

Hashes and Message Digests

A *hash* or *message digest*, is a *one-way function* since it is not practical to reverse.

A function is *cryptographically secure* if it is computationally infeasible to find:

- A message that has a **given message digest**.
- A **different message** with the same message digest.
- **Two messages** that have the same message digest.

➤ Major Algorithms:

- Ron Rivest *Message Digest* MD-family (*MD2*, *MD4* and *MD5*): 128-bit.
- NIST *Secure Hash Algorithm SHA-1*: 160-bit.

They take an *arbitrary-length* string and map it to a *fixed-length* quantity that appears to be *randomly* chosen.

For example, **two inputs** that differ by only **one bit** should have outputs that look like completely independently chosen **random** numbers.

Ideally, the message digest function should be *easy to compute*.

Like secret key algorithms, digest algorithms tends to be computed in *rounds*. The designers finds the smallest number of rounds necessary before the output passes various randomness tests and then add few more to be safe.

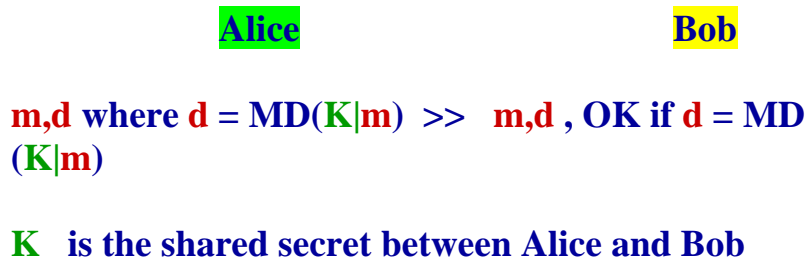
➤ **Things to do with a Hash**

❖ **Authentication:** Alice authenticating Bob:



- r is a random number,
 - MD{K|r} is the message digest of K concatenated with r.
- Alice computes MD{K|r} and if = d, then Bob must know K.

❖ **Computing a MAC:** Using Secret Key K between Alice & Bob



Message Append Attack:

This works for some MD algorithms that have the following property:

if $d=MD(x)$ then for some y , $d'=MD(x/y) = d+MD(y)$

Traudy may intercepts $\langle m,d \rangle$ and replace it with $\langle m',d' \rangle$,

where $m'=m|y$ and $d'=d+MD(y)$.

Bob receives $\langle m',d' \rangle$ and will compute:

$$MD(K|m')=MD(K|m/y)=MD(K|m)+MD(y)=d+MD(y)=d'$$

Thinking that Alice send m' !

How to avoid this flow?

- Compute $MD(m|K)$ instead of $MD(K|m)$.
- Compute $MD(K|m|K)$.
- Compute $MD(K|MD(K|m))$.

❖ Encryption:

Generating one-time pad:

Both Alice and Bob knows the shared secret K and generates:

$$b_1 = MD(K)$$

$$b_i = MD(K|b_{i-1}), i=2,3, \dots$$

Alice

Bob

send $c_i = m_i \oplus b_i$ \gg recv c_i and compute $m_i = c_i \oplus b_i$

❖ Using Secret Key for a Hash:

Unix Password Hash

Unix uses a modified DES to compute the hash of a password.

(to prevent DES hardware from cracking Unix passwords).

- **DES secret Key:**

Pack the 7-bit ASCII associated with each of the *first eight* characters of the password into 56-bit DES key.

- **Salt:**

A 12-bit random number (salt) is stored with the hashed password (to prevent dictionary attack). The salt is used to modify the DES data expansion algorithm.

- **Hashed password:**

The modified DES is used with the secret key to encrypt the **constant 0**. The result is stored with the salt as the user's hashed password.

Example:

```
% ypcat passwd | grep wahab
```

wahab:stg/i.0xxJ1zU:51:13:Dr.
wahab:/home/wahab:/usr/local/bin/tcsh

st is the salt, g/i.0xxJ1zU is the 64 bit encryption of 8 char key

(In base-64 encoding 64 bit block requires $64/6=11$ char).

MD2

It takes a message of arbitrary length and produces 128-bit message digest.

❖ Padding:

The message must be multiple of 16 octets (128-bit).

If the message is already multiple of 16 octets,
16 octets of padding are added.

Otherwise p octets ($1 \leq p \leq 15$) are added.

Each pad octet contains the value n of padding, $1 \leq n \leq 16$.

Note that there must always be padding.

Example:

consider a message m of 10 bytes: "abcdefghij"
the padding length is 6 and the padded message is:

"abcdefghij666666"

❖ Checksum: Fig. 5-4

A 16-byte checksum is appended to the message before computing the MD.

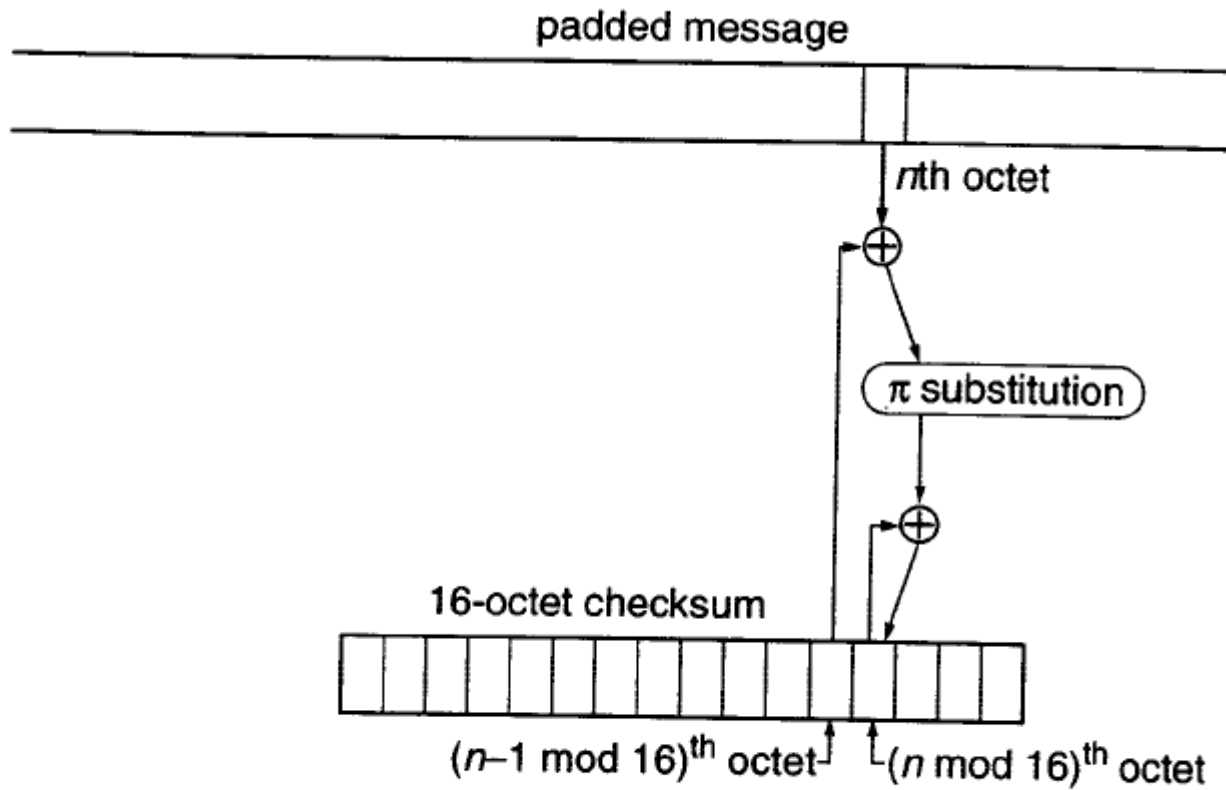


Figure 5-4. MD2 Checksum Calculation

Figure [Fig. 5-5](#) is used for Pi substitution

*Is it the binary representation of pi, one octet at a time? **No!***

41	46	67	201	162	216	124	1	61	54	84	161	236	240	6	19
98	167	5	243	192	199	115	140	152	147	43	217	188	76	130	202
30	155	87	60	253	212	224	22	103	66	111	24	138	23	229	18
190	78	196	214	218	158	222	73	160	251	245	142	187	47	238	122
169	104	121	145	21	178	7	63	148	194	16	137	11	34	95	33
128	127	93	154	90	144	50	39	53	62	204	231	191	247	151	3
255	25	48	179	72	165	181	209	215	94	146	42	172	86	170	198
79	184	56	210	150	164	125	182	118	252	107	226	156	116	4	241
69	157	112	89	100	113	135	32	134	91	207	101	230	45	168	2
27	96	37	173	174	176	185	246	28	70	97	105	52	64	126	15
85	71	163	35	221	81	175	58	195	92	249	206	186	197	234	38
44	83	13	110	133	40	132	9	211	223	205	244	65	129	77	82
106	220	55	200	108	193	171	250	36	225	123	8	12	189	177	74
120	136	149	139	227	99	232	109	233	203	213	254	59	0	29	57
242	239	183	14	102	88	208	228	166	119	114	248	235	117	75	10
49	68	80	180	143	237	31	26	219	153	141	51	159	17	131	20

Figure 5-5. MD2 π Substitution Table

❖ **Final Pass:** [Fig. 5-6](#)

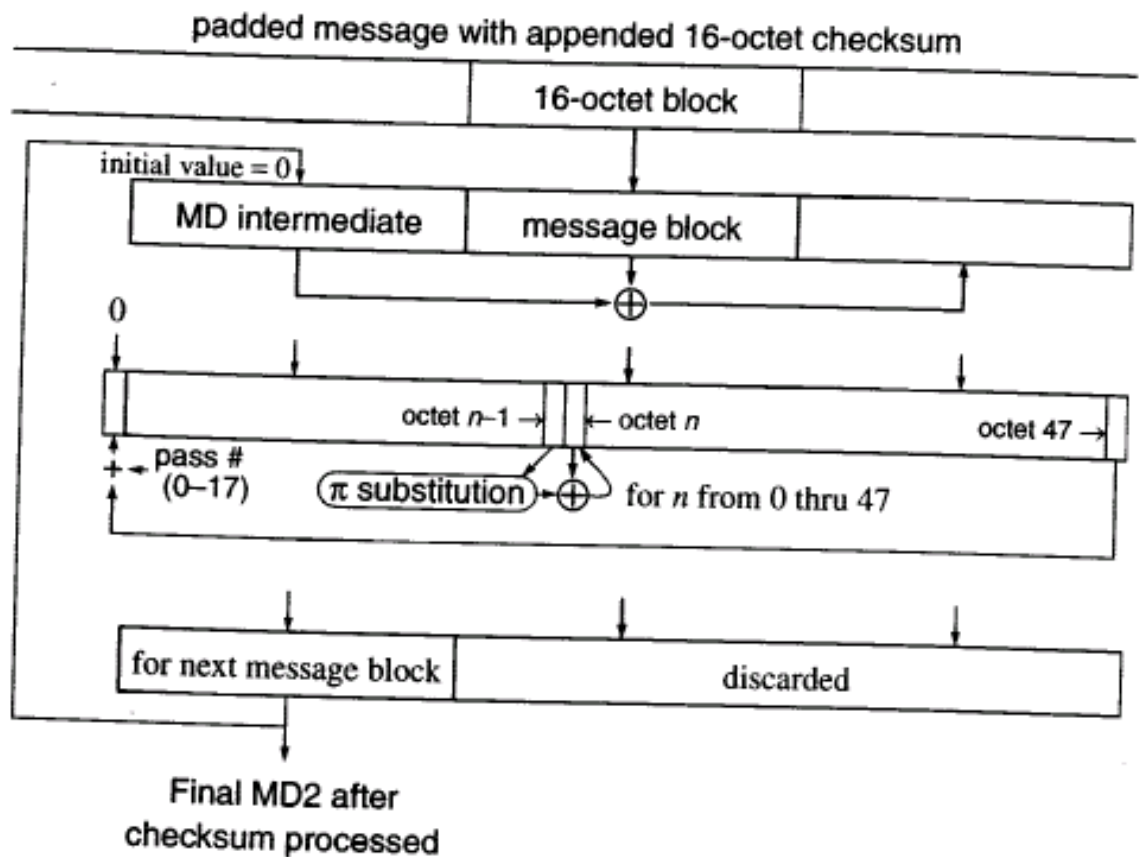


Figure 5-6. MD2 Final Pass

MD4

Was designed to be a 32-bit word oriented so it can be computed faster on 32-bit CPUs than an octet-oriented MD2.

MD5

Was designed to be more concerned with security than speed.

All the MD family produces 128-bit digest.

SHA-1

Designed by NIST to produce 160-bit digests

It is more secure than MD5 but little slower.

HMAC (hash-based MAC) Fig, 5-10 :

HMAC prepends the key to the data, digests it,

and then prepends the key to the result and digests that:

$$\text{MD} (\mathbf{K} | \text{MD} (\mathbf{K} | m))$$

It takes a variable-length *key* and a variable-sized *message* and produces a fixed-size output of the same size as the underlying digest algorithm.

The key is padded with 0s to be 512 bits.

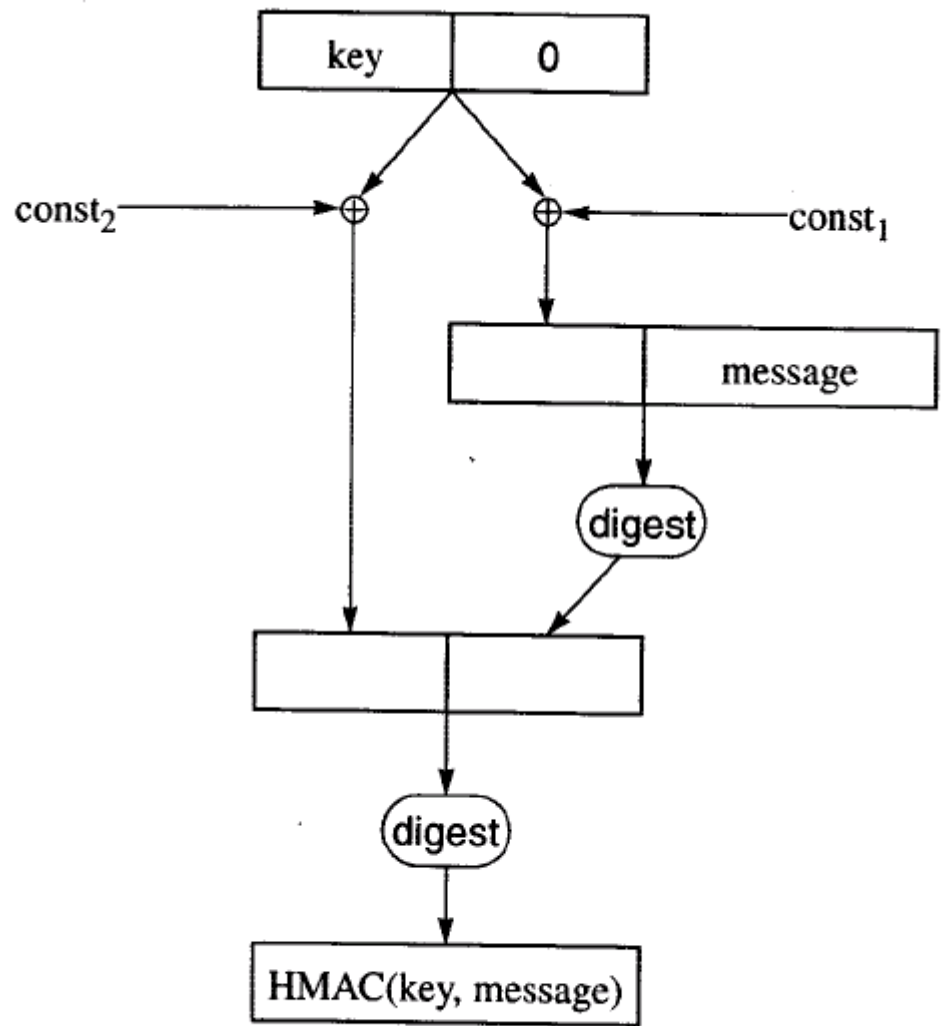


Figure 5-10. HMAC