Computer and Network Security

Computer & Information Sciences & Engineering
University Of Florida
Gainesville, Florida 32611-6120
nemo@cise.ufl.edu

Hashes
(Pfleeger Ch. 3, KPS Ch. 4)

Art

1 Definitions

1.1 One-way Function

A one-way function F is a function that is "hard" to invert. That is, if

$$F : A \to B,$$

then given some $b \in B$, it is hard to find an $a \in A$ for which

$$F(a) = b.$$

By hard, we mean that no method much faster than trying elements of A using brute force is known to invert F effectively.

This is *collision-resistance*.

1.2 Hash Function

A *hash* H is a function that

- takes inputs from a large set, A, and
- maps them to fixed length elements in a finite set, B.

1.3 One-way or Secure Hash Function

If a hash function is also a one-way function, then it is a *one-way hash* or *secure hash* function.

2 Uses of Hashes

If H is a one-way hash, then it may be used in many ways:

1. for authentication
2. as a MIC (message integrity check)
3. as a MAC (message authentication check)
4. as a PRNG (for key stream generation)
5. for password security.

3 Hash Attacks

3.1 The Birthday Problem

If there are 23 or more people in a room, the odds are better than .5 that two of them will have the same birthday.

- Assume that 365 days of the year are equally likely as birthdays
- birthdays are random among people

- With N people in the room, there are

$$P = N(N-1)/2$$

distinct pairs of people.

- For each pair, there is a probability of

$$p = 1/365$$

that the two have the same birthday.

- Expected number of matches is number of pairs times probability,

$$E(matches) = Pp.$$

- For the expected number of matches to exceed .5,

$$P > .5/p = 365/2 = 183.$$

- Thus

$$P = N(N-1)/2 > 365/2$$

hence

$$N(N-1) > 365$$
$$N > \sqrt{365} > 19.$$

- With 20 people in a room it is around an even bet that two will have the same birthday.

3.2 So What?

Well, the Birthday Problem tells us

1. that to find two messages with the same n-bit hash value, only $2^{n/2}$ candidates will have to be considered, on the average;
2. if another message m' with the same n-bit hash value $H(m)$ as some given message m is required, then on the average, 2^{n-1} candidates will have to be tested.

Since hashes must protect against intentional misuse (a user producing two messages with the same hash that mean very different things, then either

- signing one and claiming that the other was actually sent later, or
- getting someone else to sign the one and then sending the other with the signature of the first),

they must be *twice as long* as we would otherwise need for security.

4 Why use a hash?

One-way hashes are

1. small and fast to compute;
2. collision-resistant;
3. may be used with public key systems for signatures much faster than signing the entire message or document;
4. export a little better than pure cryptosystems do, yet can be used for encryption.

5 Authentication

Rather than using a challenge-response authentication method based on encrypting the challenge nonces, hashing may be used instead.

$$
\begin{aligned}
A \rightarrow B &: r_A \\
B \rightarrow A &: H(K_{AB}|r_A)
\end{aligned}
$$

1. Alice sends Bob a nonce challenge,
2. Bob replies with the hash of a common secret K_{AB} concatenated with the nonce (R_A)

Alice can then verify the hash received, and no party that does not have access to K_{AB} is likely to be able to reproduce that value. Bob can authenticate Alice similarly.

NOTA BENE: r_A must be used ONLY once by Alice, and there are a host of other subtleties that must be considered in authentication.

6 MICs and MACs

Using a one-way hash for a Message Integrity Code (MIC):

- $H(m)$ by itself does not suffice, since H is known to all
- $H(K|m)$ may be used, so that only those who know K can either generate or check the integrity of m.
- Unfortunately, several popular MD algorithms use as an intermediate value when computing the hash the same quantity that is used as the hash output at the end. This means that, given a message m and its hash, $H(m)$, some addition A may be made to m and a valid hash computed for it by initializing the MD algorithm to have $H(m)$ as its carried information.

Hashing for MICs

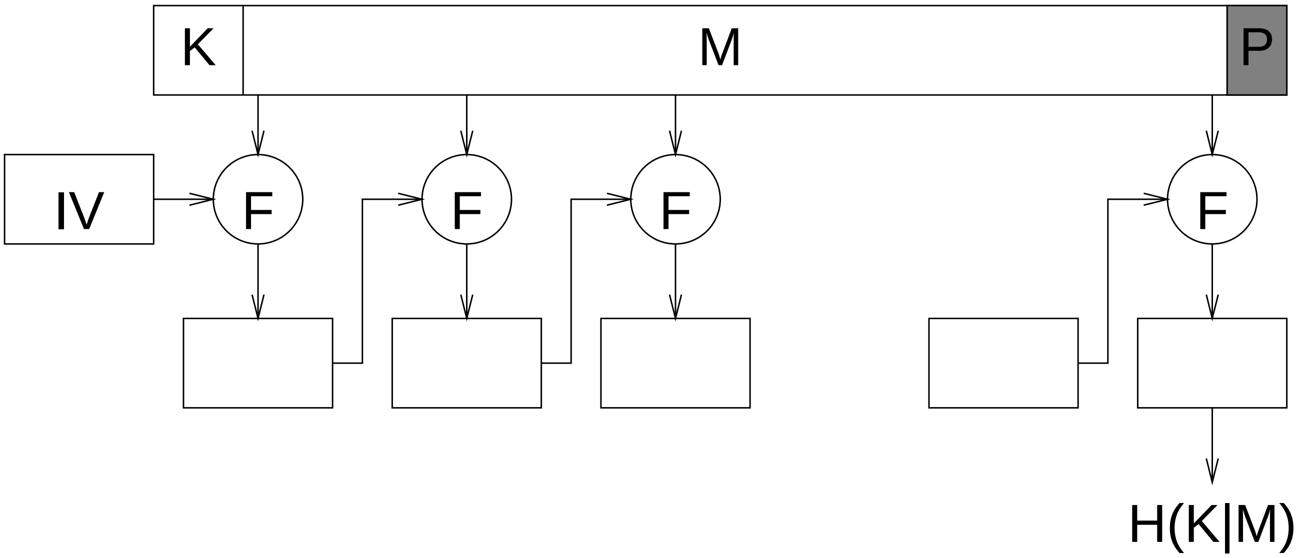

Figure 1: Secure Hash for MIC

Attack on MICs

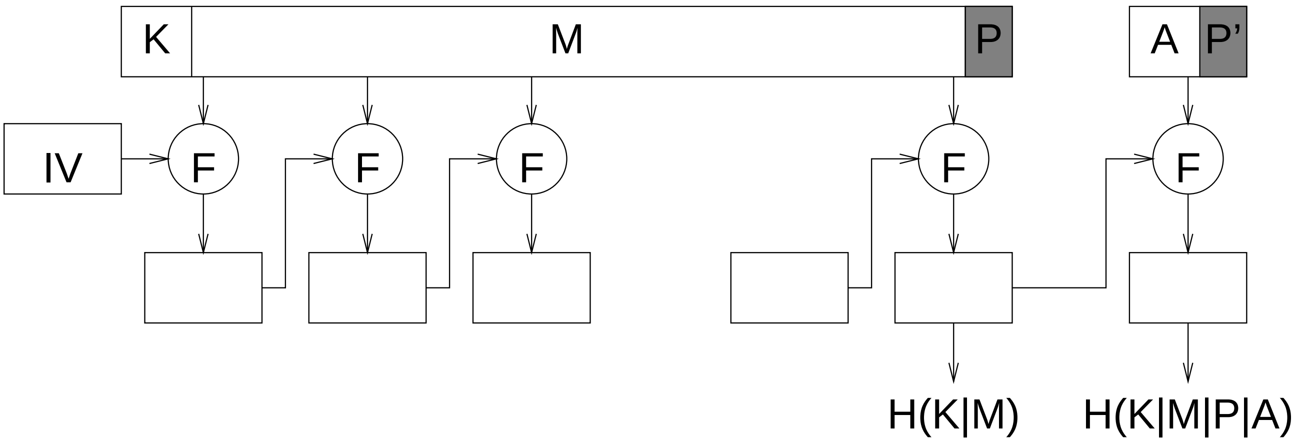

Figure 2: Attack on MIC

This is not a problem with MD2, and is easily solved in any of several ways. Two ways are as follows.

1. Use $H(m|K)$ instead.

2. Use only half of the computed $H(m)$ as the MIC

If only one other party knows the secret K, then the MIC serves as a MAC as well, authenticating that the message was sent by the only other party to have the secret.

7 Encryption with Hashes

Hash function output is unpredictable, so may be used as PRNG in a Vernam Cipher.

7.1 OFB

- $K_0 = IV$, the initialization vector, which is the key
- $K_i = H(K_{i-1}), i = 1, 2, ...$
- $C_i = P_i + K_i, i = 1, 2, ...$
- $P_i = C_i + K_i, i = 1, 2, ...$

7.2 CFB

- $C_0 = IV$
- $C_i = P_i + H(K, C_{i-1}), i = 1, 2, ...$
- $P_i = C_i + H(K, C_{i-1}), i = 1, 2, ...$

7.3 Hash in Feistel Structure

They may be used as round function in Feistel structure

7.3.1 Karn's approach

- Plaintext $P = L|R$ (two halves, L and R, concatenated)
- Key $K = l|r$ (two halves, l and r, concatenated)
- Ciphertext $C = D|E$ where
  - $E = R + H(L|l)$
  - $D = L + H(E|r)$
- Decryption given $K = l|r$ and $C = D|E$,
  - $L = D + H(E|r)$
  - $R = E + H(L|l)$

7.3.2 Weakness in Karn's Approach

Luby and Rackoff: Karn's approach susceptible to attack if two plaintext messages P and P' had the same left half, $P = L|R$ and $P' = L'|R'$ where $L = L'$.

If the attacker knows all the ciphertext, all of P and that $L' = L$, then it is easy to find R'.

7.3.3 A Fix for Karn's Approach

- $K = l|r$ as before
- $P = L|R$ as before
- $X = R + H(l|L)$
- $Y = L + H(r|X)$
- $Z = X + H(l|Y)$
- $C = Z|Y =$ the ciphertext

7.3.4 Other Ways to Fix Karn's Approach

Ensure that $L' \neq L$ ever, by prepending

- Timestamp (must protect time, resolution)
- Random number (must never reuse)
- Sequence number (must never wrap around, must remember)

8 Password Security with Hashes

8.1 Hashing Passwords

How to store information to verify passwords?

- in plaintext (bad idea)
- encrypted (could be decrypted if key were compromised)
- cryptographic hashes of passwords

In Unix and some other systems, password is

1. converted into the key for an encryption algorithm (modified version of DES in the case of Unix),
2. key used to encrypt a constant (0 in the case of Unix)
3. encryption may also be modified by a *salt* (12 bits in Unix)
4. so same password with different salts has different hash values (modifies the DES data expansion algorithm in Unix)

Hashing Passwords

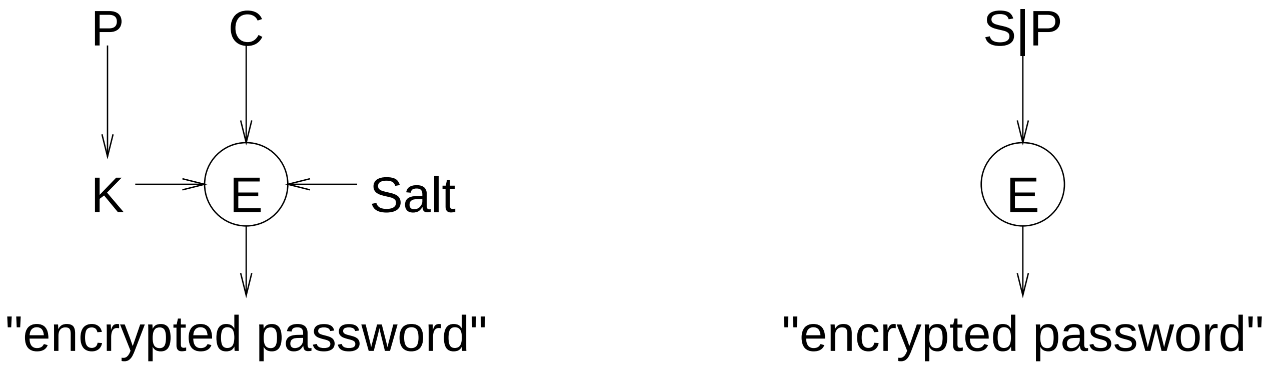

Figure 3: Hashing Passwords for Storage

8.2 Lamport's One-time Passwords

Goal: to provide a means for a central computer to verify the identity of a user in such a way that even if the verification information kept on the computer is read, the attacker cannot fool the system.

Approach: Uses one-way hashes as follows.

1. User U picks password P and lifetime N
2. U passes P through hash function N times to produce P_N:

$$P_N = H(H(H...H(P)...))\text{N times} = H^N(P).$$

3. P_N is securely given to host C along with N; C stores

$$\langle U, N, P_N \rangle$$

When U wants to authenticate herself to C,

1. U sends request to C giving her name, U
2. C replies with the number $N-1$
3. U computes $P_{N-1} = H^{N-1}(P)$ and sends this to C as message M
4. C then computes $H(M)$ and verifies that
$$H(M) = P_N$$
5. C then decrements N and replaces P_N with P_{N-1}

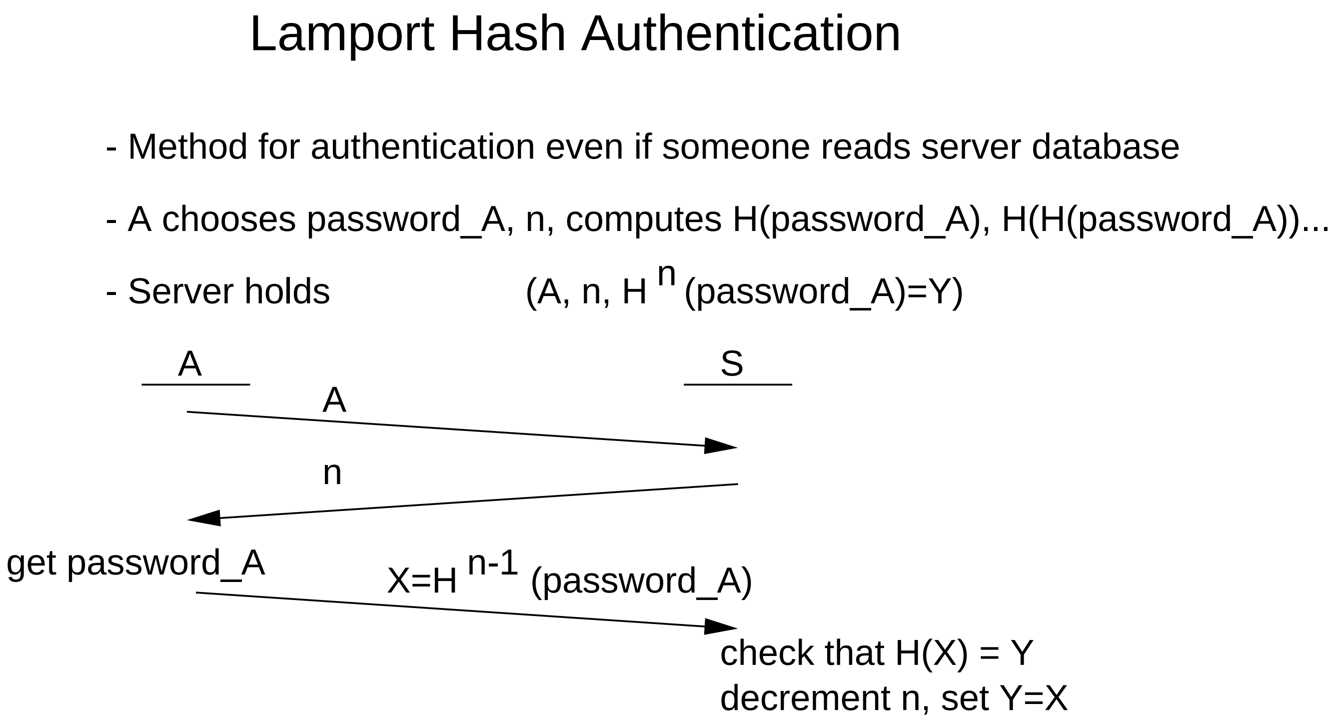

Figure 4: Lamport One-Time Hash Authentication

1. H is one-way, so only U could have produced the preimage of P_N, even if P_N and H are known.
2. Susceptible to man-in-the-middle attack
3. Requires reinitialization when N = 1
4. Must be modified if more than one host is used with same P

9 Using encryption as a hash function

1. message broken into key-sized blocks
2. blocks used as keys in the encryption fuction to encrypt repeatedly some constant (once for each key)
3. If output is too small for a usable hash, then
   (a) use more than one constant, or
   (b) reverse the key order, etc.

Hashing Using Encryption

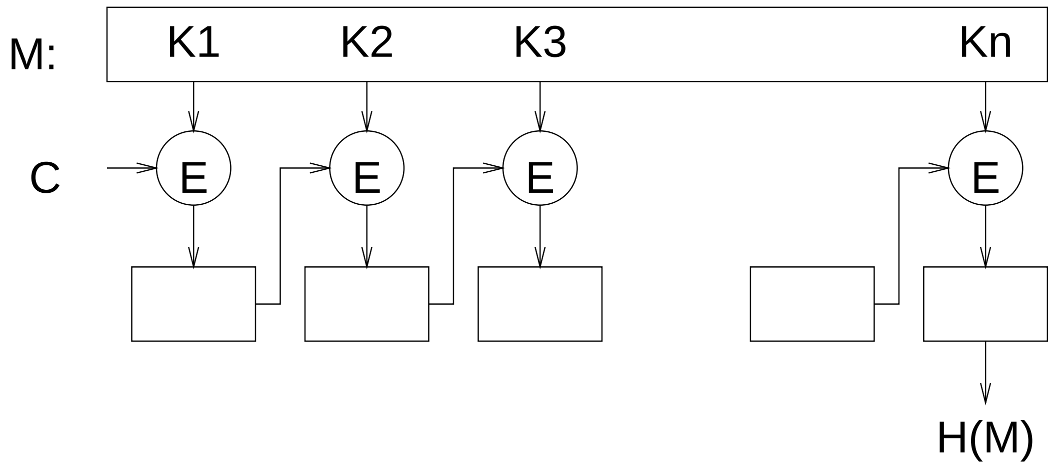

Figure 5: Hashing Using Encryption

10 Selected Hash Functions

10.1 Generally

There are a few steps that most hash functions must always take.

1. Arbitrary (bit or byte) length input is padded to a multiple of the block length - padding must be consistent but does not have to be removeable
2. Some checksum may be added to the the padded input (MD2)
3. Padded input is processed in blocks
4. Blocks may have multiple passes made over them
5. Output of preceding block is passed as initialization information for processing the next block

Generic Hash Function Operation

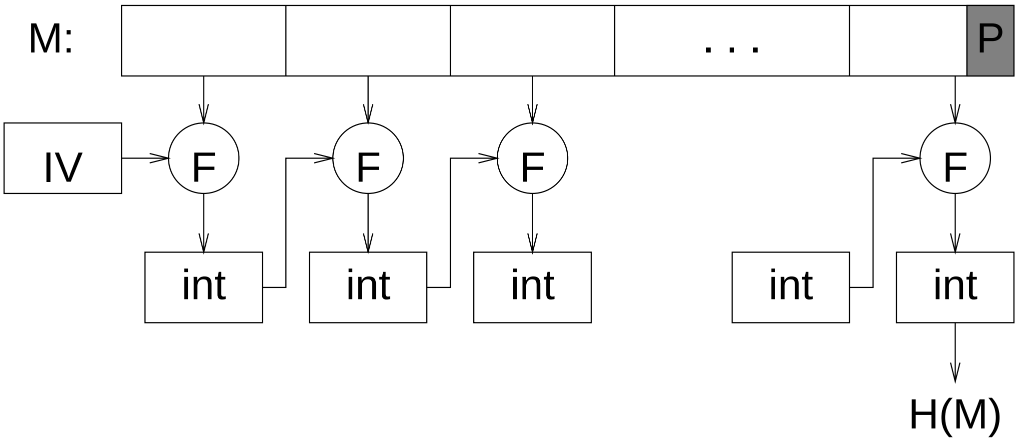

Figure 6: Generic Hash Function

10.2 MD2

1. byte-oriented computations
2. 128-bit digest
3. Arbitrary byte length input is padded to a multiple of 16 bytes. (last p bytes, $1 \leq p \leq 16$, all have value p in binary)
4. A 16-byte "checksum" is computed over the whole message and appended to its end
5. The padded, checksummed message is then processed in 16-byte chunks
6. carrying forward 128-bit intermediate results (p_i)
7. 48 bytes are processed per chunk (m_i,p_i,$m_i \oplus p_i$)
8. with 18 passes at each stage
9. to produce 16-byte digest

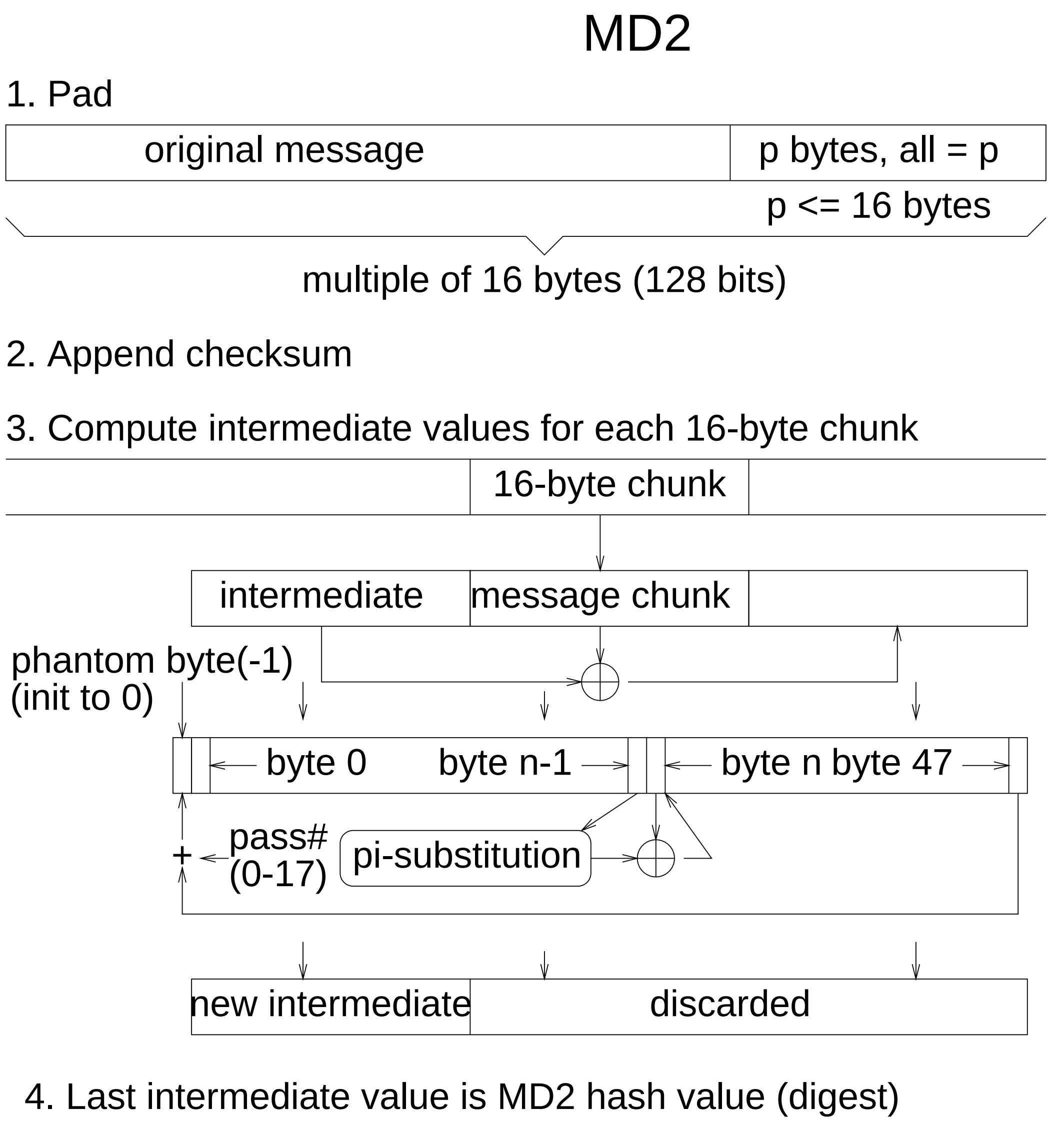

Figure 7: MD2 structure

10.3 MD4

1. 32-bit word computations
2. 128-bit digest
3. (almost) arbitrary bit length input
4. padded to 64 bits less than than a multiple of 64 bytes (512 bits)
5. 64-bit length field added to end
6. processed in 16-byte chunks,
7. carrying forward 128-bit intermediate results
8. with 3 passes per stage
9. with same constants for each message word in passes 1 and 3
10. to produce 16-byte digest

10.4 MD5

1. 32-bit word computations
2. 128-bit digest
3. (almost) arbitrary bit length input
4. padded to 64 bits less than than a multiple of 64 bytes (512 bits)
5. 64-bit length field added to end
6. processed in 16-byte chunks,
7. carrying forward 128-bit intermediate results
8. with 4 passes per stage
9. with different constants for each message word in all passes
10. to produce 16-byte digest

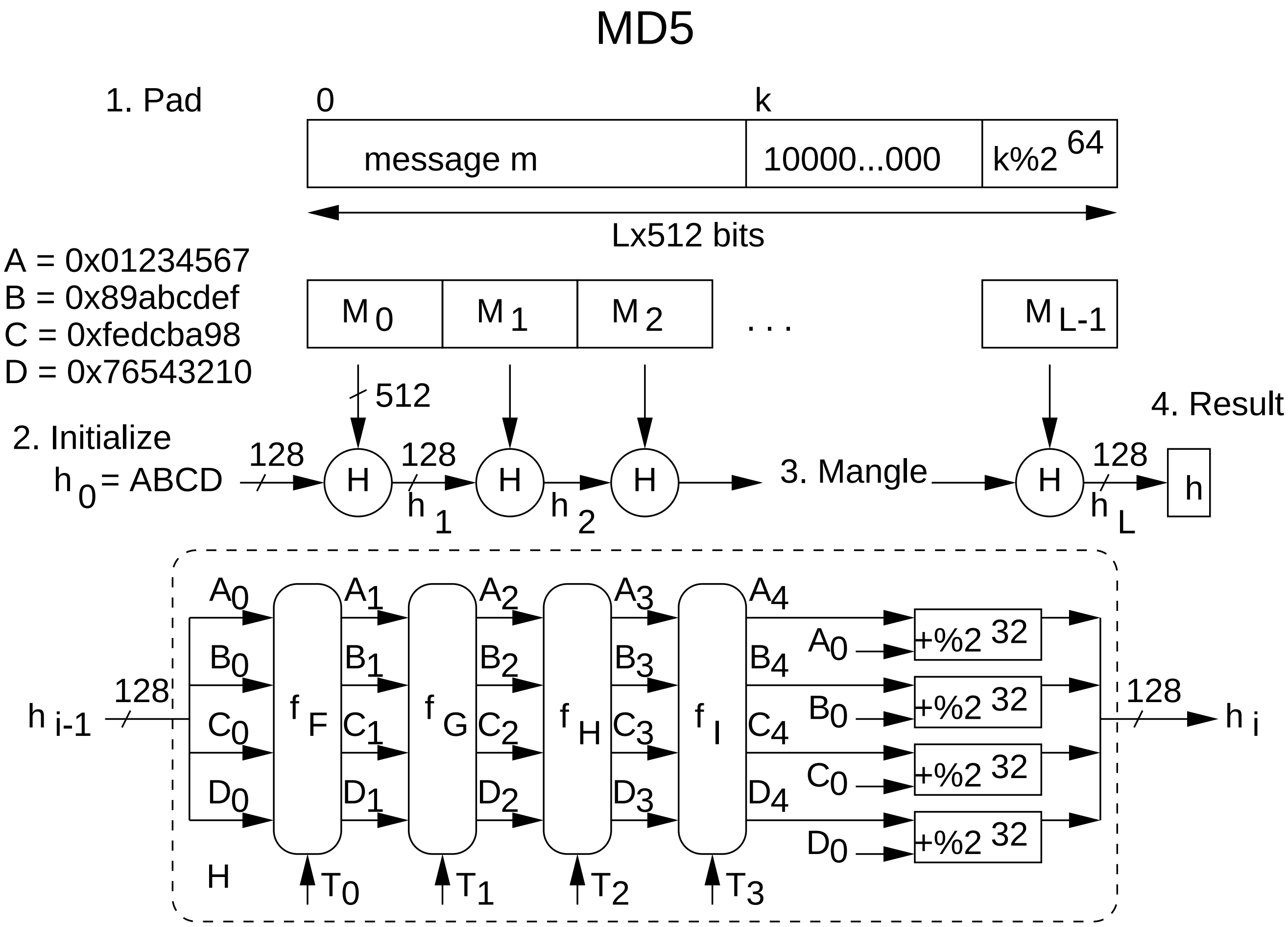

Figure 8: Overall structure of MD5

MD5 (con't.)

F(X,Y,Z) = (X & Y) | (!X & Z)   X-select

G(X,Y,Z) = (X & Z) | (Y & !Z)   Z-select

H(X,Y,Z) = (X + Y + Z)   parity

I(X,Y,Z) = Y + (X | !Z)   ???

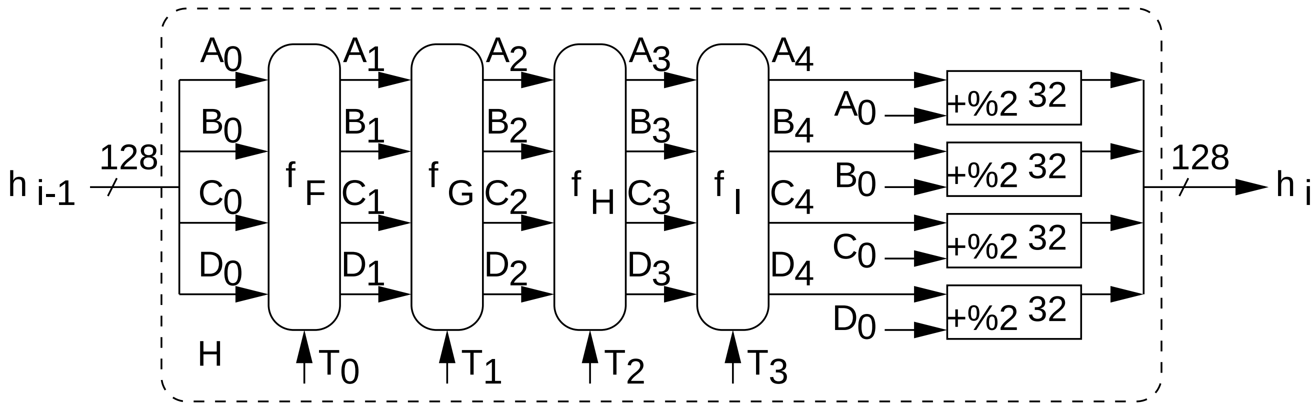

Figure 9: MD5 internals

10.5 SHA

1. 32-bit word computations
2. 160-bit digest
3. (almost) arbitrary bit length input
4. MD4/MD5 padding
5. processed in 16-byte chunks,
6. carrying forward 160-bit intermediate results
7. with 5 passes per stage
8. using complicated mangling function
9. to produce 20-byte digest

10.6 Summary

Hashes will be especially useful for

1. MICs
2. MACs/Digital Signatures

when public key cryptosystems are involved.