Ralph (Eddie) Rise

Suk-Hyun Cho

Devin Kaylor

RC4 Encryption

RC4 is an encryption algorithm that was created by Ronald Rivest of RSA Security. It is used in WEP and WPA, which are encryption protocols commonly used on wireless routers. The workings of RC4 used to be a secret, but its code was leaked onto the internet in 1994. RC4 was originally very widely used due to its simplicity and speed. Typically 16 byte keys are used for strong encryption, but shorter key lengths are also widely used due to export restrictions. Over time this code was shown to produce biased outputs towards certain sequences, mostly in first few bytes of the keystream generated. This led to a future version of the RC4 code that is more widely used today, called RC4-drop[n], in which the first *n* bytes of the keystream are dropped in order to get rid of this biased output. Some notable uses of RC4 are implemented in Microsoft Excel, Adobe's Acrobat 2.0 (1994), and BitTorrent clients.

To begin the process of RC4 encryption, you need a key, which is often user-defined and between 40-bits and 256-bits. A 40-bit key represents a five character ASCII code that gets translated into its 40 character binary equivalent (for example, the ASCII key "pwd12" is equivalent to 0111000001110111011001000011000100110010 in binary).

The next part of RC4 is the key-scheduling algorithm (KSA), listed below (from Wikipedia).

```
for i from 0 to 255
    S[i] := i
endfor
j := 0
for i from 0 to 255
    j := (j + S[i] + key[i mod keylength]) mod 256
    swap(S[i],S[j])
endfor
```

KSA creates an array *S* that contains 256 entries with the digits 0 through 255, as in the table below.

0	1	2	...	i	i+1	...	253	254	255

Each of the 256 entries in *S* are then swapped with the *j*-th entry in *S*, which is computed to be

$$j = [(j + S(i) + key[i \bmod keylength]) \bmod 256],$$

where *j* is the previous *j* value (which is initially zero). *S[i]* is the value of the current entry in *S*. *key[i mod keylength]* is either a zero or a one. For example, if we are at the 52th entry in *S* and the keylength was 40-bit, then 52 mod 40 = 12. The 13th element (because numbering for arrays begins at zero) in the binary version of "pwd12" is 0. For example, consider the first iteration of KSA with key "pwd12". Then, since i = 0, i mod 256 = 0. So, the element at the index 0 of the key is p, and its ascii value is 112. So, the new j is computed as

$$j = [(0 + 0 + 112) \bmod 256] = 112.$$

So, swapping the i-th and the j-th elements, we obtain the following array after the first iteration:

112	1	2	...	111	0	113	114	...	255

After the terminus of KSA, we obtain the seemingly random array:

[101, 124, 172, 10, 166, 26, 46, 91, 2, 137, 39, 243, 253, 25, 3, 30, 47, 238, 196, 38, 94, 149, 15, 32, 248, 51, 158, 150, 106, 183, 67, 219, 95, 177, 138, 152, 13, 188, 118, 108, 207, 151, 41, 142, 236, 103, 55, 72, 20, 244, 216, 14, 168, 90, 4, 42, 153, 64, 250, 129, 97, 225, 87, 199, 204, 100, 16, 249,

191, 82, 43, 131, 24, 169, 69, 54, 96, 77, 255, 84, 1, 143, 242, 123, 21, 93, 61, 102, 224, 107, 109, 79, 80, 23, 229, 6, 156, 181, 105, 159, 33, 141, 18, 104, 9, 56, 233, 178, 127, 111, 135, 206, 202, 128, 31, 71, 211, 222, 45, 66, 163, 189, 167, 201, 232, 17, 251, 198, 170, 155, 115, 57, 228, 98, 190, 76, 59, 239, 37, 147, 180, 240, 197, 200, 19, 0, 213, 99, 125, 44, 195, 164, 176, 121, 220, 212

, 86, 186, 34, 214, 230, 254, 40, 203, 194, 231, 162, 226, 187, 116, 208, 22, 68, 88, 192, 140,

205, 234, 119, 83, 136, 63, 12, 112, 217, 154, 184, 81, 70, 35, 174, 78, 241, 179, 210, 215, 49, 144, 130, 48, 133, 7, 209, 92, 73, 193, 28, 75, 117, 223, 50, 113, 114, 148, 173, 29, 53, 160, 8, 139, 246, 65, 252, 161, 221, 185, 27, 36, 11, 110, 237, 165, 5, 182, 145, 171, 120, 157, 134, 175, 122, 58, 235, 52, 62, 126, 85, 60, 132, 74, 245, 227, 218, 89, 247, 146]

The next part of RC4 is the pseudo-random generation algorithm (PRGA). The PRGA is below:

```
i := 0
j := 0
while GeneratingOutput:
    i := (i + 1) mod 256
    j := (j + S[i]) mod 256
    swap(S[i],S[j])
    output S[(S[i] + S[j]) mod 256]
endwhile
```

In PRGA, we begin with the array *S* that was swapped in the KSA. In PRGA, an element in *S* (at index *i*) is swapped with another element in *S* (at index *j*). Then, the next element in the encrypted text is the element of S at the index calculated by (S[i] + S[j] mod 256). At each iteration, *i* is recalculated as (i + 1) mod 256, and *j* is recalculated as (j + S[i]) mod 256. The number of iterations performed is the length of the key, and every value of S is swapped at least once beyond 256 iterations (due to the fact that *i* and *j* are calculated by some number *n* mod 256). The result of this is the code.

Following up with the previous example, let us examine the first iteration of PRGA. Since i, j = 0, i becomes 1 and j becomes (0 + S[1]) mod 256. Since S[1] = 124 (see the resulting S from KSA), j becomes 124. Then, the elements of S at 1 and 124 are swapped, yielding the following new array:

[101, **232**, 172, 10, 166, 26, 46, 91, 2, 137, 39, 243, 253, 25, 3, 30, 47, 238, 196, 38, 94, 149, 15, 32, 248, 51, 158, 150, 106, 183, 67, 219, 95, 177, 138, 152, 13, 188, 118, 108, 207, 151, 41, 142, 236, 103, 55, 72, 20, 244, 216, 14, 168, 90, 4, 42, 153, 64, 250, 129, 97, 225, 87, 199, 204, 100, 16, 249, 191, 82, 43, 131, 24, 169, 69, 54, 96, 77, 255, 84, 1, 143, 242, 123, 21, 93, 61, 102, 224, 107, 109, 79, 80, 23, 229, 6, 156, 181, 105, 159, 33, 141, 18, 104, 9, 56, 233, 178, 127, 111, 135,

206, 202, 128, 31, 71, 211, 222, 45, 66, 163, 189, 167, 201, **124**, 17, 251, 198, 170, 155, 115, 57, 228, 98, 190, 76, 59, 239, 37, 147, 180, 240, 197, 200, 19, 0, 213, 99, 125, 44, 195, 164, 176, 121, 220, 212, 86, 186, 34, 214, 230, 254, 40, 203, 194, 231, 162, 226, 187, 116, 208, 22, 68, 88, 192, 140, 205, 234, 119, 83, 136, 63, 12, 112, 217, 154, 184, 81, 70, 35, 174, 78, 241, 179, 210, 215, 49, 144, 130, 48, 133, 7, 209, 92, 73, 193, 28, 75, 117, 223, 50, 113, 114, 148, 173, 29, 53, 160, 8, 139, 246, 65, 252, 161, 221, 185, 27, 36, 11, 110, 237, 165, 5, 182, 145, 171, 120, 157, 134, 175, 122, 58, 235, 52, 62, 126, 85, 60, 132, 74, 245, 227, 218, 89, 247, 146].

Then, we add the two numbers that we have just swapped: 232+124 = 356, mod by 256 to get 100. then output the 100th element of S, which is 33. We then look at the i-th index of the input string, "Math 310 Proves!", which gives us 'M'. We take the Unicode code of that character, which is 77, and perform a bitwise AND with 33 to get 108. Finally, get the character value for the Unicode code 108, which is 'l', whose HEX representation is then "6C". This is shown in the beginning of the encrypted output string in HEX:

"6CA86FE3CBC33C162595C3E78B9C97BC"

In the case of a wireless router, the key will be required to generate the code. If the key is incorrect, then the encrypted outputs of the codes will not match and the user will not be allowed to connect to the router.