(h_{i-1}) \oplus m_i
\end{aligned}
$$

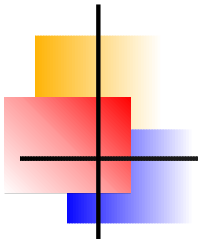
Broken scheme $h_i = E_{m_i}(h_{i-1})$.

MD4

A message must be adapted before calculation as follows:



1. One bit with value 1 is added at the end of the message.
2. A 64-bit number containing the length of the message is created. The addapted message will end by this number.
3. Several zeros are inserted between added 1 and 64-bits of message length in such a way that length of resulting adapted message is equal to the integer multiply of 512.



MD4

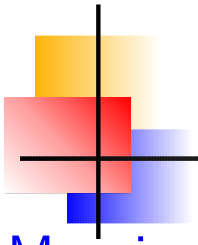
- The length of fingerprint of algorithm MD4 is 128 bits, i.e. h_i has 128 bits.
- Algorithm works with h_i as with a quadruple (A, B, C, D) of 32-bit numbers
- Processed block length of message is 512 bits.
- Algorithm works with block m_i as with 16

$$X[0], X[1], X[2], \ldots, X[15]$$

  32-bit numbers.
- First initial value of $h_0 \equiv (A, B, C, D)$ is set
- i-th 512-bit block m_i of message is expressed in the form of 16 32-bit numbers $X[0], X[1], X[2], \ldots, X[15]$ a recurently is calculated

$$h_i = f(m_i, h_{i-1})$$

- if m_k is the last block of the message then h_k is returned as the fingerprint of whole message.



MD4 – Function $f(m_i, h_{i-1})$

Meaning of used operations:

$+$ – addition mod 2^{32}
$\wedge$ – bitwise logical and
$\vee$ – bitwise logical or
$\neg$ – bitwise logical negation

MD4 uses following functioons:
$f(X, Y, Z) = (X \wedge Y) \vee ((\neg X) \wedge Z)$
$g(X, Y, Z) = (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z)$
$h(X, Y, Z) = X \oplus Y \oplus Z$

Setup of 128-bit o initial vector
$IV = (A, B, C, D)$:
$A = 67452301$
$B = efcdab89$
$C = 98badcfe$
$D = 10325476$

Funkction $h_i = f(m_i, h_{i-1})$

0 Input $(A, B, C, D) = h_{i-1}$,
$(X[0], X[1], \ldots, X[15]) = m_i$

1. Storing $h_{i-1} = (A, B, C, D)$
$AA = A$
$BB = B$
$CC = C$
$DD = D$

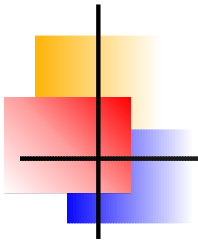
2. 1.round$(A, B, C, D, X[0-15])$

3. 2.round$(A, B, C, D, X[0-15])$

4. 3.round$(A, B, C, D, X[0-15])$

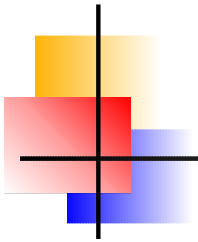
5. $A = A + AA$
$B = B + BB$
$C = C + CC$
$D = D + DD$

6 Return $h_i = (A, B, C, D)$



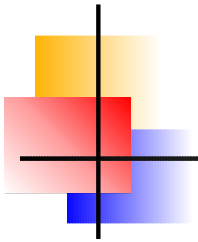
1. round MD4

$$
\begin{aligned}
1.\quad & A = (A + f(B,C,D) + X[0]) <<< 3 \\
2.\quad & D = (D + f(A,B,C) + X[1]) <<< 7 \\
3.\quad & C = (C + f(D,A,B) + X[2]) <<< 11 \\
4.\quad & B = (B + f(C,D,A) + X[3]) <<< 19 \\
5.\quad & A = (A + f(B,C,D) + X[4]) <<< 3 \\
6.\quad & D = (D + f(A,B,C) + X[5]) <<< 7 \\
7.\quad & C = (C + f(D,A,B) + X[6]) <<< 11 \\
8.\quad & B = (B + f(C,D,A) + X[7]) <<< 19 \\
9.\quad & A = (A + f(B,C,D) + X[8]) <<< 3 \\
10.\quad & D = (D + f(A,B,C) + X[9]) <<< 7 \\
11.\quad & C = (C + f(D,A,B) + X[10]) <<< 11 \\
12.\quad & B = (B + f(C,D,A) + X[11]) <<< 19 \\
13.\quad & A = (A + f(B,C,D) + X[12]) <<< 3 \\
14.\quad & D = (D + f(A,B,C) + X[13]) <<< 7 \\
15.\quad & C = (C + f(D,A,B) + X[14]) <<< 11 \\
16.\quad & B = (B + f(C,D,A) + X[15]) <<< 19
\end{aligned}
$$



2. round MD4

$$
\begin{aligned}
1.\quad & A = (A + g(B,C,D) + X[0] + 5a827999) <<< 3 \\
2.\quad & D = (D + g(A,B,C) + X[4] + 5a827999) <<< 5 \\
3.\quad & C = (C + g(D,A,B) + X[8] + 5a827999) <<< 9 \\
4.\quad & B = (B + g(C,D,A) + X[12] + 5a827999) <<< 13 \\
5.\quad & A = (A + g(B,C,D) + X[1] + 5a827999) <<< 3 \\
6.\quad & D = (D + g(A,B,C) + X[5] + 5a827999) <<< 5 \\
7.\quad & C = (C + g(D,A,B) + X[9] + 5a827999) <<< 9 \\
8.\quad & B = (B + g(C,D,A) + X[13] + 5a827999) <<< 13 \\
9.\quad & A = (A + g(B,C,D) + X[2] + 5a827999) <<< 3 \\
10.\quad & D = (D + g(A,B,C) + X[6] + 5a827999) <<< 5 \\
11.\quad & C = (C + g(D,A,B) + X[10] + 5a827999) <<< 9 \\
12.\quad & B = (B + g(C,D,A) + X[14] + 5a827999) <<< 13 \\
13.\quad & A = (A + g(B,C,D) + X[3] + 5a827999) <<< 3 \\
14.\quad & D = (D + g(A,B,C) + X[7] + 5a827999) <<< 5 \\
15.\quad & C = (C + g(D,A,B) + X[11 + 5a827999]) <<< 9 \\
16.\quad & B = (B + g(C,D,A) + X[15] + 5a827999) <<< 13
\end{aligned}
$$



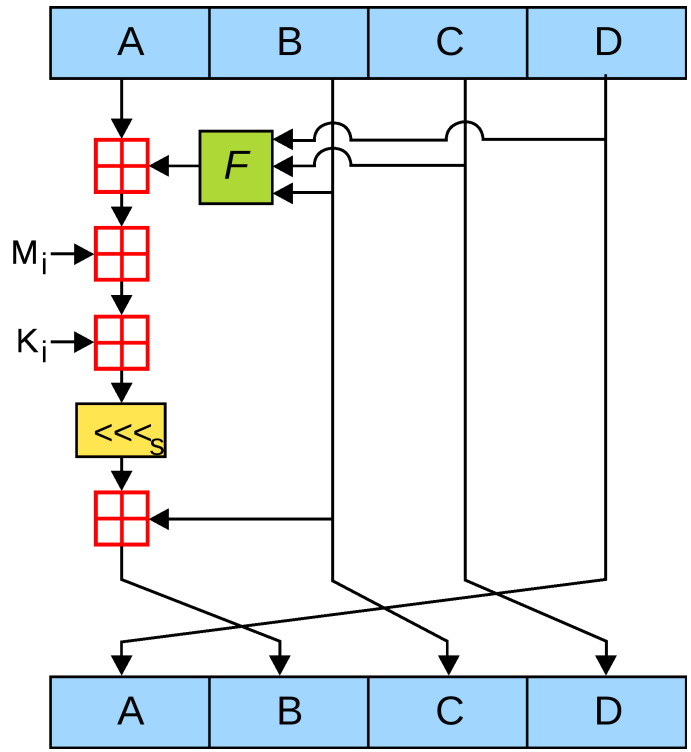
3. round MD4

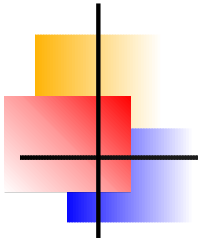
$$
\begin{aligned}
1.\quad & A &=& \ (A + h(B,C,D) + X[0] \ + 6ed9eba1) <<< 3 \\
2.\quad & D &=& \ (D + h(A,B,C) + X[8] \ + 6ed9eba1) <<< 9 \\
3.\quad & C &=& \ (C + h(D,A,B) + X[4] \ + 6ed9eba1) <<< 11 \\
4.\quad & B &=& \ (B + h(C,D,A) + X[12] + 6ed9eba1) <<< 15 \\
5.\quad & A &=& \ (A + h(B,C,D) + X[2] \ + 6ed9eba1) <<< 3 \\
6.\quad & D &=& \ (D + h(A,B,C) + X[10] + 6ed9eba1) <<< 9 \\
7.\quad & C &=& \ (C + h(D,A,B) + X[6] \ + 6ed9eba1) <<< 11 \\
8.\quad & B &=& \ (B + h(C,D,A) + X[14] + 6ed9eba1) <<< 15 \\
9.\quad & A &=& \ (A + h(B,C,D) + X[1] \ + 6ed9eba1) <<< 3 \\
10.\quad & D &=& \ (D + h(A,B,C) + X[9] \ + 6ed9eba1) <<< 9 \\
11.\quad & C &=& \ (C + h(D,A,B) + X[5] \ + 6ed9eba1) <<< 11 \\
12.\quad & B &=& \ (B + h(C,D,A) + X[13] + 6ed9eba1) <<< 15 \\
13.\quad & A &=& \ (A + h(B,C,D) + X[3] \ + 6ed9eba1) <<< 3 \\
14.\quad & D &=& \ (D + h(A,B,C) + X[11] + 6ed9eba1) <<< 9 \\
15.\quad & C &=& \ (C + h(D,A,B) + X[7] \ + 6ed9eba1) <<< 11 \\
16.\quad & B &=& \ (B + h(C,D,A) + X[15] + 6ed9eba1) <<< 15
\end{aligned}
$$

MD5 Algorithm

MD5 was designed by Ronald Rivest in 1991 to replace an earlier hash function MD4

1. It is a strengthening of algorithm MD4.
2. It produces a 128-bit hash.
3. It works with 512-bit block of message.
4. It has 4 rounds instead of 3.
5. It uses changed funcions:
   $F(X, Y, Z) = (X \wedge Y) \vee (\neg X \wedge Z)$
   $G(X, Y, Z) = (X \wedge Z) \vee (Y \wedge \neg Z)$
   $H(X, Y, Z) = X \oplus Y \oplus Z$
   $I(X, Y, Z) = Y \oplus (X \vee \neg Z)$
6. It runs 30% slower than MD4





Security of MD5

The security of the MD5 hash function is severely compromised.

A collision attack exists that can find collisions within seconds on a computer with a 2.6 GHz Pentium 4 processor (complexity of $2^{24.1}$). The ability to find collisions has been greatly aided by the use of Graphics Processing Units.

MD5 is not suitable for applications like SSL certificates or digital signatures that rely on its collision resistence.

SHA algoritmus

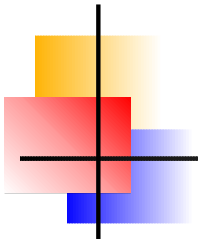
SHA-1 (Secure Hash Algorithm 1) is a cryptographic hash function designed by the United States NSA[1] and is a U.S. Federal Information Processing Standard published by the U. S. NIST[2].

1. SHA-1 produces a 160-bit (20-byte) hash value.
   A SHA-1 hash value is typically rendered as a hexadecimal number, 40 digits long.
2. It processes 512-bit block of message $W[0], W[1], \ldots, W[15]$. This block is expanded for $16 \geq j \leq 79$ as follows

$$W[j] = W[j-3] \oplus W[j-8] \oplus W[j-14] \oplus W[j-16]$$

3. It has 4 rouds every with 20 steps

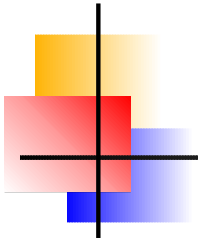
[1] National Security Agency
[2] National Institute of Standards and Technology



SHA-1

```
Initialize hash value for this chunk:
a = h0; b = h1; c = h2; d = h3; e = h4;
Main loop:
for( i=0; i < 80; i++)
  {if (0 <= i) and (i <= 19) then   {f = (b and c) or ((not b) and d;)
                                     k = 0x5A827999;}
  else if( 20<=i) and (i <= 39)     {f = b xor c xor d;
                                     k = 0x6ED9EBA1;}
  else if(40 <= i) and (i <= 59) {f = (b and c) or (b and d)or(c and d);
                                     k = 0x8F1BBCDC;}
  else if(60<= i) and (i <= 79)   {f = b xor c xor d;
                                     k = 0xCA62C1D6;}
  temp = (a leftrotate 5) + f + e + k + w[i];
  e = d; d = c; c = b leftrotate 30; b = a; a = temp;
}

  Add this chunk's hash to result so far:
  h0 = h0 + a; h1 = h1 + b; h2 = h2 + c; h3 = h3 + d; h4 = h4 + e;
```



SHA –Security

On 23 February 2017, Google announced the SHAttered attack, in which they generated two different PDF files with the same SHA-1 hash in roughly $2^{63.1}$ SHA-1 evaluations.
This attack is about 100,000 times faster than brute forcing a SHA-1 collision with a birthday attack, which was estimated to take 2^{80} SHA-1 evaluations.
The attack required the equivalent processing power as 6500 years of single-CPU computations and 110 years of single-GPU computations".

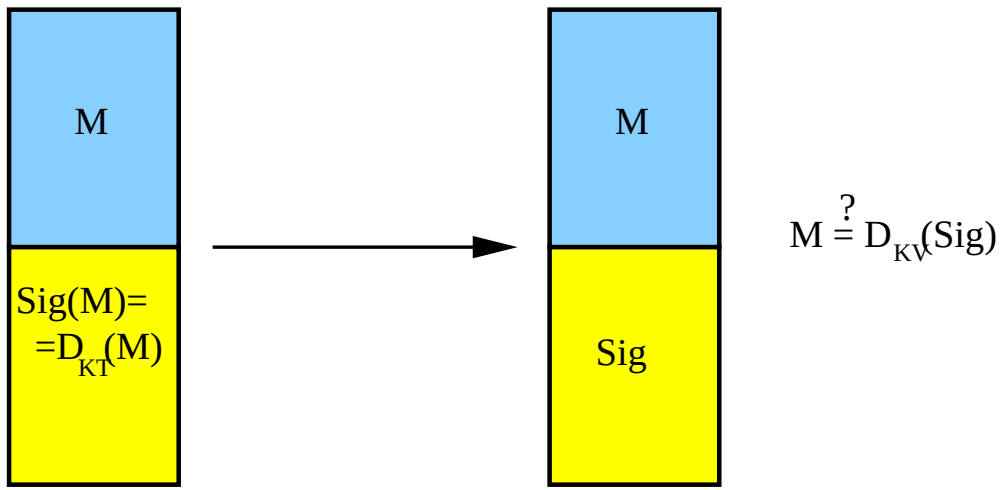
NIST[3] added three additional hash functions in the SHA family.
The algorithms are collectively known as SHA-2, named after their digest lengths (in bits):
SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256.
SHA-2 includes significant changes from its predecessor, SHA-1.

The NIST hash function competition selected a new hash function, SHA-3, in 2012. The SHA-3 algorithm is not derived from SHA-2.

[3]National Institute of Standards and Technology

Digital Signature

- The sender A with public key KV_A and private key KT_A signes his message M such that he attaches the result of deciphering of message M with his private key KT_A: $Sig(M) = D_{KT_A}(M)$.



- The receiver B tests the autenticity of signature so that he encipheres the signature with public key KV_A of the sender – calculates

$$M' = E_{KV_A}(Sig(M))$$

and check wheather $M = M'$.

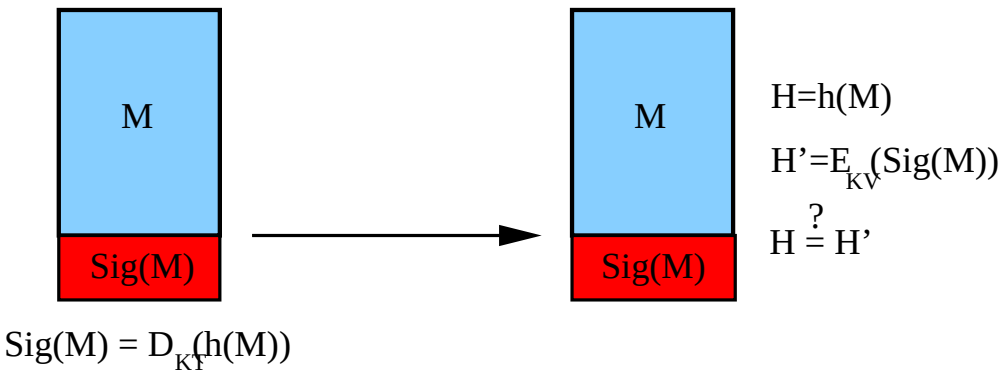
If $M' \neq M$, then the message was forged or the signarure is not genuine.

If $M' = M$, then the signature is genuine and message is unchanged.

The only person – the sender A – could create $Sig(M) = D_{KT_A}(M)$ for message M since he is the only participant who has private key KT_A.

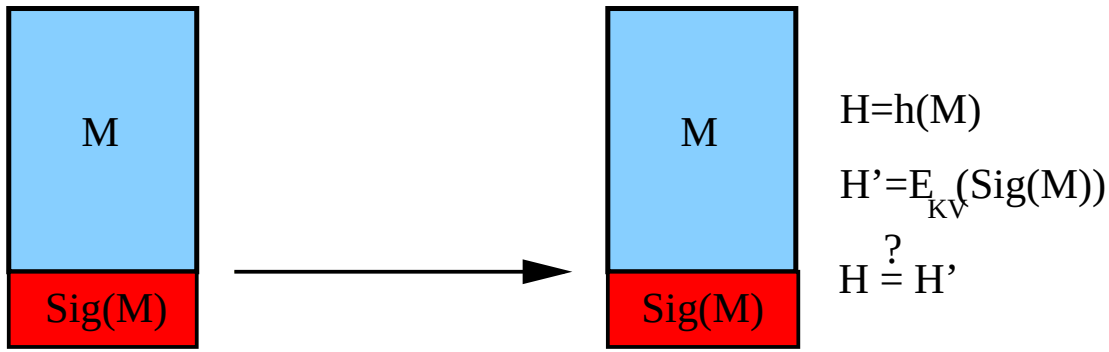
Digital Signature

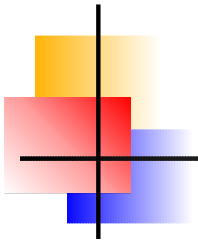
- The sender A with couple of public and private key KV_A, KT_A signes his message M so that
  1. Calculates fingerprint $h(M)$ of the message M.
  2. Encipheres figerprint $h(M)$ by his private key:
     $$Sig(M) = D_{KT_A}(h(M)).$$
  3. Attaches $Sig(M)$ to message M as his digital signature



- The receiver B checks the autencticity of signature as follows
  1. Calculates fingerprint $H = h(M)$ of received message M
  2. Enciphers signature $Sig(M)$ by public key of sender A:
     $H' = E_{KV_A}(Sig(M))$.
  3. Checks wheather $H' = H$.
     If $H' \neq H$, then the message was forged or signature is not genuine.
     Af $H' = H$, then signature is geniune and message is not changed.
     The is the only man – participant A – who could create
     $Sig(M) = D_{KT_A}(h(M))$ to message M since he is the only one who has the private key KT_A.

Digital Signature

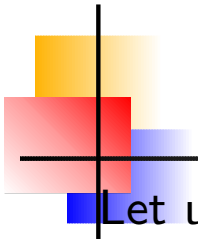




Time Stamp

In many situations, people need to certify that a document existed on a certain date. In some cases, the party that signed the earliest copy of document winns the case (e.g. contract of sale) in other cases (e.g. last will) the latest document is valid.

- Add a part $X = MD(M), PUB$ to the document M where $X = MD(M)$ is the fingerprint of M and PUB is a publicly known information which originated in the day of signing the message M. Existence of PUB proves, that the document could not be signed earlier since iformation PUB was not known before.
- Add $Y = sig(MD(M), PUB)$ as a signature.
- Publish the tripple $MD(M)$, PUB, $sig(MD(M), PUB)$ in a newspaper. This proves that we have signed the document with fingerprint $MD(M)$, information PUB and signature $sig(MD(M), PUB)$ not later then in the day of publishing.



One-Way unctions

Let us have an equation

$$2^x = a.$$

with unknown x in the domain of real numbers
The solution of above equation is $x = \log_2(a)$.

Similarily, the solution of equation $z^x = a$, where $z > 0$ and x is unknown is $x = \log_z(a)$.

Problem of Discrete Logarithm

We know the prime number p and numbers $0 < a < p$ and $1 < s < p$.
For what integer exponenet x it holds

$$s^x = a \mod p?$$

Another formulation: We are searching a solution x of equation:

$$s^x = a$$

in the field $\mathbb{Z}_p$.
This problem is called a **discrete logarithm problem**.
Discrete logarithm problem is a hard problem for large prime p.

Diffie - Hellmann Key Exchange

A a **B** make an agreement about a large prime number p and number s, $1 < s < p$.

Numbers s, p can be public and can be used repeatedly even for more participants.

A

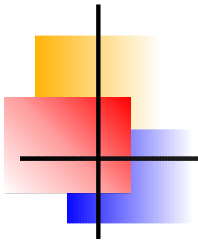
- Chooses $a < p$ secret.
- Calculates $\alpha = s^a \mod p$.
- Transmits α to **B** .
- Receives β.
- Calculates key $K_A = \beta^a \mod p$.

B

- Chooses $b < p$ secret.
- Calculates $\beta = s^b \mod p$.
- Transmits β to **A** .
- Receives α.
- Calculates key $K_B = \alpha^b \mod p$.

Does it realy hold $K_A = K_B$?
It holds:

$$K_A = \beta^\alpha = \left(s^b\right)^a = s^{ab} = (s^a)^b = \alpha^b = K_B \mod p$$



Diffie - Hellmann Key Exchange

A peril: Intruder in the middle attacack

$$\mathbf{A} \xrightarrow{\alpha=s^a} \mathbf{X} \xrightarrow{\alpha'=s^{a'}} \mathbf{B}$$

$$\mathbf{A} \xleftarrow{\beta'=s^{b'}} \mathbf{X} \xleftarrow{\beta=s^b} \mathbf{B}$$

$$\mathbf{A} \overset{K_1=s^{ab'}}{\longleftrightarrow} \mathbf{X} \overset{K_2=s^{a'b}}{\longleftrightarrow} \mathbf{B}$$

Intruder **X** captures communication between **A** and **B** and during key exchange procedure sends to **B** instead of α his α' and to **A** his β' instead of β.
Then **X** communicate with **A** as **B** with key $K_1 = s^{ab'}$ and with **B** as **A** with key $K_2 = s^{a'b}$.

Passwords

User	Computer
password	Checks wheather it matches with stored password
password	Checks wheather it matches MD(password) matches with stored MD

The first way is unacceptibe !!!
A programmer, who has made such error, should be sacked in seconds and never allowed to write any commercial apllication.
PASSWORDS MUST NOT BE STORED IN OPEN FORM.

Dictionary attack

- first names
- geografical names
- astronomical terms
- mythical persons
- persons from holy scriptures
- chemical elements
- names of days and months
- names of famous actors, singers, musicians, painters, etc.
- titles of books

User	Salt	MD(Heslo, Salt)
peterp	$EA1DFC48_H$	128 bitov