Hashes and Message Digests

Luca Veltri
(mail.to: luca.veltri@unipr.it)

Corso di Sicurezza nelle reti, a.a. 2009/2010
http://www.tlc.unipr.it/veltri

Hashe Functions

Hash Function

- Also known as Message Digest
- It is a function that takes an input message and produce an output (hash value, or message digest)
- The input can be a variable-length bit string, the output is a fixed-length bit string (e.g. 128 bits)
- It is a one-way function
  - It is not practical to figure out which input corresponds to a given output

\[ h=H(m) \]

- e.g. MD2, MD5 (RFC1321), SHA-1, SHA-1

Hash function properties

- The message digest should look “randomly generated”
- It must be computationally infeasible to find a message with a given prespecified message digest
- It should be impossible to find two messages that has the same digest (although the function is not one-to-one)

How many bits should the output have?

- How many bits should the output have in order to prevent someone from being able to find two messages with the same hash?
- If the message digest has \( m \) bits, then it would take \( 2^{m/2} \) messages chosen at random (Birthday Paradox)
  - However sometime it is not sufficient for an attacker to find out just two messages with the same hash; in such case, a brute-force attack requires \( 2^m \) searches
- That is why message digest functions have output of at least 128 bits (in place of just 64 as for symmetric cryptography)
### About the hash function

- **Message digest function are like alchemy**
  - It’s a bunch of steps that each mangle the message more and more
  - A plausible way of constructing a message digest function is to combine lots of “perverse” operations
  - However, the message digest should remain easy to compute

- **Often, hash function uses constants (magic numbers)**
  - Often the algorithm designers specify how they chose a particular number (to prevent suspects on particular properties of the chosen number)
    - \( \pi \)
    - Published books with random numbers (A book has been published in 1939)

### MD4

- **Designed by Ronald L. Rivest of MIT**
- **Can handle message with an arbitrary number of bits**
- **Produce a 128 bit hash**
  - 32-bit-world-oriented (instead of byte-oriented schemes like MD2)
- **Message padding**
  - The message must be a multiple of 512 bits (16 words);
  - The message is padded by adding one “1” bit and padded with “0”s until bit \( Nx512-64 \)
  - The remaining 64 bit represent the number of unpadded message bits, \( mod \ 2^{64} \)
- **Message processed in 512-bit blocks (16 words)**
- **Each step makes three passes over the message block**
- **Message digest computed on 128-bit quantity (4 words)**

### MD4 scheme

- **Digest**
  - Each pass modifies \( d_0, d_1, d_2, d_3 \)
  - Using \( d_0, d_1, d_2, d_3, m_0, m_1, m_2, m_3, \ldots, m_{15} \)

### MD5

- **Designed to be less concerned with speed and more with security**
- **Very similar to MD4; the main differences are:**
  - 4 passes over each 128-bit (16-byte/4-word) chunk
  - Different functions
  - Uses a different constant \( T \) for each message word for each pass (4 passes \( x \) 16 message words = 64 32-bit constants)
- **The message padding is the same as in MD4**
Hash Functions

MD5 scheme

- Padding
  - The message is always completed with bit of padding in such a way that its length modulo 512 is 448 bits
  - A multiple of 512 minus 64 bits
  - Padding is added from 1 to 512 bits
  - Padding bits are a 1 followed by zeros
- MD5 initialization
  - Message length is converted to 64 bits
- MD5 processing
  - Elaboration of the message in blocks of 512 bits (16 words)
  - Occurs block by block, each block has 4 phases of elaboration
  - In each phase, a different function is used, respectively F, G, H, I
  - Each phase uses as input the buffer ABCD of 128 bits, the current block Y_q of 512 bits, 1/4 of a table of 64 values T[1..64] based on the function sine (tabulated values)
  - The output of the fourth phase is word by word added to the input (modular addition 32)
  - The output of the last calculation is the message digest final

MD5 padding

- Padding
  - Message always completed with padding bits in such a way that its length modulo 512 is 448 bits
  - A multiple of 512 minus 64 bits
  - Padding is added from 1 to 512 bits
  - Padding bits are a 1 followed by zeros

MD5 initialization

- MD5 padding
  - Padding
  - Message length is converted to 64 bits
- MD5 processing
  - Elaboration of the message in blocks of 512 bits (16 words)
  - A partire dal buffer iniziale, per ogni blocco si effettuano 4 fasi di elaborazione
  - In ognuna di esse viene eseguita 1 funzione differente, indicata rispettivamente con F, G, H e I
  - Ciò che è utilizzato in ingresso
  - Il buffer ABCD da 128 bit,
  - Il blocco corrente Y_q da 512 bit,
  - 1/4 di una tabella di 64 valori T[1..64] basati sulla funzione seno (valori tabellati)
  - L'uscita della quarta fase viene sommata word a word con l'ingresso (somma modulo 32)
  - L'uscita dell'ultima elaborazione è il message digest finale
MD5 processing

\[ A = B + (A + g(B, C, D) + X[k] + T[i] \ll s) \]

- Pass 1
  \[ g(x, y, z) = F(x, y, z) = (x \land y) \lor (\neg x \land z) \]

- Pass 2
  \[ G(x, y, z) \]

- Pass 3 ...
  \[ H(x, y, z) \]

- Pass 4 ...
  \[ I(x, y, z) \]

MD5 processing (4 passes)

Secure Hash Standard (SHS/SHA)

- Set of cryptographically secure hash algorithms specified by NIST as message digest functions

- The original specification of the algorithm was published in 1993 as the Secure Hash Standard, FIPS PUB 180, by NIST (SHA-0)

  - Secure Hash Algorithm (SHA)

- Successively revised by the following standards
  - SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512
  - the latter four variants are sometimes collectively referred to as SHA-2
  - SHA-1 (and SHA) produces a message digest that is 160 bits long
  - the other algorithms produce digests that are respectively 224, 256, 384, 512 bits long

- SHA-1 is employed in several widely used security applications and protocols
  - TLS/SSL, PGP, SSH, S/MIME, IPsec, etc.
SHA standards

<table>
<thead>
<tr>
<th>Algorithm variant</th>
<th>Output dimension (bit)</th>
<th>State dimension (bit)</th>
<th>Block dimension (bit)</th>
<th>Max. message dimension (bit)</th>
<th>Word dimension (bit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA-0</td>
<td>160</td>
<td>160</td>
<td>512</td>
<td>$2^{64} - 1$</td>
<td>32</td>
</tr>
<tr>
<td>SHA-1</td>
<td>160</td>
<td>160</td>
<td>512</td>
<td>$2^{64} - 1$</td>
<td>32</td>
</tr>
<tr>
<td>SHA-2</td>
<td>512/256</td>
<td>256</td>
<td>512</td>
<td>$2^{64} - 1$</td>
<td>32</td>
</tr>
<tr>
<td>SHA-256/384/512</td>
<td>512/384</td>
<td>1024</td>
<td>1024</td>
<td>$2^{512} - 1$</td>
<td>64</td>
</tr>
</tbody>
</table>

SHA-1

- SHA-0 was superseded by the revised version SHA-1, published in 1995
  - SHA-1 differs from SHA-0 only by a single bitwise rotation in the message schedule of its compression function
  - This was done, according to the NSA, to correct a flaw in the original algorithm which reduced its cryptographic security
- SHA-1 (as well as SHA-0) produces a 160-bit (5-word blocks) digest from a message with a maximum length of $(2^{64} - 1)$ bits
  - Not a problem, since it would take several hundred years to transmit at 10Gb/s and it would take even longer (hundreds of centuries) to compute SHA-1 at 100MIPS

SHA-1 (cont.)

- Based on principles similar to those used by MD4 and MD5 message digest algorithms Pad the message as in MD4 and MD5 (except that the message is limited to $2^{64}$ bits)
- Operates in stages (as MD4, MD5)
  - Makes 5 passes for each block of data (3 in MD4 and 4 in MD5)
  - Uses a different 160-bit mangle function in each stage
- Little slower than MD5 and (presumably) little more secure

SHA-2

- SHA-224, SHA-256, SHA-384, and SHA-512
- SHA-256 and SHA-512 are computed with 32- and 64-bit words, respectively
  - Use different shift amounts and additive constants
  - Different number of rounds
- SHA-224 and SHA-384 are simply truncated versions of the first two, computed with different initial values
Future of SHA

- SHA-1 has been compromised
- SHA-2 security is not yet as well-established
  - not received as much scrutiny as SHA-1
  - although no attacks have yet been reported, SHA-2 is algorithmically similar to SHA-1
- An open competition for a new SHA-3 function has been started by NIST on November 2, 2007
  - similar to the development process for AES
  - submissions are due October 31, 2008
  - the proclamation of a winner and publication of the new standard are scheduled to take place in 2012

What doing with a Hash

- Encryption
  - encryption should be easy with MD, but what about decryption? ;-) 
  - one-time pad
    - just as OFB, generating a pseudorandom bit stream and encrypting the message just by a simple $\oplus$
    - the pseudorandom stream is generated starting from a MD of a secret: $b_1=MD(K_{AB}\|IV)$, $b_2=MD(K_{AB}\|b_1)$, .. , $b_{k+1}=MD(K_{AB}\|b_k)$
    - same problems as OFB
  - mixing in the plaintext
    - as in OFB, the plaintext is mixed in the bit stream generation
    - $c_1=m_1\oplus b_1$, $c_2=m_2\oplus b_2$, .. , $c_k=m_k\oplus b_k$

- Password Hashing
  - a system may know/store just the hash of a passwd

- Message fingerprint
  - maintaining a copy of a message digest of some data/program in place of the copy of the entire data (for integrity check)

- Digital signature
  - Signing the MD of a message instead of the entire message
    - for efficiency (MDs are easier to compute than public-key algorithms)

- Authentication
  - similar to secret key cryptography

\[
\begin{align*}
&\text{MD}(K_{AB}\|r_A) \\
&\text{MD}(K_{AB}\|r_B)
\end{align*}
\]

Computing a MIC (Message Integrity Check) or MAC

- the obvious thought is that MD(m) is a MIC for m, but it isn’t; anyone can compute MD(m)
- the way is to send also a (shared) secret
  - if the secret is put at the beginning (MD(K,m)), such MD algorithm might become weak since the attacks may continue the MD computation adding a padding
  - however putting the secret at the end might expose the secret
  - a solution could be sending just one half of the hash

msg $\rightarrow$ hash $\rightarrow$ MIC $\rightarrow$ msg

pwd $\rightarrow$ hash $\rightarrow$ MIC $\rightarrow$ result
Using secret key algorithm as Hash Function

- A hash algorithm can be replaced by a block ciphers
  - using $H_0=0$ and zero-pad of final block
  - compute: $H_i = E_{M_i}[H_{i-1}]$
  - and use final block as the hash value
  - similar to CBC but without a key
- resulting hash can be too small (64-bit)
- not very fast to compute

\[
\begin{align*}
m_1 & \rightarrow \text{key} \rightarrow \text{encrypt} \\
m_2 & \rightarrow \text{key} \rightarrow \text{encrypt} \\
\end{align*}
\]

Unix password hashing

- The MD5-crypt function is really not a straight implementation of MD5
  - first the password and salt are MD5 hashed together in a first digest
  - then 1000 iteration loops continuously remix the password, salt and intermediate digest values
  - the output of the last of these rounds is the resulting hash
- A typical output of the stored password together with username, salt, and other information is:

\[
\text{alice:$1$BZftq3sP$x$EeZmr2fGEnKjVAxzj:12747:0:99999:7:::}
\]

- where $1$ indicates the use of MD5-crypt, while BZftq3sP is the base-64 encoding of the salt and xEeZmr2fGEnKjVAxzjQo68 is the password hash

Using secret key algorithm as Hash Function

- Example: the original UNIX password hash (crypt function)
  - first convert the passwd (the message) into a secret key
    - the 7bit ASCII codes of the first 8 chars form the 56bit key
    - the key is used to encrypt the number 0 with a modified DES
      - 25 DES passes are performed
      - the modified DES is used to prevent HW accelerators designed to DES to be used to reverse the passwd hash
      - the modified algorithm uses a 12-bit random number (salt)
    - the salt and the final ciphertext are base64-encoded into a printable string stored in the password or shadow file
- Currently, the most common crypt function used by Unix/Linux systems supports both the original DES-based and hash-based algorithms (e.g. MD5-crypt function), where common hash function such as MD5 or SHA-1 are used
  - such functions generally allow users to have any length password (> 8bytes), and do not limit the password to ASCII (7-bit) text