Hashing, One-Time Signatures, and MACs

Digital Signatures

A signature is (according to the Miriam-Webster dictionary):

1. (a) The act of signing one's name to something
   (b) the name of a person written with his or her own hand
2. a feature in the appearance or qualities of a natural object formerly held to indicate its utility in medicine
3. (a) a letter or figure placed usually at the bottom of the first page on each sheet of printed pages (as of a book) as a direction to the binder in arranging and gathering the sheets
   (b) one unit of a book comprising a group of printed sheets that are folded and stitched together
4. the part of a medical prescription that contains the directions to the patient
5. something (as a tune, style, or logo) that serves to set apart or identify; also : a characteristic mark

Digital Signatures (cont.)

Actually, signature encompases two functionalities:

- Writing the name of a person, in his own hands, as a confirmation.
- Commitment.

Distinguish between

- Identification: Assures the identity.
- Commitment: Assures the commitment.

Digital Signatures (cont.)

It is possible to have identification without commitment, and vice versa:

- An anonymous letter has neither.
- A company letter has an identifying title.
- A check is a commitment, even if it has no identification.

Digital Signatures (cont.)

A **Digital signature $S(M)$**:

1. Computable by the signer for any message M.
2. Everybody (and the receiver in particular) can verify its originality.
3. It is impossible to forge a signature.
4. The signer cannot claim that a message he signed is forged.

One Way Functions

Informal Definition:A **one way function** $Y = f(X)$ is a function which is efficient to calculate but difficult to invert: for a given Y it is difficult to find any X such that $Y = f(X)$.
Note: There is no relationship between a one way function and an invertible function.
Example:$Y = f(X) = \mathrm{AES}_X(0)$ is a one way function, if there is no successful attack on AES which finds the key X from the ciphertext Y.

Lamport and Diffie's Signature Scheme

Preparation:

1. A one way function $Y = f(X)$ is selected.
2. Each user U chooses $2n$ random values $X_0, X_1, \ldots, X_{2n-1}$, and computes $Y_0, Y_1, \ldots, Y_{2n-1}$ by $Y_i = f(X_i)$.
3. U publishes the vector $Y = (Y_0, Y_1, \ldots, Y_{2n-1})$ in a public file under his name (i.e., in a newspaper, or in a public file maintained by a trusted center).
4. U publishes in advance as many vectors as the number of signatures he is expected to sign.

Lamport and Diffie's Signature Scheme (cont.)

Signature generation:

1. A wants to sign an n-bit message M to B ($M = m_0 m_1 \ldots m_{n-1}$).
2. A chooses one of his unused vectors from the public file, and sends it to B.
3. B verifies the existence of the vector in the public file.
4. A and B mark the vector as used in the public file.
5. A computes the signature $S = S_0 S_1 \ldots S_{n-1}$ by

$$S_i = \begin{cases} X_{2i}, & \text{if } m_i = 0; \\ X_{2i+1}, & \text{if } m_i = 1 \end{cases}$$

and sends the signature S to B.

Lamport and Diffie's Signature Scheme (cont.)

Signature verification:

1. B verifies whether for all the i's

$$f(S_i) = \begin{cases} Y_{2i}, & \text{if } m_i = 0; \\ Y_{2i+1}, & \text{if } m_i = 1 \end{cases}$$

Lamport and Diffie's Signature Scheme (cont.)

Proof to a judge (and anybody else):

1. B sends the signature S and the vector Y to the judge.
2. The judge verifies that the vector Y appears in the public file as a vector of A.
3. The judge verifies whether for all the i's

$$f(S_i) = \begin{cases} Y_{2i}, & \text{if } m_i = 0; \\ Y_{2i+1}, & \text{if } m_i = 1 \end{cases}$$

Lamport and Diffie's Signature Scheme (cont.)

Security:

If B can forge A's signature, he can invert the one way function f!

Even if he is already given a signature of some message using some vector, still he needs to invert the one way function f in order to forge a different message using the same vector.

Hashing

Problem:To sign a long message of 1,000,000 bits, a vector of 2,000,000 $f(X_i)$ should be prepared in advance. The length of the signature is 128,000,000 bits if $f(X) = \mathrm{AES}_X(0)$.
Solution:Hashing.

Hashing (cont.)

Definition:A **cryptographic hash function**, or briefly a **hash function**, is a function $H : \{0,1\}^* \to \{0,1\}^n$ for some constant n, which satisfies:

1. It is easy to compute $H(M)$ for any M.
2. Given $h \in \{0,1\}^n$, it is computationally difficult to find a **preimage**: a message $M \in \{0,1\}^*$ such that $h = H(M)$.
3. It is computationally difficult to find a **collision**: a pair of messages M_1 and M_2 such that $H(M_1) = H(M_2)$.

Hashing (cont.)

Usage:Given a long message M, we sign $H(M)$.
Other Applications of Hash Functions:

- Keeping $H(M)$ can protect a long message M against modification.
- The "one-wayness" property can be used in protocols, where it is required that nobody can invert the function.
- Publishing $H(M)$ can be used as a commitment on M.

Hashing (cont.)

Claim:Forging a signature $S(H(M))$ is difficult.

1. If the attacker chooses M, he can compute $H(M)$ but cannot sign it.
2. If he chooses $H(M)$, he can neither sign, nor find M.
3. If he has a valid signature on M_1, he knows $H(M_1)$ and the signature $S(H(M_1))$. If he can find another message M such that $H(M) = H(M_1)$, he has M's signature, but it is difficult to find such an M.

Rabin's Hashing using DES

Let a message $M = m_1 m_2 \ldots m_l$, where each m_i is 56-bit long. Let S_0 be some standard constant.

$$
\begin{aligned}
n_1 &= \mathrm{DES}_{m_1}(S_0) \\
n_2 &= \mathrm{DES}_{m_2}(n_1) \\
n_3 &= \mathrm{DES}_{m_3}(n_2) \\
&\vdots \\
n_l &= \mathrm{DES}_{m_l}(n_{l-1}) \\
H(M) &\stackrel{\Delta}{=} n_l
\end{aligned}
$$

Rabin's Hashing using DES (cont.)

Drawback:This function changes the DES key every block. Changing DES keys is inefficient in most DES hardware and software.
Security:This hash function is not secure (using DES).

- It is easy to find a collision: in about 2^{32} messages, the birthday paradox predicts that with probability higher than half there are two distinct messages hashing to the same value.
- Preimages X can be found for any hash value h. (Hint: build X from two halves, and use the birthday paradox).
- Rabin's hashing is secure when used with (secure) ciphers whose block size is at least 128 bits (e.g., AES).

The Required Hash Size

This method suggests that the hash function should be collision free (paragraph 3 in the definition should hold).

1. B chooses a pair of messages M_1 and M_2 satisfying $H(M_1) = H(M_2)$, where M_1 is a message that A will accept and sign, and M_2 is a message which B prefers, but A will not agree to sign.
2. B requests A to sign $H(M_1)$.
3. A signs $S(H(M_1))$.
4. B receives $S(H(M_1))$, and then concludes that the signature on M_2 is $S(H(M_2)) = S(H(M_1))$.
5. B can claim in court that A signed on M_2.

Alternatively, A can choose such two messages, sign one of them, and later claim in court that he signed the other message.

The Required Hash Size (cont.)

How to find a pair of messages satisfying $H(M_1) = H(M_2)$?

Assume the hash value size is $n = 64$ bits.

B chooses 2^{32} messages which A will accept $M_1,\ldots,M_{2^{32}}$, and 2^{32} messages which A will not accept $M'_1,\ldots,M'_{2^{32}}$.

The Required Hash Size (cont.)

For example, B chooses 2^{32} messages M_i, which differ in 32 words, each of them has two choices:

The bank A $\left\{\begin{array}{l}\text{will}\\ \text{promises to}\end{array}\right\}$ $\left\{\begin{array}{l}\text{give}\\ \text{let}\end{array}\right\}$ B an amount of 100 $\left\{\begin{array}{l}\text{US}\\ \text{American}\end{array}\right\}$ dollars $\left\{\begin{array}{l}\text{before}\\ \text{until}\end{array}\right\}$ April 2011. $\left\{\begin{array}{l}\text{Then,}\\ \text{Later,}\end{array}\right\}$ B will $\left\{\begin{array}{l}\text{use}\\ \text{invest}\end{array}\right\}$ this amount for ...

and 2^{32} messages M'_j of the form:

The bank A $\left\{\begin{array}{l}\text{will}\\ \text{promises to}\end{array}\right\}$ $\left\{\begin{array}{l}\text{give}\\ \text{let}\end{array}\right\}$ B the amount of at least $\left\{\begin{array}{l}\text{twenty}\\ \text{forty}\end{array}\right\}$ $\left\{\begin{array}{l}\text{million}\\ \text{billion}\end{array}\right\}$ $\left\{\begin{array}{l}\text{US}\\ \text{American}\end{array}\right\}$ dollars $\left\{\begin{array}{l}\text{which}\\ \text{that}\end{array}\right\}$ are given as present, and $\left\{\begin{array}{l}\text{should}\\ \text{will}\end{array}\right\}$ not be returned ...

The Required Hash Size (cont.)

By the birthday paradox, there is a high probability that there is some pair of message M_i and M'_j such that $H(M_i) = H(M'_j)$. Both messages have the **same signature**.

Conclusion:The hash value size must be at least $n = 128$ bits, for which the birthday paradox requires about 2^{64} complexity to find such a pair.

Notice also that by the birthday paradox there is a high probability for a collision of two elements of the same set when the size of the set is about the square root of the number of different possible elements.

The Birthday Paradox

Assume that H can have m distinct outputs ($m = 2^n$), and assume that for each input value H choose the output at random, independently from the output of the other inputs.

We can look at H as a function which throw a ball into a set of m boxes, and the ball enters to one of the boxes at random (to the box $H(i)$).

If we throw k balls, we receive m^k assignments of the balls into the boxes. Only $m(m-1)(m-2)\cdots(m-k+1)$ of them do not include any pair of balls in the same box.

Thus, the probability that there will not be any collision is

$$\frac{m!}{(m-k)!m^k}$$

The Birthday Paradox (cont.)

and the probability of one or more collisions is

$$\begin{aligned} p(m,k) &= 1 - \frac{m!}{(m-k)!m^k} \\ &= 1 - \frac{(m-1)(m-2)\cdots(m-k+1)}{m^{k-1}} \\ &= 1 - \left(1-\frac{1}{m}\right)\left(1-\frac{2}{m}\right)\cdots\left(1-\frac{k-1}{m}\right) \end{aligned}$$

but for any $0 < X < 1$,

$$1 - X < 1 - X + X^2\left(1-\frac{X}{3}\right)/2 + X^4\left(1-\frac{X}{5}\right)/24 + \ldots = e^{-X}$$

The Birthday Paradox (cont.)

and thus $(1-\frac{i}{m}) < e^{-\frac{i}{m}}$:

$$\begin{aligned} p(m,k) &= 1-\left(1-\frac{1}{m}\right)\left(1-\frac{2}{m}\right)\cdots\left(1-\frac{k-1}{m}\right) \\ &> 1-e^{-\frac{1}{m}}e^{-\frac{2}{m}}\cdots e^{-\frac{k-1}{m}} \\ &= 1-e^{-\frac{1+2+\ldots+(k-1)}{m}} = 1-e^{-\frac{k(k-1)}{2m}} \end{aligned}$$

For a large k, in order to get $p(m,k) > \frac{1}{2}$:

$$k \geq \sqrt{2m\ln 2} = 1.17\sqrt{m}.$$

The Birthday Paradox (cont.)

When we deal with two distinct subsets of a global set, and one of these subsets is chosen randomly from the global set, a similar result holds:
Let X by the global set with N elements. Let S_1 be a subset of X (chosen either in deterministic or random way) of size $\sqrt{N}$, and let S_2 be a subset of X with $\sqrt{N}$ elements chosen randomly. Then, with probability $1 - \frac{1}{e} = 63.2\%$ there is at least one element s such that $s \in S_1$ and $s \in S_2$.

Exercise:Prove the above claim.
Find the minimal value t satisfying that if $|S_1| = |S_2| = t$ then there is an element in the two subsets with probability at least 50%.

The Birthday Paradox (cont.)

Example:A birth date: there are 365 days in a year, thus in a group of $\sqrt{2 \cdot 365 \cdot \ln 2} = 22.5$ children, there are two children with the same birthday with probability about half.

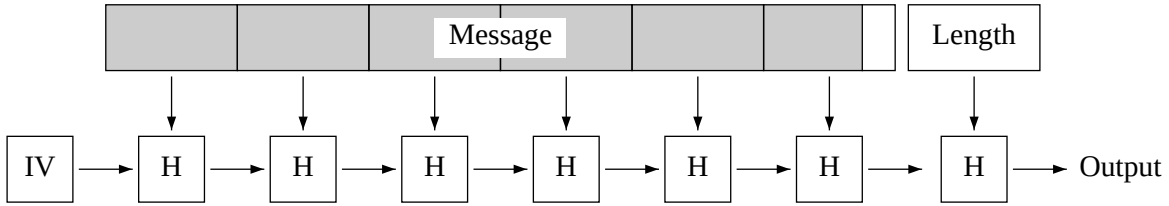
Example:A hash function with 64-bit hash value: $n = 64$, $m = 2^{64}$. We should compute about $\sqrt{2 \cdot 2^{64} \cdot \ln 2} = 1.17 \cdot 2^{32}$ hashes to find a collision with probability about half.

If $m = 2^{128}$, $1.17 \cdot 2^{64}$ hashes are required.

Example:Out of n users of a system, about $\sqrt{n}$ are dishonest. A system administrator picks users at random and check whether they are dishonest. It is expected that after checking $\sqrt{n}$ users he would find at least one dishonest user with probability about 63%.

Hash Functions

Most practical hash functions $h(M)$ divide the messages M into fixed-length blocks M_1, M_2, etc., pad the last block and append the message length to the last block. The resultant last block (after all paddings) is denoted by M_n. Then, the hash function applies a collision free function H on each of the blocks sequentially.



The function H takes as inputs the result of the application of H on the previous block (or a fixed initial value in the first block), and the block itself, and results with a hash value. The hash value is an input to the application of H on the next block.

Hash Functions (cont.)

The result of H on the last block is the hashed value of the message $h(M)$.

$$
\begin{aligned}
h_0 &= IV = \text{a fixed initial value} \\
h_1 &= H(h_0, M_1) \\
&\vdots \\
h_i &= H(h_{i-1}, M_i) \\
&\vdots \\
h_n &= H(h_{n-1}, M_n) \\
h(M) &\stackrel{\Delta}{=} h_n
\end{aligned}
$$

Hash Functions (cont.)

Theorem:If H is collision free, then also h is collision free.
Proof:By contradiction. Assume the contrary. Thus, either

1. For a given Y, it is possible to find a message M such that $h(M) = Y$. In this case also $H(h_{n-1}, M_n) = Y$, and thus H is not collision free. Contradiction.

Hash Functions (cont.)

2. It is possible to find two distinct messages M and M^* such that $h(M) = h(M^*)$. Let k be the smallest integer $k > 0$ such that either $h_{n-k-1} \neq h^*_{n^*-k-1}$ or $M_{n-k} \neq M^*_{n^*-k}$. In both cases $h_{n-k} = h^*_{n^*-k}$, and thus $H(h_{n-k-1}, M_{n-k}) = H(h^*_{n^*-k-1}, M^*_{n^*-k})$, and thus H is not collision free.

   We remain with the case in which one message (without loss of generality) M^* is a postfix of the second message M, and $h_{n^*-n} = IV$. However, in this case we actually find $H(h_{n^*-n-1}, M_{n^*-n}) = IV$, which is impossible for collision free functions. Contradiction.

QED

Practical Hash Functions

Two approaches for the design of hash functions are:

1. To base the function H on a block cipher.
2. To design a special function H, not based on a block cipher.

The second approach is the more popular nowadays.

Practical Hash Functions (cont.)

Hash function of the second approach include:

1. Snefru (128–224 bits) (broken, 1990).
2. MD4 (128 bits) (broken, 1995).
3. MD5 (128 bits) (broken, 2004).
4. The Secure Hash Standard (SHA, SHA-1) (160 bits) (broken, 2004, 2005).
5. The Secure Hash Standard SHA-224, SHA-256, SHA-384, and SHA-512 (224, 256, 384, 512 bits, respectively)
6. RIPEMD (160 bits).
7. Tiger (192 bits).

MD5, RIPEMD, and all SHA's are based on the structure of MD4 with various improvements.

SHA-1

The Secure Hash Standard was designed by the NSA, following the structure of Rivest's MD4 and MD5. The first standard was SHA (now called SHA-0). It was later changed slightly to SHA-1, due to some unknown weakness found by the NSA.

Step 1:Append padding bits: Given an m-bit message, a single bit "1" is appended as the $m+1$th bit and then $(448-(m+1)) \bmod 512$ (between 0 and 511) zero bits are appended. As a result, the message becomes 64-bit shy of being a multiple of 512 bits long.

Step 2:Append length: A 64-bit representation of the message length m is appended, making the result a multiple of 512 bits long.

The result is divided into 512-bit blocks, denoted by $M_1, M_2, \ldots, M_n$.

SHA-1 (cont.)

Step 3:The five 32-bit words A, B, C, D and E are used to keep the 160-bit hash values h_i.
Their initial value (h_0) is (in hexadecimal)

$$\begin{aligned} A &= \texttt{67452301} \\ B &= \texttt{EFCDAB89} \\ C &= \texttt{98BADCFE} \\ D &= \texttt{10325476} \\ E &= \texttt{C3D2E1F0}. \end{aligned}$$

Step 4:For each block $X = M_i$, the function $H(h_{i-1}, X)$ is applied on the previous value of $h_{i-1} = (A, B, C, D, E)$ and the block. The result remains in $h_i = (A, B, C, D, E)$.
Step 5:The hash value is the 160-bit value $h_n = (A, B, C, D, E)$.

The Function H of SHA-1

1. Divide $X = M_i$ into 16 32-bit words: W_0, W_1, W_2, $\ldots$, W_{15}.

2. for $t = 16$ to 79 compute $W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \lll 1$, where $X \lll Y$ (cyclicly) rotates X to the left by Y bits.

   Remark:The one-bit rotate in computing W_t was not included in SHA, and is the only difference between SHA and SHA-1.

3. Save A as AA, B as BB, C as CC, D as DD, and E as EE.

4. For $t = 0$ to 79 do

   (a) $T = A \lll 5 + f_t(B, C, D) + E + W_t + K_t$.

   (b) $E = D$, $D = C$, $C = B \lll 30$, $B = A$, $A = T$.

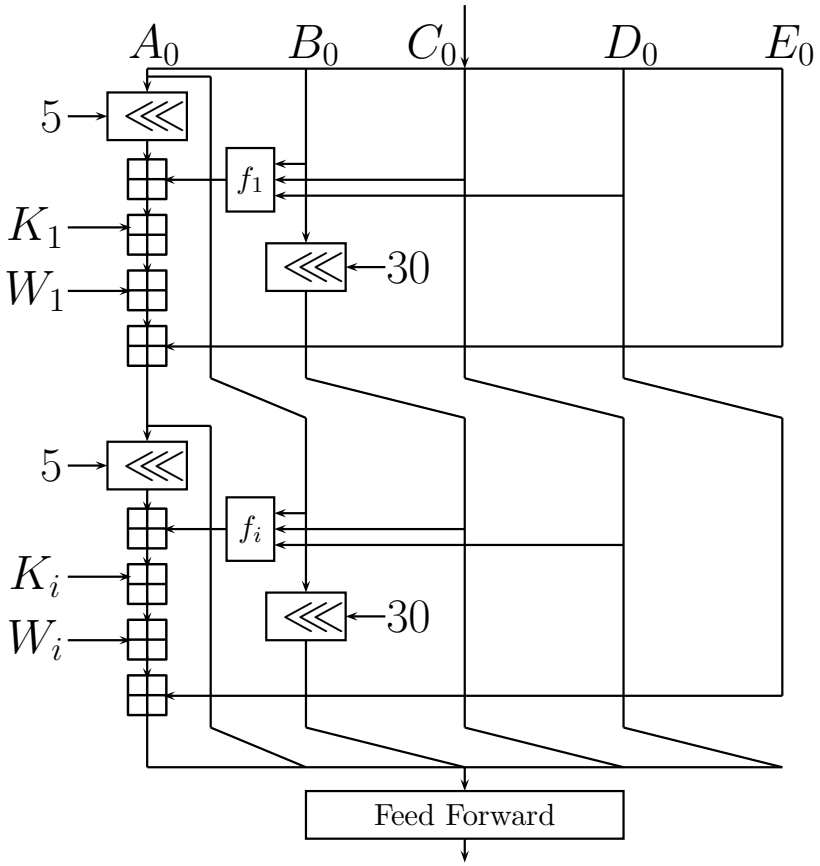
5. Perform $A = A + AA$, $B = B + BB$, $C = C + CC$, $D = D + DD$, and $E = E + EE$ (modulo 2^{32}).

The Function H of SHA-1 (cont.)

6. The function f_t and the values K_t used above are:

$$
\begin{array}{rll}
0 \le t \le 19: & f_t(X,Y,Z) = XY \vee (\neg X)Z & K_t = \texttt{5A827999} \\
20 \le t \le 39: & f_t(X,Y,Z) = X \oplus Y \oplus Z & K_t = \texttt{6ED9EBA1} \\
40 \le t \le 59: & f_t(X,Y,Z) = XY \vee XZ \vee YZ & K_t = \texttt{8F1BBCDC} \\
60 \le t \le 79: & f_t(X,Y,Z) = X \oplus Y \oplus Z & K_t = \texttt{CA62C1D6}
\end{array}
$$

The Function H of SHA-1 (cont.)



Message Authentication Codes

Message authentication codes (MAC) are used to protect information against modification. They mix the messages cryptographically under a secret key, and the result (the MAC) is appended to the message. The receiver can then recompute the MAC and verify its correctness. It should be impossible for an attacker to forge a message and still be able to compute the correct MAC without knowing the secret key.

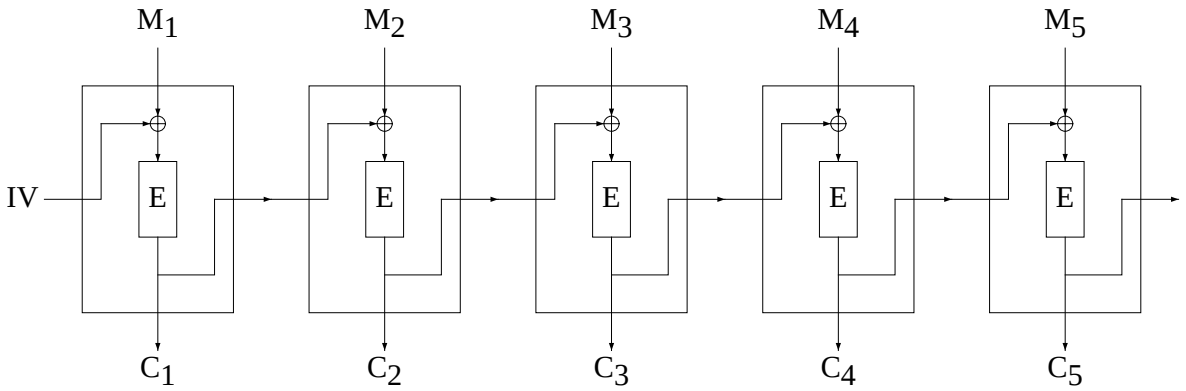
The purpose is similar to signing messages against forging, however, usually signature schemes are much slower, and MAC schemes are as fast as symmetric encryption.

CBC MAC

One very useful MAC function used in the industry (and adopted by standard committees) is the CBC-MAC.
This MAC computes a CBC mode on the data (under a key designated for authentication),

$$C_i = E_K(M_i \oplus C_{i-1}),$$

and takes the last block (or two blocks, or half a block) as the MAC value.



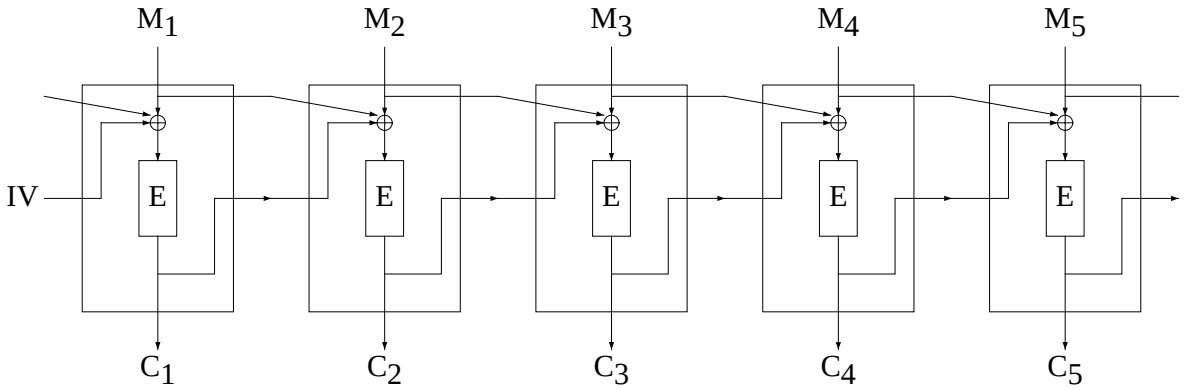
Example: PCBC MAC

Another MAC which was used by early versions of Kerberos was PCBC, which was intended to unify encryption and MAC together. (CBC cannot be used for encryption and MAC with the same key, as forgers can keep the last ciphertext blocks unchanged; therefore it requires two CBC mode computations: one for encryption and another for MAC).

Example: PCBC MAC (cont.)

PCBC is similar to the CBC mode, but also feeds the previous message block into the next one, increasing the mixing of the data.
PCBC computes

$$C_i = E_K(M_i \oplus M_{i-1} \oplus C_{i-1}).$$



It seems that this MAC is even better than the CBC MAC, due to the additional mixing, that ensures that errors in ciphertext transmission propagate further.

Example: PCBC MAC (cont.)

However, it was later found that exchanging the order of the ciphertext blocks (thus modifying the rest of the message blocks in some unpredictable way):

$$D_K(C_i) \oplus M_i = M_{i-1} \oplus C_{i-1}.$$

And thus

$$M_i \oplus C_i = (C_i \oplus D_K(C_i)) \oplus M_{i-1} \oplus C_{i-1}$$

from which we get

$$M_n \oplus C_n = IV \oplus \sum_{j=1}^{n} C_i \oplus D_K(C_i)$$

where IV is the initial value (i.e., $IV = M_0 \oplus C_0$).
From this equation it is easy to see that the order of the ciphertext blocks does not change the final MAC value.

MACs Using Hash Functions

MACs can be built using hash functions. One such possibility can be to prepend the key to the message and to hash them together:

$$\mathrm{MAC1}_K(M) = H(K\|M).$$

In this construction, it is easy to append data to the end of a message and predict the MAC of the longer message without knowing the key.
A better solution is

$$\mathrm{MAC2}_K(M) = H(K\|M\|K).$$

Even better solutions require using the hash function twice.

HMAC

HMAC is a generic MAC which use an hash function to compute a MAC.

$$\text{HMAC-H}_K(M) = H\left(K \oplus \text{opad} \| H\left((K \oplus \text{ipad}) \| M\right)\right),$$

where opad is a block of 64 bytes 36_x and ipad is a block of 64 bytes $5c_x$. It accepts a variable length key K, to which zeroes are appended to form a full block.

The instance using a hash function H it is called HMAC-H.

The most known MAC in the HMAC family is HMAC-MD5, which serves as the standard MAC in the Internet, including in IPSEC. HMAC-SHA-1 is also used.

Remark:MD5 is similar to SHA-1, but with 4 words only (A, B, C, D), smaller number of rounds (64), and slightly different round functions.