Chapter 6
Hash Functions

February 15, 2010

6

Hash functions have been used in computing from the earliest days, and have a particular relevance to cryptography - in particular to digital signatures.

Hash functions are also sometimes known as "message digest functions" or "message compression functions".

A hash function takes a long string of data and maps it into a pseudo-random m-bit value. Setting $2^m = N$ we have N possible such values. These can be treated as integer values or as addresses of records in memory or of 1-bit records in a bit map.

6.1Uses of hash functions

1. **Traditional**
   For spreading records (often fixed length) over a file in an "even" fashion.

   Example $H(Person's\ Name\ and\ Address) \longrightarrow Points\ to\ record.$

   This avoids allocation space to letters of alphabet. Obviously we may get a clash, handled as follows:

   Writing: $H(N + A) \ = \ Address\ of\ record$

$$\begin{array}{llll} \rightarrow & Is\ address\ slot\ full? & \overset{}{\underset{No}{\longrightarrow}} & Store\ data\ there \\ \uparrow & & & \\ \uparrow & \quad \downarrow Yes & & \\ \uparrow & & & \\ \uparrow & Increment\ address\ by\ 1 & & \\ \leftarrow & Look\ at\ next\ record\ slot? & & \end{array}$$

Reading: $H(N+A) = Address\ of\ slot$

$$\begin{array}{llll} \rightarrow & Is\ item\ in\ slot\ the\ required\ one? & \overset{}{\underset{Yes}{\longrightarrow}} & Retrieve\ data \\ \uparrow & & & \\ \uparrow & \quad \downarrow No & & \\ \uparrow & & & \\ \uparrow & Increment\ address\ by\ 1 & & \\ \leftarrow & Look\ at\ next\ record\ slot & & \end{array}$$

2. **Footprints**
   To test if some data (example: an RSA public key) has been used before use bit-maps. $H(Data) = Address\ of\ bit$.

   In practice use several distinct $H_1(),\ H_2(),\ \ldots\ H_k()$. If bits at $H_1(Data),\ H_2(Data),\ \ldots\ H_k(Data)$ are all set then assume data has been used before and (depending on the application) discard data, and generate some new data - example: RSA moduli.

   If bits at $H_1(Data),\ H_2(Data),\ \ldots\ H_k(Data)$ not all set place. "Ones" in all the addressed bits and accept data. It can't have been used before.

3. **Unique Digest of Messages**
   $N = H(Message) = Digest$. Here we trust (unless we keep a bit map of hashes) that no "hits" occur. A hit is when $H(m_1) = H(m_2)$ for distinct messages $m_1,\ m_2$. We want $H()$ to be such that:

   (a) Given m_1 and $H(m_1)$ we can't find m_2 such that $H(m_1) = H(m_2)$.
   (b) and cannot find a pair $m_1,\ m_2$ such that $H(m_1) = H(m_2)$.

   (a) Would allow an attacker to alter or replace an existing signed message, and append the signature of m_1 to that of m_2.
   (b) Would allow an attacker to submit an innocent message m_1 for hashing and signature; and then swap m_2 for m_1 and append the signature of m_1 to m_2.

   Clearly a good $H()$ randomises, so that changing a bit in the data changes fifty percent of bits in its hash. i.e. it is similar to an encryption

algorithm.
Question: Could we use an encryption algorithm $E(K, m)$ as a hash?
-With fixed, public K?
-With secret K? - i.e. a MAC

6.2 Probability of hits

This depends on N the number of slots available and t the number of tries (which have filled slots). The expected number of hits per slot $= \frac{t}{N} = \mu$. We treat the number of hits, k per slot as having a Poisson distribution. i.e. $Prob(k\ hits\ per\ slot) = \frac{\mu^k}{k!}e^{-\mu}$. So

$$\begin{aligned} \text{Prob(slot is empty)} &= \mathrm{e}^{-\mu} \sim 1 - \mu + \tfrac{\mu^2}{2}\ldots \\ \text{Prob(slot is not empty)} &= 1\text{-}\mathrm{e}^{-\mu} \sim \mu - \tfrac{\mu^2}{2}\ldots \\ \text{Prob(Two or more hits/slot)} &= 1\text{-}(\mathrm{e}^{-\mu} + \mu e^{-\mu}) \\ &= 1\text{-}(1\text{-}\mu + \tfrac{\mu^2}{2} + \mu - \mu^2) \\ &= (\mu^2)/(2) \end{aligned}$$

For **(1)** *Traditional* use we can use relatively large μ since hits are not a disaster , but merely slow down performance.

For **(2)** *footprints, k hashes into one bit-map.* (example: RSA modulus). So after t objects footprinted we have kt bits set (many several times) $(\mu = \frac{t}{N})$

$$\begin{aligned} \text{Prob(1 bit is set)} &= 1 - \mathrm{e}^{-k\mu} \\ \text{Prob(all k arbitrarily picked k bits are all set)} &= (1\text{-}\mathrm{e}^{-k\mu})^k \\ \text{P=Prob(really new data not giving} & \\ \text{footprint which suggests it's a repeat)} &= 1\text{-}(1\text{-}\mathrm{e}^{-k\mu})^k \end{aligned}$$

Example

$\mu = 0.1$	k	P	$\mu = 0.5$	k	P	$\mu = 0.5$	k	P
Up	1	0.905	*Down*	1	0.607	*Up*	1	0.819
↓	2	0.967	↓	2	0.600	↓	2	0.891
↓	3	0.983	↓	3	0.531	↓	3	0.908
↓	4	0.988	↓	4	0.441	*Down*	4	0.908
↓	5	0.991	↓	5	0.348	↓	5	0.899

For **(3)** *unique hashes for digital signatures*
$Prob(slot\ has\ two\ or\ more\ hits) \sim \frac{\mu^2}{2}$
Expected number of slots with two or more hits$\approx \frac{N.\mu^2}{2}$. For security that there aren't such slots want

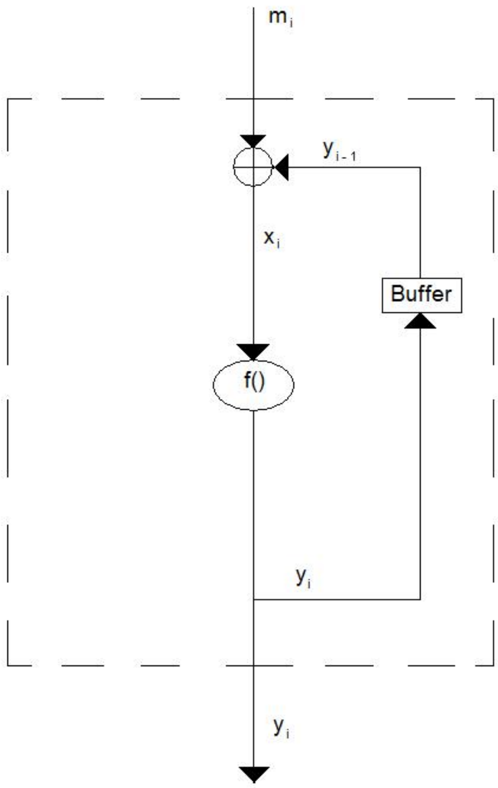
$$\begin{aligned} \text{Expected Number} &<< 1 \\ i.e.\ (\mathrm{N}\mu^2)/(2) &<< 1 \\ i.e.\ \mu &<< (2/\mathrm{N})^{1/2} \\ i.e.\ \mathrm{t} &<< (2\mathrm{N})^{1/2} \end{aligned}$$

(Expected number of tries before a repeat = $\sqrt{(\pi N)/(2)}$ from the birthday paradox). Say $t \sim (N)^{1/2} \sim (2^m)^{1/2} \rightarrow log_2 t \sim \frac{m}{2}$. Suppose $m = 160$ (bits) and $t \sim 2^{80}$ hashes before a hit. 2^{80} is infeasible ($\sim 10^{24}$ ops. One year $= 3 * 10^{16}$ secs)

6.3 The Structure of Hash Functions

CBC-MAC Structure

This is a basic structure:



The function $f()$ could be encryptor $E(k; x_i)$ where k is a key - but for a Hash there is no secrecy and so k is a known constant. We have $y_i = f(m_i \oplus y_{i-1})$. (Clearly we need an initial value (IV), y_0, and the final $Hash(message) = y_n$).

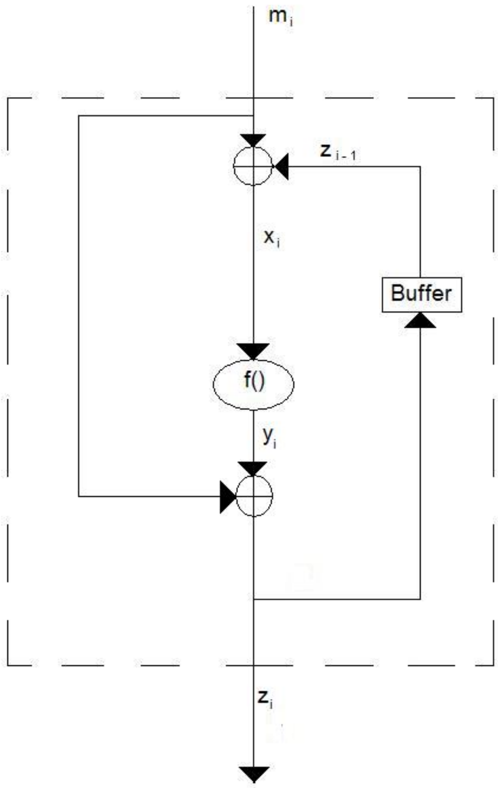
This scheme is not secure. A fraudster can introduce a spurious m'_i, and then choose an m'_{i+1} so that we return to the original $y_{i+1} = f(m_{i+1} + y_i)$ (and the same final resulting Hash from a modified message). Thus

$$\begin{aligned} &\textit{Change} \quad \mathrm{m}_i \text{ to } m'_i \text{ to give } y'_i = f(m'_i + y_{i-1}) \\ &\textit{Change} \quad \mathrm{m}_{i+1} \text{ to } m'_{i+1} \text{ to give } y'_{i+1} = y_{i+1} \\ &\textit{or} \quad \mathrm{m}'_{i+1} + y'_i = m_{i+1} + y_i \\ &\textit{so set} \quad \mathrm{m}'_{i+1} = m_{i+1} - y'_i + f(m_i + y_{i-1}) \end{aligned}$$

(y'_i and y_{i-1} are observed by the fraudster and $f()$ is supposedly known)

RIPE-MAC Structure

An improved structure is as follows:



Output $Z_i = m_i + f(x_i) = m_i + f(m_i + z_{i-1})$
To attack this the fraudster changes m_i to m'_i giving

$$\begin{aligned} \mathrm{y}'_i &= \mathrm{f}(\mathrm{m}'_i + z_{i-1}) \\ \mathrm{z}'_i &= \mathrm{y}'_i + m'_i \\ \textit{He wants } \mathrm{z}'_{i+1} &= \mathrm{z}_{i+1} \\ \textit{or } \mathrm{m}'_{i+1} + f(m'_{i+1} + z'_i) &= \mathrm{m}_{i+1} + f(m_{i+1} + m_i + f(m_i + z_{i-1})) \\ \textit{where } \mathrm{z}_i &= \mathrm{m}_i + f(m_i + z_{i-1}) \end{aligned}$$

This cannot be solved for m'_{i+1} in terms of the other known quantities (including $f()$).

A further possible chosen plain-text attack is like this:

$$\begin{aligned} \textit{The fraudster finds } & \mathrm{H}_1 = H(m_1) \text{ from message } m_1 \\ & \mathrm{H}_2 = H(m_2) \text{ from message } m_2 \\ \textit{and } & \mathrm{H}_3 = H(m_1|m_3) \end{aligned}$$

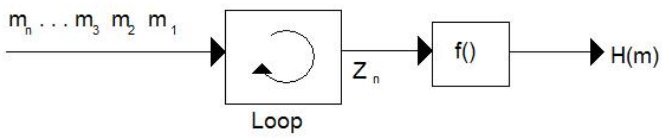
where $(m_1|m_3)$ means m_1 concatenated by m_3, a single feedback item.

Now $H_3 = f(m_3 + H_1) + m_3$. Consider $H_4 = H(m_2|m_4)$ where $(m_2|m_4)$ is m_2 concatenated by a new m_4, a single feed-break item.

$$\begin{aligned} \mathrm{H}_4 &= \mathrm{f}(\mathrm{m}_4 + H_2) + m_4 \\ \textit{Attacker sets}\ \mathrm{m}_4 &= \mathrm{m}_3 + H_1 - H_2 \\ \textit{so}\ \mathrm{H}_4 &= \mathrm{f}(\mathrm{m}_3 + H_1) + m_3 + H_1 - H_2 \\ \textit{or}\ \mathrm{H}_4 &= \mathrm{H}_3 + (H_1 - H_2) \end{aligned}$$

The attacker has formed a message $(m_2|m_4)$ and its Hash H_4 from known $(m_1,\ H_1)$, $(m_2,\ H_2)$ and $(m_3,\ H_3)$ without even knowing $f()$! The attacker has subverted a hash which is secret.

To counter this we add a further $f()$ before the final production of the hash:



Giving $H(m) = f(m_n + f(m_n + z_{n-1}))$. (Or in above notation $H_4 = f(m_4 + f(m_4 + H_2))$ and the attacker cannot find H_4 without knowing $f()$).

6.4 Keyed Hash

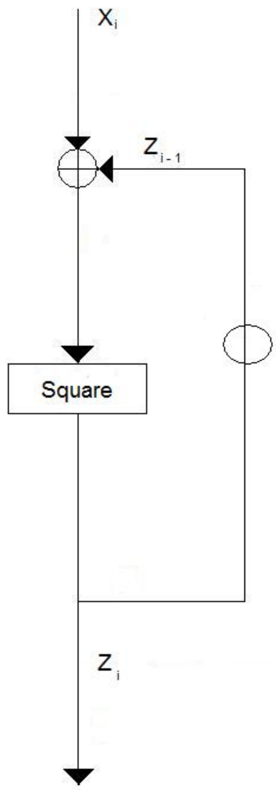
Instead of trying to build Hash functions from encryption MAC/CBC structures we could build MACs (message authentication codes) from Hash functions, by introducing a key to the Hash. For example $MAC(m) = H(key : message : key)$. This has certain weakness, and the recommended Keyed-Hash structure is:

$$
\begin{aligned}
\text{HMAC(K,message)} &= \text{H(K} \oplus opad,\ H(K\ \oplus\ ipad,\ message)) \\
where\ \text{K} &= \text{Key} \\
ipad &= \text{Hex(36)}\ repeated\ B\ times \\
\\
\text{(B} &= \text{length of block processed (in bytes))} \\
\text{(B} &= 64 \rightarrow 512\ bit\ blocks) \\
\\
opad &= \text{Hex(5C)}\ repeated\ \text{B}\ times
\end{aligned}
$$

Obviously any Hash function needs conventions for the message itself:

1. Does it need padding to reach a certain length?
2. Should its real length be included with the message?
3. Should the date be included?
4. Should the originator's ID/Certificate be included? (cf KDSA Chapter 5)

6.5 The Original Hash recommended for Digital Signatures was Square-Mod-n



$Z_i = (Z_{i-1} + X_i)^2 \ mod \ n$

But $X_i = 1111x_{i1L}|1111x_{i1R}|1111x_{i2L}|1111x_{i2R}|\ldots|1111x_{ikL}|1111x_{ikR}|$

i.e. X_i is $2k$ bytes wide but handle k input bytes at a time.
An attacker wants $X_i' = Z_{i-1} + X_i - Z_{i-1}'$ or

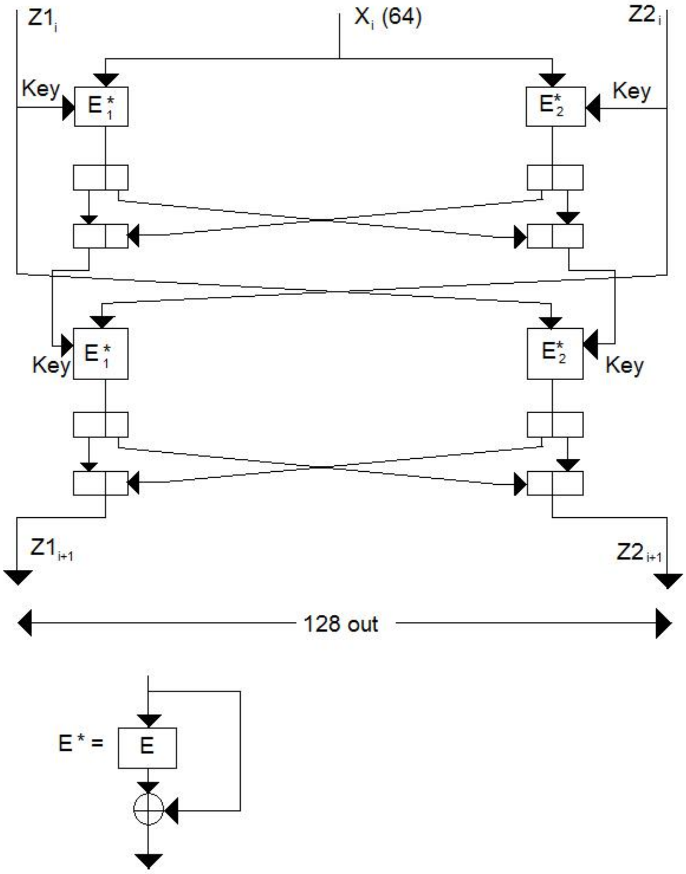
$$\mathrm{X}_i' = (Z_{i-1} - Z_{i-1}') + X_i \qquad . \ (0.1)$$

But the x_i' we put in will be expanded to $X_i' = 1111x_{i1L}'|$ etc. So equation **??** above will only hold if $(Z_{i-1} - Z_{i-1}') = 0000xxxx0000xxxx$ which has probability (2^{-8k}).

However there are clearly problems with all-zero input etc.

MDC is another proposed Hash, using DES encryption $F()$

64-bit in; 128-bit out.



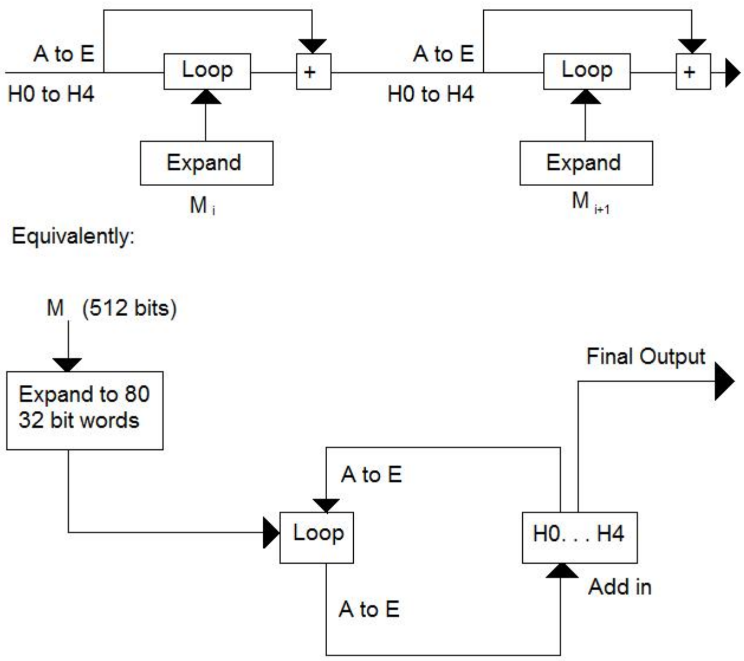
$\mathrm{E}_1^*(x) = E^*(K_1,\ x) \quad \mathrm{K}_1 = K\ but\ set\ first\ 2\ bits\ to\ 0\ 1$
$\mathrm{E}_2^*(x) = E^*(K_2,\ x) \quad \mathrm{K}_2 = K\ but\ set\ first\ 2\ bits\ to\ 1\ 0$

6.6

RIPE-MD is another EU developed Hash function handling 512-bit input blocks and yielding 128-bit output. It was later extended to RIPE-160 to give 160-bit output.

6.7 SHA-1 (Secure Hash Algorithm)

This is the most used hash function in cryptography. 512 input buts are processed at a time. Output is 160 bits. Each round has as inputs $H0,\ H1,\ H2,\ H3,\ H4$ (Five 32-bit words of feedback=hash to date) and 512 bits of message M_i. A round contains a loop, iterated 80 times. The output of the loop is added to the input $H0,\ H1,\ H2,\ H3,\ H4$ to give the new Hash-to-date.



1. Divide 512-bit M_i into sixteen 32-bit words $W(0)$ to $W(15)$
2. Construct $W(16)$ to $W(79)$ from them
3. Set $A = H0,\ B = H1,\ C = H2,\ D = H3,\ E = H4$

4. Loop 80 $t = 0, 79$
   $Temp = S^5(A) + f(t, B, C, D) + E + W(t) + K(t)$
   $E = D,\ D = C,\ C = S^{30}(B),\ B = A,\ A = Temp$

5. $H0 = H0 + A,\ H1 = H1 + B,\ H2 = H2 + C,\ H3 = H3 + D,$
   $H4 = H4 + E$

6. $i = i + 1$ next block of input

Notes on SHA-1

1. "+" = Addition (drop overflow)

2. $S^n(x)$ = Rotate 32-bit X n positions left

3. .

$0 \leq t \leq 19$	f(t, B, C, D)=(B&C) OR ($\overline{B}$&D)
$20 \leq t \leq 39$	f(t, B, C, D)=B XOR C XOR D
$40 \leq t \leq 59$	f(t, B, C, D)=(B&C) OR (B&D) OR (C&D)
$60 \leq t \leq 79$	f(t, B, C, D)=B XOR C XOR D

   (& = *logical AND*, $\overline{B}$=complement of B)

4. .

$0 \leq t \leq 19$	K(t)=5A827999 *hex*
$20 \leq t \leq 39$	K(t)=6ED9EBA1
$40 \leq t \leq 59$	K(t)=8F1BBCDC
$60 \leq t \leq 79$	K(t)=CA62C1D6

5. Initial values for $H()'s$ are:

H0	=	67452301
H1	=	EFCDAB89
H2	=	98BADCFE
H3	=	10325476
H4	=	C3D281F0

6. Expansion procedure for $t = 16$ *to* 79
   $W(t) = S^1(W(t-3)\ XOR\ W(t-8)\ XOR\ W(t-14)\ XOR\ W(t-16))$

7. Padding of input message to produce $n * 512$ *bits*

$$\begin{array}{ccccc} \textit{message} & & \textit{Padding} & & \textit{message length} \\ |101\ldots0011 & | & 100\ldots00 & | & \leftarrow 64\ \textit{bits}\ \rightarrow | \\ \longleftarrow\longleftarrow\longleftarrow & & \text{n}{*}512\ \textit{bits} & & \longrightarrow\longrightarrow\longrightarrow \end{array}$$

The message length field of 64 bits given the bit length of the message ($< 2^{64}$).

6.8 Further Developments

SHA-1, although widely used, has certain weaknesses - probably more theoretical than practical. Proposals for a better hash function have been invited (in the manner of the AES project, see Chapter 2), but as yet no selection of the short-listed or winning candidates has been made.