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.


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:

**Alice**                        **Bob**

**challenge:**  r  >>>>>>>   r

**response:**  d  <<<<<<<<  d = MD{K|r}

- 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 = d, then Bob must know K.

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

**Alice**                        **Bob**

m,d where d = MD(K|m) >> m,d, OK if d = MD (K|m)

K is the shared secret between Alice and 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 \(<m,d>\) and replace it with \(<m',d'>\), where \( m' = m|y \) and \( d' = d + MD(y) \).

**Bob** receives \(<m',d'>\) 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:

\[
b_1 = MD(K)
\]

\[
b_i = MD(K|b_{i-1}), \ i = 2,3, \ ...
\]

Alice \hspace{1cm} Bob
send \( c_i = m_i \circledast b_i \)  \( \Rightarrow \) \( \text{recv } c_i \) and compute \( m_i = c_i \circledast b_i \)

- **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:/g/i.0xxJ1zU:51:13:Dr.
wahab:/home/wahab:/usr/local/bin/tcsh

s1 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 (\( 1 <= p <= 15 \)) are added.

  Each pad octet contains the value \( n \) of padding, \( 1 <= n <= 16 \).

  *Note that there must always be padding.*

- **Example:**

  consider a message \( m \) of 10 bytes: "abcdefhij"
The padding length is 6 and the padded message is:

  "abcdefhij666666"

- **Checksum:** *Fig. 5-4*
A 16-byte checksum is appended to the message before computing the MD.

Figure 5-5 is used for Pi substitution

Is it the binary representation of pi, one octet at a time? **No!**
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>41</th>
<th>46</th>
<th>67</th>
<th>201</th>
<th>162</th>
<th>216</th>
<th>124</th>
<th>1</th>
<th>61</th>
<th>54</th>
<th>84</th>
<th>161</th>
<th>236</th>
<th>240</th>
<th>6</th>
<th>19</th>
</tr>
</thead>
<tbody>
<tr>
<td>98</td>
<td>167</td>
<td>5</td>
<td>243</td>
<td>192</td>
<td>199</td>
<td>115</td>
<td>140</td>
<td>152</td>
<td>147</td>
<td>43</td>
<td>217</td>
<td>188</td>
<td>76</td>
<td>130</td>
<td>202</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>155</td>
<td>87</td>
<td>60</td>
<td>253</td>
<td>212</td>
<td>224</td>
<td>22</td>
<td>103</td>
<td>66</td>
<td>111</td>
<td>24</td>
<td>138</td>
<td>23</td>
<td>229</td>
<td>18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>190</td>
<td>78</td>
<td>196</td>
<td>214</td>
<td>218</td>
<td>158</td>
<td>222</td>
<td>73</td>
<td>160</td>
<td>251</td>
<td>245</td>
<td>142</td>
<td>187</td>
<td>47</td>
<td>238</td>
<td>122</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>169</td>
<td>104</td>
<td>121</td>
<td>145</td>
<td>21</td>
<td>178</td>
<td>7</td>
<td>63</td>
<td>148</td>
<td>194</td>
<td>16</td>
<td>137</td>
<td>11</td>
<td>34</td>
<td>95</td>
<td>33</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>127</td>
<td>93</td>
<td>154</td>
<td>90</td>
<td>144</td>
<td>50</td>
<td>39</td>
<td>53</td>
<td>62</td>
<td>204</td>
<td>231</td>
<td>191</td>
<td>247</td>
<td>151</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>255</td>
<td>25</td>
<td>48</td>
<td>179</td>
<td>72</td>
<td>165</td>
<td>181</td>
<td>209</td>
<td>215</td>
<td>94</td>
<td>146</td>
<td>42</td>
<td>172</td>
<td>86</td>
<td>170</td>
<td>198</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>79</td>
<td>184</td>
<td>56</td>
<td>210</td>
<td>150</td>
<td>164</td>
<td>125</td>
<td>182</td>
<td>118</td>
<td>252</td>
<td>107</td>
<td>226</td>
<td>156</td>
<td>116</td>
<td>4</td>
<td>241</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>69</td>
<td>157</td>
<td>112</td>
<td>89</td>
<td>100</td>
<td>113</td>
<td>135</td>
<td>32</td>
<td>134</td>
<td>91</td>
<td>207</td>
<td>101</td>
<td>230</td>
<td>45</td>
<td>168</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>96</td>
<td>37</td>
<td>173</td>
<td>174</td>
<td>176</td>
<td>185</td>
<td>246</td>
<td>28</td>
<td>70</td>
<td>97</td>
<td>105</td>
<td>52</td>
<td>64</td>
<td>126</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>85</td>
<td>71</td>
<td>163</td>
<td>35</td>
<td>221</td>
<td>81</td>
<td>175</td>
<td>58</td>
<td>195</td>
<td>92</td>
<td>249</td>
<td>206</td>
<td>186</td>
<td>197</td>
<td>234</td>
<td>38</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>83</td>
<td>13</td>
<td>110</td>
<td>133</td>
<td>40</td>
<td>132</td>
<td>9</td>
<td>211</td>
<td>223</td>
<td>205</td>
<td>244</td>
<td>65</td>
<td>129</td>
<td>77</td>
<td>82</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>106</td>
<td>220</td>
<td>55</td>
<td>200</td>
<td>108</td>
<td>193</td>
<td>171</td>
<td>250</td>
<td>36</td>
<td>225</td>
<td>123</td>
<td>8</td>
<td>12</td>
<td>189</td>
<td>177</td>
<td>74</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>136</td>
<td>149</td>
<td>139</td>
<td>227</td>
<td>99</td>
<td>232</td>
<td>109</td>
<td>233</td>
<td>203</td>
<td>213</td>
<td>254</td>
<td>59</td>
<td>0</td>
<td>29</td>
<td>57</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>242</td>
<td>239</td>
<td>183</td>
<td>14</td>
<td>102</td>
<td>88</td>
<td>208</td>
<td>228</td>
<td>166</td>
<td>119</td>
<td>114</td>
<td>248</td>
<td>235</td>
<td>117</td>
<td>75</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>68</td>
<td>80</td>
<td>180</td>
<td>143</td>
<td>237</td>
<td>31</td>
<td>26</td>
<td>219</td>
<td>153</td>
<td>141</td>
<td>51</td>
<td>159</td>
<td>17</td>
<td>131</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 5-5.** MD2 $\pi$ Substitution Table

- **Final Pass:** *Fig. 5-6*
**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 a 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:

\[
\text{MD ( } K | \text{MD ( } K | m \text{ ) )}
\]

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