



Using RC4 as a Card Deck Cipher

Benjamin Braatz, sean@heptasean.de



Benjamin Braatz:
Using RC4 as a Card Deck Cipher

Version: 2006-10-12

Subject: Manual Crptography
Keywords: RC4, cryptography, Solitaire

Copyright © Benjamin Braatz

This work is licenced under the Creative Commons Attribution 2.0 Germany Licence. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/2.0/de/> or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.



1. Introduction

In Neal Stephenson's "Cryptonomicon"¹ the Solitaire² cipher (called "Pontifex" in the book) designed by Bruce Schneier is used as a cryptographic algorithm carried out manually with a deck of playing cards. However, this algorithm contained a bias³ discovered by Paul Crowley.

According to a note on the Wikipedia talk page⁴ concerning Solitaire, Crowley considers RC4-52 the best hand cipher. For this reason, I am describing in this document a way to carry out RC4 with a deck of playing cards and a corresponding C implementation of the algorithm.

2. The Algorithm

2.1. The Deck

The Algorithm should be carried out with a normal deck with 52 cards plus two jokers. The usual RC4 algorithm uses 8 bit or 256 numbers. To be able to carry it out with playing cards, we modify this to use only 52 numbers, where we use the following correspondence between playing cards and numbers:

♦ A ≅ 1	♥ A ≅ 14	♠ A ≅ 27	♣ A ≅ 40
♦ 2 ≅ 2	♥ 2 ≅ 15	♠ 2 ≅ 28	♣ 2 ≅ 41
⋮	⋮	⋮	⋮
♦ 10 ≅ 10	♥ 10 ≅ 23	♠ 10 ≅ 36	♣ 10 ≅ 49
♦ J ≅ 11	♥ J ≅ 24	♠ J ≅ 37	♣ J ≅ 50
♦ Q ≅ 12	♥ Q ≅ 25	♠ Q ≅ 38	♣ Q ≅ 51
♦ K ≅ 13	♥ K ≅ 26	♠ K ≅ 39	♣ K ≅ 52

2.2. Initialisation

The plain RC4 algorithm starts with initialising the S-boxes with a key. We do not use this here, because initialisation of the S-boxes can be done by just shuffling the deck.

The C implementation does have no code for shuffling. It requires a string representing the initial status of the deck coded by 104 characters ("D" for diamonds, "H" for hearts, "S" for spades, "C" for clubs, "A" or "1", "2" through "9", "T" for 10, "J", "Q", "K").

The deck (playing the role of the S-boxes) is saved in an array S which contains bytes in the range 1 through 52. The normal RC4 uses 0 through 255, but we start with 1 here to allow the manual procedure to use the usual values of the cards starting with 1.

¹<http://www.cryptonomicon.com/>

²<http://www.schneier.com/solitaire.html>

³<http://www.ciphergoth.org/crypto/solitaire/>

⁴[http://en.wikipedia.org/wiki/Talk:Solitaire_\(cipher\)](http://en.wikipedia.org/wiki/Talk:Solitaire_(cipher))



2.3. Pseudo-Random Generation

Before generating the first pseudo-random number, the jokers (corresponding to the counters i and j in Java) have to be initialised by placing them on top of the deck (with the deck being face-up). This corresponds to $i=0$ and $j=0$ in C.

We need to distinguish the two jokers (Joker A corresponding to counter i and Joker B to counter j). During all card counting in this and the following steps, the jokers are ignored and not counted.

Step 1: Joker A is moved one card down, corresponding to $i = (i + 1) \% 52$ in C. The card below Joker A (in its new position) is converted to a number and Joker B is moved this number of cards down, corresponding to $j = (j + S[i]) \% 52$. If one of the jokers reaches the bottom of the deck it is placed on top again.

Step 2: The cards below the jokers are swapped, implemented by $temp = S[i], S[i] = S[j]$ and $S[j] = temp$ in C.

Step 3: Convert the values of the two cards below the jokers into numbers and add them (modulo 52). Count this number from the top of the deck and convert that card into a number. The resulting number is the generated pseudo-random number. In C this is calculated by $temp = S[(S[i] + S[j] - 1) \% 52]$.

2.4. Encryption and Decryption

The pseudo-random numbers generated in the last subsection are used to encrypt and decrypt a message by shifting along the alphabet, i.e., adding modulo 26, where only uppercase letters "A" through "Z" encoded as 0 through 25 are considered.

3. Example

In this section we will provide an example encryption of the string

HELLO WORLD SOLIT AIRE

by the deck with the initial ordering (from top to bottom)

♠ J	♠ K	♣ 4	♥ 4	♠ 6	♣ 9	♦ 3	♣ J	♦ J	♠ 9	♣ A	♥ A	♥ 7
♣ 6	♠ 2	♦ A	♥ 6	♦ 7	♣ Q	♥ J	♦ 8	♦ 9	♣ 3	♠ Q	♠ 4	♦ K
♣ 7	♦ 5	♦ 2	♣ 8	♣ 5	♥ 8	♥ Q	♣ 2	♥ K	♦ 6	♥ 9	♣ K	♦ 10
♥ 2	♥ 5	♠ A	♦ 4	♥ 10	♠ 10	♦ Q	♣ 10	♠ 5	♠ 8	♠ 7	♥ 3	♠ 3

We start with both jokers (JA and JB) on top:

JA JB ♠ J ...



Now we move Joker A one card down (remember, that jokers do not count), so that it is above the king of spades. Hence, we have to move Joker B 39 cards down. This can quite easily be accomplished by counting 13 cards for every suit before (in this case diamonds and hearts are before spades) and then the value of the card (13 for king in this case). Joker B is inserted below the ten of diamonds and above the two of hearts. The situation after Step 1 is:

♠ J JA ♠ K ... ♠ 10 JB ♥ 2 ...

In Step 2 the cards below the jokers are swapped, so that the king of spades and the two of hearts are exchanged:

♠ J JA ♥ 2 ... ♠ 10 JB ♠ K ...

In Step 3 the output card is computed to be the second card (because of ♥ 2 ≐ 15 below Joker A and ♠ K ≐ 39 below Joker B and (15 + 39) mod 52 = 2), which is the two of hearts, whose value (15) is the output value.

This procedure has to be repeated 19 times in order to obtain shifts for the 19 characters of the plaintext.

Round	Situation after Step 2	Output	
		Card	Number
1	♠ J JA ♥ 2 ... ♠ 10 JB ♠ K ...	♥ 2	15
2	... ♥ 2 JA ♣ 5 ... ♣ 8 JB ♣ 4 ...	♥ K	26
3	... ♣ 5 JA ♠ 5 ... ♣ 10 JB ♥ 4 ...	♥ 4	17
4	... ♠ 5 JA ♠ 5 ... ♣ 7 JB ♠ 6 ...	♥ 9	22
5	... ♠ 5 JA ♠ Q ... ♣ 3 JB ♣ 9 ...	♣ 2	41
6	... ♠ Q JA ♣ 7 ... ♠ K JB ♠ 3 ...	♠ 8	34
7	... ♣ 7 JA ♠ 4 ... ♣ 9 JB ♣ J ...	♠ 6	32
8	... ♠ 4 JA ♠ 6 ... ♥ K JB ♠ J ...	♥ 6	19
9	... ♠ 6 JA ♣ Q ... ♠ 7 JB ♠ 9 ...	♣ 2	41
10	... ♠ Q JB ♣ A ... ♣ Q JA ♣ 7 ...	♣ 2	41
11	... ♣ 7 JA ♠ 8 ... ♥ J JB ♥ A ...	♠ 9	9
12	... ♠ 8 JA ♥ 5 ... ♠ K JB ♥ 7 ...	♣ K	52
13	... ♥ 5 JA ♣ 2 ... ♥ Q JB ♠ 2 ...	♣ 6	45
14	... ♠ 6 JB ♠ 2 ... ♣ 2 JA ♣ Q ...	♠ 3	3
15	... ♣ Q JA ♣ 7 ... ♠ 2 JB ♠ A ...	♣ 10	49
16	... ♣ 7 JA ♣ 8 ... ♠ 2 JB ♥ 6 ...	♣ 2	41
17	... ♣ 8 JA ♥ 9 ... ♠ J JB ♠ 7 ...	♠ 2	2
18	... ♥ 9 JA ♥ J ... ♥ J JB ♠ 9 ...	♣ A	40
19	... ♥ 2 JB ♠ 9 ... ♥ J JA ♣ 5 ...	♠ 3	3

4. Analysis

The algorithm has a key space of $52! \approx 2^{225}$ keys. The states encountered in the pseudo-random generation are determined by the S-boxes (52!) and the counters i and j . This leads to a total of $52! \cdot 52^2 \approx 2^{236}$ states.



We can also compute this by eliminating from the total states of the deck (54!) the irrelevant states where one of the jokers (or both) are at the bottom of the deck (equivalent to the same joker(s) on top of the deck) and Joker B is placed directly above Joker A (equivalent to Joker A directly above Joker B). The first case leads to 53 states where Joker A is at the bottom and Joker B somewhere else, 53 states where Joker B is at the bottom and Joker A somewhere else for each of the 52! orderings of the main cards (without jokers). The second case leads to 52 states for each of the 52! orderings of the main cards (without jokers), because the two jokers can be in 52 different places from top to next to bottom (bottom was already removed for the first case). Summarising we get

$$\begin{aligned} 54! - 2 \cdot 53 \cdot 52! - 52 \cdot 52! &= \\ (54 \cdot 53 - 2 \cdot 53 - 52) \cdot 52! &= \\ 52 \cdot 52 \cdot 52! & \end{aligned}$$

relevant states of the card deck.

Only the Steps 1 and 2 in the pseudo-random generation modify the state of the generator. These two steps are reversible: Step 2 is reversed by just swapping the cards below the jokers back to their original positions. Then, Step 1 can be reversed by moving Joker B up the number of cards indicated by the card below Joker A. Finally Joker A is moved one card up.

Some further analysis still has to be done. The example above already shows far too many occurrences of ♣ 2. Maybe this shows a bias in the algorithm. ...

A. C Implementation

The main part of the C source code (excluding the comments and the licence) is shown below.

```
31 #include <stdlib.h>
#include <stdio.h>

void usage(void) {
    fprintf(stderr,
36         "Usage: rc4-52 -e <deck>\n"
        "         encrypts with the given deck.\n"
        "         rc4-52 -d <deck>\n"
        "         decrypts with the given deck.\n"
        "         Deck is given as a string of 104 characters\n"
41         "         with D for diamonds, H for hearts, S for\n"
        "         spades, C for clubs, A or 1, 2 through 9, T\n"
        "         for ten, J, Q and K.\n"
        "         Encryption and decryption is done from\n"
        "         stdin to stdout.\n");
46     exit(EXIT_FAILURE);
}
```



```
char cardToChar(char suit , char value) {
    char c;
51     if (suit == 'D' || suit == 'd')
        c = 0;
    else if (suit == 'H' || suit == 'h')
        c = 13;
56     else if (suit == 'S' || suit == 's')
        c = 26;
    else if (suit == 'C' || suit == 'c')
        c = 39;
    else return 0;
61     if (value >= '1' && value <= '9')
        c += value - '0';
    else if (value == 'A' || value == 'a')
        c += 1;
    else if (value == 'T' || value == 't')
66     c += 10;
    else if (value == 'J' || value == 'j')
        c += 11;
    else if (value == 'Q' || value == 'q')
        c += 12;
71     else if (value == 'K' || value == 'k')
        c += 13;
    else return 0;
    return c;
}
76 int getAlpha(void) {
    int i;

    while (1) {
81     i = getchar();
        if (i == EOF)
            return 0;
        if (i >= 'A' && i <= 'Z')
            return i;
86     if (i >= 'a' && i <= 'z')
            return i - 'a' + 'A';
    }
}

91 int main(int argc , char **argv) {
    char S[52];
    char temp;
    int i , j , in , mode , counter;
```



```
96  /* process args */
    if (argc != 3) {
        fprintf(stderr, "Wrong number of arguments!\n");
        usage();
    }
101 /* en- or decryption */
    if (argv[1][0] == '-') {
        if (argv[1][1] == 'd') {
            mode = -1;
        } else if (argv[1][1] == 'e') {
106         mode = 1;
        } else {
            fprintf(stderr, "Unknown option!\n");
            usage();
        }
111     if (argv[1][2]) {
        fprintf(stderr, "First argument too long!\n");
        usage();
    }
    } else {
116     fprintf(stderr, "First argument must start with '-'\n");
        usage();
    }
    /* process deck */
    for (counter = 0; counter < 52; counter++) {
121     temp = cardToChar(argv[2][2 * counter],
                        argv[2][2 * counter + 1]);

        if (temp)
            S[counter] = temp;
        else {
126         fprintf(stderr, "Error in specification of card %i!\n",
                    counter + 1);
            usage();
        }
    }
131 /* main loop */
    i = 0; j = 0;
    while ((in = getAlpha()) != 0) {
        i = (i + 1) % 52;
        j = (j + S[i]) % 52;
136     temp = S[i]; S[i] = S[j]; S[j] = temp;
        temp = S[(S[i] + S[j] - 1) % 52];
        printf("%c", ((in - 'A' + mode * temp + 52) % 26) + 'A');
    }
    printf("\n");
141     exit(EXIT_SUCCESS);
}
```