# Contents

# Introduction

Despite its age, RC4 is still one of the most widely used stream ciphers in the world. It

Extract the secret key from given inner state at the end of KSA. Select the best algorithm and implement it.

**TODO** co budu vyuzivat, jak se to v case vyvijelo, cemu se budu venovat, tohle resili nas

# 1. The RC4 stream cipher

The stream cipher RC4 was designed by Ronald L. Rivest for RSA Data Security in 1987 [**?** ]. It was incorporated in RSA's cryptographic library and was first commercially used in Lotus Notes. The RC4 algorithm was a trade secret, but it was reverse-engineered and anonymously published in Cypherpunks mailing list [**?** ] in 1994. The code was accepted to be genuine as its output matched the output produced in software using licensed RC4. To avoid trademark issues, the RC4 cipher is often referred to as ARC4 or ARCFOUR, meaning Alleged RC4.

For its simplicity and speed in software implementation it became very popular. Among implementations in products such as Skype (in modified form), Microsoft Office XP, Lotus Notes or Oracle Secure SQL, it finds application in network protocols such as WEP (Wired Equivalent Privacy), WPA (Wi-Fi Protected Access), SSL (Secure Socket Layer) and formerly in TLS (Transport Layer Security). It has been said that RC4 was the most widely used stream cipher in the world.

**TODO** lepe napsat, snazili se to ohnout nejak spatne, vzali proudovou sifru a pouzili ji spatne — wrong mode, podivat se, jak to bylo v nejakem clanku + citace **KoreK attack** In recent years many attacks have been performed on the cipher, especially on implementations involving wrong use of the initialization vector, such as WEP. In 2015 on the basis of speculations that some state agencies have capabilities to break RC4 used in TLS has IETF published RFC 7565, which prohibited use of RC4 in TLS protocol. Similar recommendation has been issued by Microsoft and Mozilla, but RC4 is still used in many systems, mostly due to legacy reasons or ease of implementation.

RC4 is also commonly implemented in viruses. Its probably thanks to its shortness and the ease of implementation - it can be implemented in few lines of simple to understand code.

## 1.1 Description of the cipher

RC4 is a stream cipher, encryption uses a pseudo-random stream of bits, which are combined with the plaintext using bit-wise exlusive-or (XOR), similarly to Vernam cipher. XOR is the involution, so decryption operates the same way. To generate the pseudo random sequence, RC4 uses secret internal state which consists of:

1. a permutation of $\mathbb{Z}_N = \{0, ..., N-1\}$, where $N = 2^n$
2. two $n$-bit index pointers $i, j$ used to randomize permutation table.

The pseudo-random indice $j$ is secret, but $i$ is public, its value in any stage of the stream is known. Commonly $N = 256$, so $i$ and $j$ are 8-bit and the cipher is byte-oriented. Key length $l$ is defined as the number of bytes in the key and can be in range $1 \leq l \leq N$. Most applications uses the key length between 5 – 16 bytes (corresponding to 40 – 128 bits key).

RC4 consists of two algorithms (given bellow), Key-Scheduling Algorithm (KSA), which initializes the internal permutation involving the secret key, and

Pseudo-Random Generation Algorithm (PRGA), which uses the permutation to generate pseudo-random bytes. The algorithms are described bellow. Any addition related to RC4 will be addition modulo $N$, unless specified otherwise.

**Algorithm 1: KSA**

**Input:** key $K$ of length $l$
**Output:** permutation $S$
**begin**
  *Initialization:*
  $j = 0$
  *Scrambling:*
  **for** $i = 0, \cdots, N-1$ **do**
    $j = j + S[i] + K[i \bmod l]$
    Swap(S[i],S[j])
  **return** $S$

**Algorithm 2: PRGA**

**Input:** permutation $S$
**Output:** one keystream byte
**begin**
  *Initialization*
  $i = 0$ , $j = 0$
  *Keystream generation loop*
  i = i + 1
  j = j + S[i]
  Swap(S[i],S[j])
  t = S[i] + S[j]
  **return** $S[t]$

# 2. Previous attacks

Two broad approaches - KSA, PRGA
- Distinguishing attacks = PRGA-based
- Weak keys - KSA
- IV mode - WEP
- State recovery attacks
- RC4 nomore

# 3. Theoretical analysis of the KSA

**TODO** je to po popisu utoku, takze to chce rict, ktery resim

This thesis is focused to deriving the secret key from the inner state after KSA. The permutation after the KSA algorithm is biased towards the secret key. Class of key retrieval algorithms which uses the inner state can benefit from that fact to extract complete key or information about some of its bytes.

There are two types in biases The most likely value y-th element of permutation after complete KSA is **TODO** In 1995 Roos observed, that 2

$$S_N[y] = \frac{y(y+1)}{2} + K[0...y]$$

This equation has high probability on **TODO** formulace first $\sim 40$ entries is

It is because with hight probability happens event **TODO** event described bellow. Similarly with last $\sim 40$ entries and inverse permutation.

We can also use the fact, that with certain probability values of indices $j_i$ can be found in the inner permutation or the inverse permutation and using the update rule

$$j_{i+1} = j_i + S[i] + K[i \bmod l]$$

.

**TODO** Roos in 1995

Most likely value of , denoted $S_N[y]$, is given by

$$S_N[y] = \frac{y(y+1)}{2} + K[0...y]$$

Hlavni myslenka, nektere klice jsou pravdepodobnejsi nez jine, z toho vnitrniho stavu!!

We will use both facts in the key retrieval algorithm.

## 3.1 Trocha pravdepodobnosti

**Definition 1.** *A discrete probability space is a triple* $(\Omega, \Pr)$ *consisting of:*

1. *A nonempty countably infinite set* $\Omega$.
2. *A function* $\Pr : \Omega \to R$, *called probability measure, satisfying the following properties:*
   (a) $\forall \omega \in \Omega : 0 \leq Pr(\omega) \leq 1$
   (b) $\sum_{\omega \in \Omega} \Pr(\omega) = 1$.

**Definition 2.** *Let* $(\Omega, \Pr)$ *be a discrete probability space. An event* $E$ *is any finite subset of* $\Omega$. *The probability of* $E$ *is defined as* $\Pr(E) = \sum_{\omega \in E} \Pr(\omega)$ *and* $\Pr(\emptyset) = 0$.

**Definition 3.** *Let* $(\Omega, \Pr)$ *be a discrete probability space and* $E_1, E_2 \subseteq \Omega$ *where* $E_2 \neq 0$. *Then we define conditional probability as follows:*

$$Pr(E_1|E_2) = \frac{\Pr(E_1 \cap E_2)}{\Pr(E_2)}$$

**Lemma 1.** ***TODO*** *zakladni vztahy*

$$\Pr(A \cup B) = \Pr(A) + \Pr(B) - \Pr(A \cap B)$$

$$\Pr(\overline{A}) = 1 - \Pr(A)$$

**Definition 4.** *We will write, that co to sakra melo byt*

**Theorem 2.** ***TODO*** *veta o uplne pravdepodobnosti*

**Definition 5.** *Two events* $A$ *and* $B$ *are independent (often written as* $A \perp B$*) if and only if their joint probability equals the product of their probabilities:*

$$\Pr(A \cap B) = \Pr(A) \Pr(B).$$

**TODO** random distribution, uniform at random

## 3.2 Notations and basic assumptions

**Notation.** *Let* $K$ *be the array of key bytes with length* $l$. *We will denote*

$$K[a...b] \coloneqq \sum_{i=a}^{b} K[i \bmod l]$$

*The indice* $j$ *in* $r$*-th round of the KSA will be noted* $j_r$, *the permutation after* $r$ *rounds will be* $S_r$. *Therefore* $S_N$ *is the permutation after complete KSA.*

$S^{-1}$ *will stand for the inverse of the permutation* $S$, *i.e.* $\forall y \in \{0, ..., N-1\}$ : $S[y] = v \Rightarrow S^{-1}[v] = y$.

*Mozna jeste* $K[a] = K[a \bmod l]$

## 3.3 Roos bias

**Lemma 3.** *prerekvizita vety 1 Assume that the index* $j$ *takes its value from* $\mathbb{Z}_N$ *uniformly at random at each round of the KSA. Then* $\forall y \in \mathbb{Z}_N$:

$$\Pr\left(j_{y+1} = \frac{y(y+1)}{2} + K[0...y]\right) \approx \left(\frac{N-1}{N}\right)^{1+\frac{y(y+1)}{2}} + \frac{1}{N}$$

*Proof.* Let $E_y$ denote the event that $j_{y+1} = \sum_{x=0}^{y}(y + K[xmodl])$ □

**Lemma 4.** ***TODO*** *prerekvizita vety 1*

**Theorem 5.** *[? ] Assume that during the KSA the index* $j$ *takes its values uniformly at random from* $\mathbb{Z}_N$. *Then* $\forall 0 \leq i \leq r-1, 1 \leq r \leq N$

$$\Pr(S_r[i] = K[0...i] + \frac{i(i+1)}{2}) \geq (\frac{N-i}{N})(\frac{N-1}{1})^{\frac{i(i+1)}{2}+r} + \frac{1}{N}$$

*Proof.* **TODO** dukaz □

*Corollary.* **TODO** zobecneni na posledni kolo nebo predchozi vetu rovnou smerovat tam?

**TODO** tabulka s aktualnimi hodnotami
**TODO** to same pro InvS

**Theorem 6.** *After the complete KSA,*

$$\Pr(S_N[S_N[y]] = K[0...i] + \frac{i(i+1)}{2}) \approx$$

*clanek 4, appendix, na zacatku graf*

**TODO** tabulka s aktualnimi hodnotami, dukaz

**Notation.**

$$C_y = S_N[y] - \frac{y(y+1)}{2}$$

**TODO** zobecneni na sekvence
**TODO** inverzni sekvence
**TODO** vyyiti tohoto na ziskani klice - rovnice

## 3.4 Substracting equations

Let $i_1 < i_2$. If $C_{i_1} = K[0...i_1]$ and $C_{i_2} = K[0...i_2]$, then we can substract the values and get

$$C_{i_2} - C_{i_1} = K[0...i_2] - K[0...i_1] = K[i_1 + 1...i_2]$$

.

This holds with the product of the individual probabilities of $C_i$

## 3.5 Useful distributions for The Key Recovering Algorithm

**Definition 6.** *If $j_i = S[i]$, we call this as event 1 has occured for index $i$, and denote as $E_1$.*

**Theorem 7.**

$$P(S[i] = j_i) \geq (1 - \frac{1}{N})^i (1 - \frac{i-1}{N})(1 - \frac{1}{N})^{N-i-1} + \frac{1}{N}$$

**Definition 7.** *If $j_i = S[i]$, we call this as event 1' has occured for index $i$, and denote as $E_1'$.*

**Theorem 8.**

$$P(S^{-1}[i] = j_i) \geq (\frac{i}{N})(1 - \frac{1}{N})^{N-1} + \frac{1}{N}$$

*Proof.* $(1 - \frac{1}{N})^i (\frac{i}{N})(1 - \frac{1}{N})^{N-i-1} + \frac{1}{N}$ □

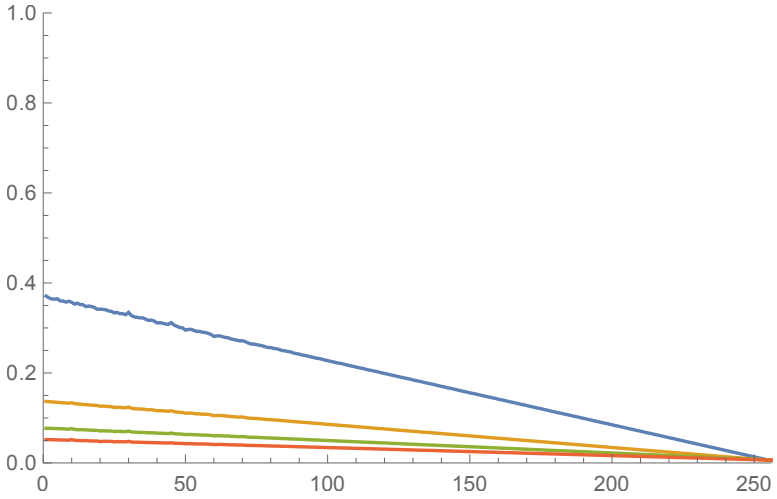


Figure 3.1: hallo

**Theorem 9.** $P(S[i] = j_i \vee S^{-1}[i] = j_i)$ *used for BuildKeyTable algorithm in paper 1*

**TODO** $P(S[S[i]] = j_i)$ - dukazy nikde nejsou...
**TODO** $P(S^{-1}[S^{-1}[i]] = j_i)$
**TODO** $P(S[S[S[i]] = j_i)$
**TODO** experimentalne... rychle to konverguje

# 4. Going back to permutation after KSA

In order to perform key recovery attack, it is needed to get the internal state of the cipher first. A class of attacks focused on obtaining the permutation from the PRGA keystream are State recovery attacks **TODO** dopsat state rec utoky na keystream.

If we have access to the operation memory and know the details of the implementation, it is easy to find RC4 internal permutation candidates. We can use *sliding window* method to go through the memory and search for permutations on $N$ items. We are looking on the windows of $N$ bytes (if $N < 256$), sum of which is $\frac{N(N+1)}{2}$. If the sum is not correct, we will slide the window, i.e. read new byte, discard the last and update the sum by adding new byte and subtracting last byte value. If the sum is correct, we need to test if the set is permutation, i.e. every value is represented exactly once.

The indices $i, j$ are a bit more complicated to determine. Index $i$ is the number of rounds, therefore the number of keystream output bytes. Index $j$ is pseudorandom, we need to either know the location in memory of this value or perform an exhaustive search over all $N$ possible values of $j$.

**TODO** WEP: Uses with IV — form a session key. Easily extracted with the knowledge of the session key and initialization vector.

After finding current state of RC4, it is easily possible to trace back to the initial permutation performing state-reversing loop of *PRGAreverse* algorithm. We will need one of the following:

- The permutation, $i, j$ and the number of rounds $R$.
- The permutation, $i, j$
- The permutation and number of rounds.

**TODO** priklad na deseti prvcich The algoritm *PRGAreverse* describes the last option. If the whole state is known, the algorithm will be trivial, because it will be only necessary to execute the while loop. If only $i, j$ are known, the algorithm will be similar, only the for loop will disappear and the condition for while loop will be $i = j = 0$. On the other side if neither $j$ nor $R$ values are known, we need to exhaustive search all values of $i, j$ pointers. If the condition $i = j = 0$ is fulfilled, it does not necessarily mean, that it is the only permutation candidate for chosen $i, j$. The pointers could be equal zero also during the PRGA, which mean we would need to continue searching for the right state. **TODO** jestli se mi chce, odhadnout pocet kroku

$i_R = R \bmod N$ All subtractions except $r = r - 1$ in Algorithm *PRGAreverse* are modulo $N$.

**Algorithm 3: PRGAreverse**

**Input:** Number of rounds $R$
Internal state $S_R$
$i_R = R \bmod N$
**Output:** Candidates for $S_N$
**begin**
  **for** $j_R \in \{0, ..., N-1\}$ **do**
    $i = i_R$
    $j = R \bmod N$
    $S = S_R$
    $r = R$
    **while** $r > 0$ **do**
      Swap($S[i]$,$S[j]$)
      $j = j - S[i]$
      $i = i - 1$
      $r = r - 1$
    **if** $j = 0$ **then**
      report $S$ as a candidate to $S_N$

# 5. The Key Recovering Algorithm

Tady bych chtel popsat ten, ktery ve finale pouziju. Nebo vsechny?

# Conclusion