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 nasledujici, ten a ten umel to a to, nasledujici clanek to vylepsil, jak, mym cilem je co(sice to udelali, ale nedodali zdrojaky, je to tam vagne popsane a tak)

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. Preliminaries

TODO kapitolu dodelat pozdeji TODO k tomuto nejaky uvod?

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

1. *A nonempty countably infinite set* Ω.
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* Ω. *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

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

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

Theorem 2. *(The law of total probability)*

$$\Pr(E) = \sum_n \Pr(A \cap B_n) = \sum_n \Pr(A \,|\, B_n) \Pr(B_n) \tag{3.1}$$

TODO veta o uplne pravdepodobnosti

Definition 4. TODO Uniform distribution takto nebo musim zavadet jevy jako zobrazeni? $\forall \omega \in \Omega : \Pr(\omega) = \frac{1}{|\Omega|}$

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:*

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

Notation. $\overline{A}$ *is complementary event to* A

4. Theoretical analysis of the KSA

This thesis is /redfocused on deriving/ on derivation of the secret key from the inner state after the KSA. This is possible thanks to the fact, that the permutation after the KSA algorithm is biased towards the secret key. From the aforementioned attacks we focus on a class of key retrieval algorithms which uses this fact to extract complete key or information about some of its bytes.

There are different types in biases in the permutation after the KSA. Later we will use some of them in the key retrieval algorithm to extract the secret key with certain probability.

It was first experimentally observed by Roos in 1995 [**?**] that the most likely value of the inner permutation, denoted S_N, is given by

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

This equation holds with high probability for roughly 40 initial entries. Theoretical analysis was later provided by G. Paul and S. Maitra in [**?**] and subsequently generalized by citace as follows:

$$S_N[i_2] - S_N[i_1] = K[i_1 + 1...i_2] + \frac{i_2(i_2+1)}{2} - \frac{i_1(i_1+1)}{2}$$

Which has high probability also with higher indices and can be used together with tools described bellow to effectively retrieve the secret key.

4.1 Notations and basic assumptions

Notation. *Let K be an array of key bytes of length l. $K[a]$ will always be understood as $K[a \bmod l]$ and we will write*

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

The index j in r-th round of the KSA will be denoted by j_r, S_r denotes the permutation after r-th round of the KSA. Therefore S_N is the permutation after complete KSA. For simplicity, S without index will mean S_N, i.e. the permutation after the complete KSA. S^{-1} will stand for the inverse permutation of S, i.e. $\forall y \in \{0, ..., N-1\} : S[y] = v \Rightarrow S^{-1}[v] = y$.

4.2 Bias towards the sequence sums of key bytes

In the KSA, in every round the index i is incremented by one and the pseudo-random index j is calculated by following rule:

$$j_{i+1} = j_i + S_i[i] + K[i]$$

If we recursively substitute for j, we can write the update rule in the following form:

$$j_{i+1} = \sum_{r=0}^{i} S_r[r] + \sum_{r=0}^{i} K[r].$$

Assuming that $S_i[i] = i$ we obtain

$$j_{i+1} = \sum_{r=0}^{i} r + \sum_{r=0}^{i} K[r] = \frac{i(i+1)}{2} + K[0...i].$$

TODO ODSTRANIT Y ----> I

We will show, that with certain probability $S_N[i] = S_r[i] = j_{i+1}$, therefore

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

.

Despite j being updated using deterministic formula, it is a linear function of a pseudorandom key and therefore itself pseudorandom. Thanks to that we can consider j as being random.

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

$$\Pr\left(j_{i+1} = \frac{i(i+1)}{2} + K[0...i]\right) \approx \left(\frac{N-1}{N}\right)^{1+\frac{i(i+1)}{2}} + \frac{1}{N} \tag{4.1}$$

Proof. Let E_i denote the event that

$$j_{i+1} = \sum_{r=0}^{i} (i + K[r])$$

and A_i denote the event that

$$\forall r \in [0, i] : \ S_r[r] = r$$

.

From the law of total probability 3.1 we have

$$\Pr(E_i) = \Pr(E_i \mid A_i) \Pr(A_i) + \Pr(E_i \mid \overline{A_i}) \Pr(\overline{A_i})$$

Index j is uniformly distributed, hence $\Pr(E_i|\overline{A_i}) \approx \frac{1}{N}$ and from the reasoning above, $\Pr(E_i \mid A_i) = 1$.

We will show by induction that

$$\Pr(A_i) = \left(\frac{N-1}{N}\right)^{\frac{i(i+1)}{2}}.$$

The case $\Pr(A_0 = 1)$ is trivial. For $i \geq 1$

$$
\begin{aligned}
\Pr(A_i) &= \Pr(A_{i-1} \cap (S_i[i] = i)) \\
&= \Pr(A_{i-1}) \cdot \Pr((S_i[i] = i)) \text{ nezavisle, zminit?} \\
&\overset{IH}{=} \left(\frac{N-1}{N}\right)^{\frac{(i-1)i}{2}} \cdot \Pr((S_i[i] = i)) \\
&= \left(\frac{N-1}{N}\right)^{\frac{(i-1)i}{2}} \left(\frac{N-1}{N}\right)^{i} \\
&= \left(\frac{N-1}{N}\right)^{\frac{i(i+1)}{2}}
\end{aligned}
$$

It is worth mentioning that we used the fact that the events A_i and $S_i[i] = i$ are independent. Together we get

$$
\begin{aligned}
\Pr(E_i) &= \Pr(E_i \mid A_i) \Pr(A_i) + \Pr(E_i \mid \overline{A_i}) \Pr(\overline{A_i}) \\
&\approx 1 \cdot \left(\frac{N-1}{N}\right)^{\frac{i(i+1)}{2}} + \frac{1}{N}\left(1 - \left(\frac{N-1}{N}\right)^{\frac{i(i+1)}{2}}\right) \\
&= \left(\frac{N-1}{N}\right)^{\frac{i(i+1)}{2}} + \frac{1}{N} - \frac{1}{N}\left(\frac{N-1}{N}\right)^{\frac{i(i+1)}{2}} \\
&= \left(\frac{N-1}{N}\right)^{\frac{i(i+1)}{2}} \cdot \left(1 - \frac{1}{N}\right) + \frac{1}{N} \\
&= \left(\frac{N-1}{N}\right)^{1+\frac{i(i+1)}{2}} + \frac{1}{N}
\end{aligned}
$$

which completes the proof.

□

TODO kdyz se shoduje s experimenty, tak nakreslit jen rovnici? A proc je tak mimo dole?

Lemma 4. *Assume that the index j takes its value from $\mathbb{Z}_N$ independently and uniformly at random at each round of the KSA. Then, for the permutation after the KSA, $0 \leq i \leq N-1$*

$$
\Pr(S_N[i] = j_{i+1}) \approx \left(\frac{N-i}{N}\right)\left(\frac{N-1}{N}\right)^{N-1} \tag{4.2}
$$

Proof. In round $i+1$ elements of the permutation S on positions i and j_{i+1} are swapped. Therefore $S_{i+1}[i] = S_i[j_{i+1}]$.

Lemma holds under these assumptions:

1. $j_r \neq j_i$ for $r \in \{0, ..., i-1\}$, i.e. $S[j_i]$ is not swapped until the i-th iteration.
2. $j_i \geq i$, i.e. $S[i]$ is swapped with item with greater index.
3. $j_r \neq i$ for $r \in \{i+1, ..., N-1\}$, i.e. $S[i]$ is involved in remaining rounds.

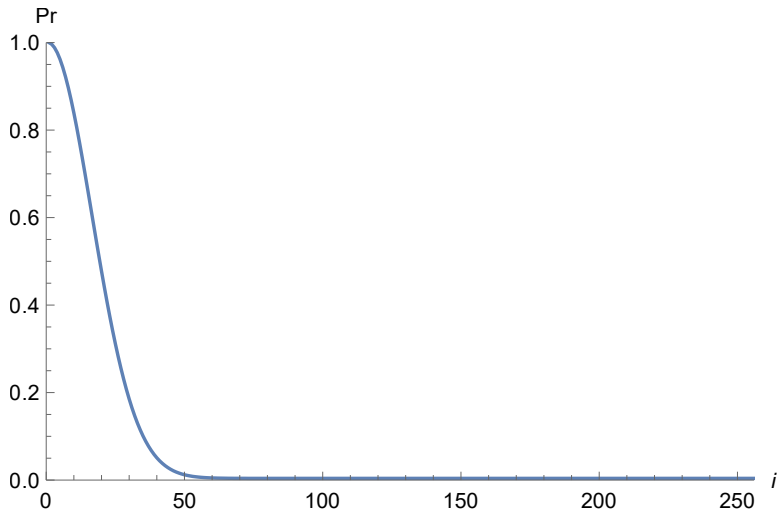

Figure 4.1: $\Pr\left(j_{i+1} = \frac{i(i+1)}{2} + K[0...i]\right)$

$S_i[j_{i+1}] = j_{i+1}$ if this element is not involved in any of the previous rounds. This happens if and only if $j_{i+1} \notin \{0, ..., i-1\} \cup \{j_1, ..., j_i\}$.

$\Pr(j_{i+1} \notin \{0, ..., i-1\}) = \left(\frac{N-i}{N}\right)$ and $\Pr(j_{i+1} \notin \{j_0, ..., j_i\}) = \left(\frac{N-1}{N}\right)^i$, hence

$$\Pr(S_{i+1}[i] = j_{i+1}) = \left(\frac{N-i}{N}\right)\left(\frac{N-1}{N}\right)^i.$$

If in next $N-1-i$ rounds the index i is not touched by any of the subsequent j indices, i.e. $\forall r \in \{i+1, ..., N\}\ j_r \neq i$, the value will stay on place, thus $S_N[j_{i+1}] = j_{i+1}$. This holds with probability $(\frac{N-1}{N})^{N-1-i}$. Together we get

$$\Pr(S_N[i] = j_{i+1}) = \left(\frac{N-i}{N}\right)\left(\frac{N-1}{N}\right)^i\left(\frac{N-1}{N}\right)^{N-1-i}.$$

□

Experimental data TODO popisky v mathematice

1. $S_r[r] = r$ for $r \in \{0, ..., i\}$, i.e. the value on index r was not swapped before the r-th iteration. lemma 1
2. $S_i[j_{i+1}] = j_{i+1}$, which happens if the index j_{i+1} was not involved in previous rounds as in lemma 2
3. $j_r \neq i$ for $r \in \{i+1, ..., N-1\}$, i.e. $S[i]$ is involved in any of the remaining rounds. as in lemma 2

Theorem 5. *[?] Assume that during the KSA the index j takes its values uniformly at random from $\mathbb{Z}_N$. Then $\forall i \in \mathbb{Z}_{Nd}$*

$$\Pr(S_N[i] = K[0...i] + \frac{i(i+1)}{2}) \approx \left(\frac{N-i}{N}\right)\left(\frac{N-1}{1}\right)^{\frac{i(i+1)}{2}+N} + \frac{1}{N} \tag{4.3}$$

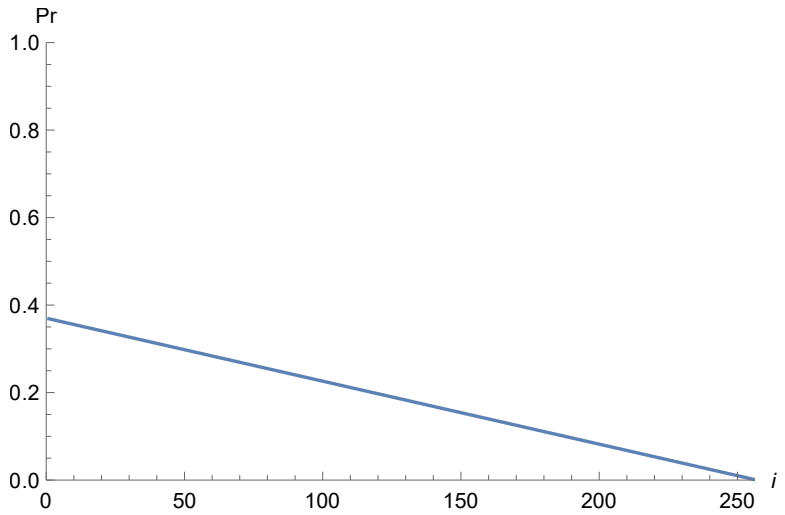

Figure 4.2: $\Pr(S_N[i] = j_{i+1})$

Proof. The underlying assumptions are:

Let A_i denote event that $\forall r \in \{0, ..., i\} S_r[r] = r$ as in Lemma odkaz.

One way to obtain the result is by combining Lemma 1 and Lemma 2 together. We have

$$S_N[i] = S_{i+1} = j_{i+1} = \sum_{r=0}^{i} (i + K[r])$$

with probability co zbytek lemmatu 1? Proc se nepocita najednou s random association?

$$\begin{aligned}\Pr(A_i) \cdot \Pr(S_N[i] = j_{i+1}) &= \left(\frac{N-1}{N}\right)^{\frac{i(i+1)}{2}} \left(\frac{N-i}{N}\right)\left(\frac{N-1}{N}\right)^{N-1} \\ &= \left(\frac{N-i}{N}\right) \cdot \left(\frac{N-1}{N}\right)^{\frac{i(i+1)}{2}+N-1}\end{aligned}$$

Another possibility is that the event above will not happen and yet the equation holds due to random association. This

$$\left(1 - \left(\tfrac{N-i}{N}\right) \cdot \left(\tfrac{N-1}{N}\right)^{\frac{i(i+1)}{2}+N-1}\right) \cdot \frac{1}{N}$$

By combining these we obtain the result as follows:

$$\begin{aligned}&\left(\frac{N-i}{N}\right) \cdot \left(\frac{N-1}{N}\right)^{\frac{i(i+1)}{2}+N-1} + \left(1 - \left(\frac{N-i}{N}\right) \cdot \left(\frac{N-1}{N}\right)^{\frac{i(i+1)}{2}+N-1}\right) \cdot \frac{1}{N} \\ &= \left(1 - \frac{1}{N}\right) \cdot \left(\frac{N-i}{N}\right) \cdot \left(\frac{N-1}{N}\right)^{\frac{i(i+1)}{2}+N-1} + \frac{1}{N} \\ &= \left(\frac{N-i}{N}\right) \cdot \left(\frac{N-1}{N}\right)^{\frac{i(i+1)}{2}+N} + \frac{1}{N}\end{aligned}$$

□

Example. ahpj

TODO tabulka s aktualnimi hodnotami

TODO to same pro InvS

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Pr	.371	.368	.364	.358	.351	.343	.334	.324	.313	.301	.288	.275	.262	.248	.234	.220
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Pr	.206	.192	.179	.165	.153	.14	.129	.117	.107	.097	.087	.079	.071	.063	.056	.050
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
Pr	.045	.039	.035	.031	.027	.024	.021	.019	.016	.015	.013	.011	.01	.009	.008	.008

Table 4.1: The probabilities of 4.3

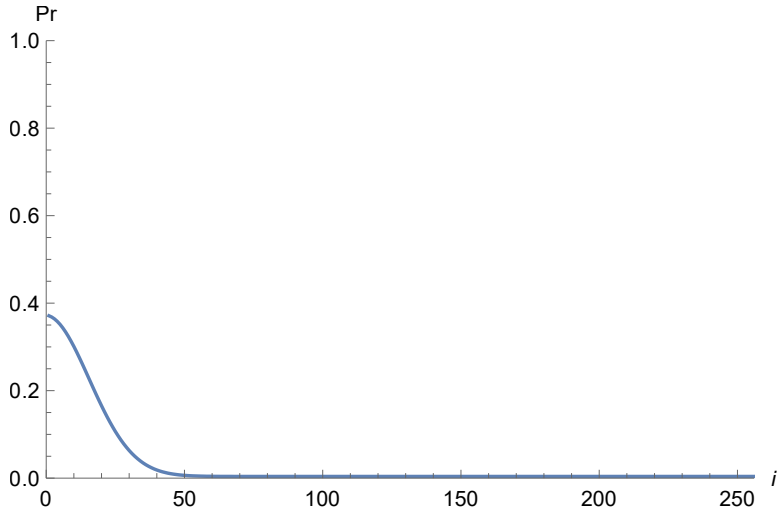

Figure 4.3:

Theorem 6. *After the complete KSA,*

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

clanek 4, appendix, na zacatku graf

TODO tabulka s aktualnimi hodnotami, dukaz

Notation.

$$C_i = S_N[i] - \frac{i(i+1)}{2}$$

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

4.3 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

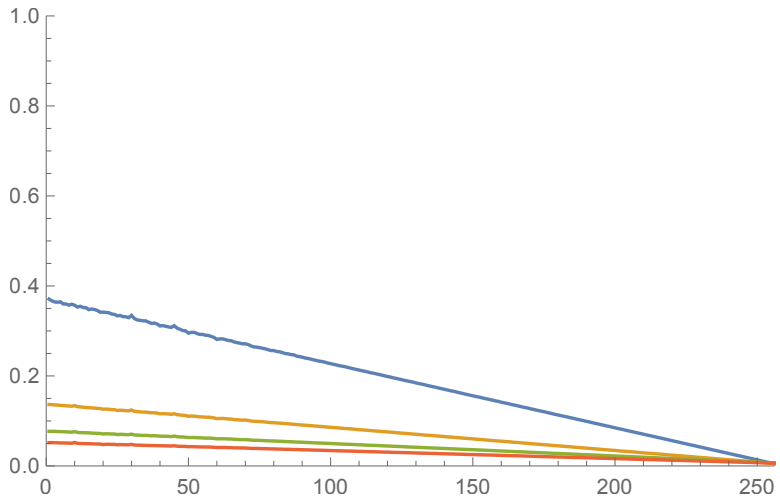

Figure 4.4: hallo

4.4 Useful distributions for The Key Recovering Algorithm

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

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

.

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

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
TODO experimentalne vsechno dohromady je cca 0.6

We will use both facts in the key retrieval algorithm.

5. 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

6. The Key Recovering Algorithm

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

TODO Some keys are more probable than others, we will sort them according to weight and try first nc number of candidates? Dat to tam?

Conclusion