Cryptography

Lab no. 2 – till 26 III 2017

[You can get max 10 points for this list]
RC4 encryption scheme uses two algorithms KSA(N, T) which takes a secret key K as an input, and outputs an array (permutation) S of size N. Algorithm PRGA(N) outputs pseudo-random bytes from S.

Algorithm 1: $\mathsf{KSA}_k(\mathsf{N,T})$ – $k[i]$ returns ith BYTE of the key. L is the length of the key in bytes.

```
1 for i from 0 to N − 1 do
2 |   S[i] := i
3 end
4 j := 0;
5 for i from 0 to T do
6 |   j := (j + S[i] + k[i mod L]) mod N;
7 |   swap(S[i mod N], S[j mod N]);
8 end
```

Algorithm 2: $\mathsf{PRGA}_S(\mathsf{N})$

```
i := 0;
j := 0;
while GeneratingOutput do
|   i := (i + 1) mod N;
|   j := (j + S[i]) mod N;
|   swap(S[i], S[j]);
|   K := S[(S[i] + S[j]) mod N];
|   outputK
end
```

Algorithm 3: $\mathsf{KSA\text{-}RS}_k(\mathsf{N,T})$ – $k[i]$ returns ith BIT of key k. L denotes length of the key in bits.

```
for i from 0 to N − 1 do
|   S[i] := i
end
for r from 0 to T do
|   Top = array();
|   Bottom = array();;
|   for i from 0 to N do
|   |   if key[rN + i mod L] == 0 then
|   |   |   Top.push(i)
|   |   else
|   |   |   Bottom.push(i)
|   |   end
|   end
|   foreach Top as i ⇒ v do
|   |   newS[i] := S[v]
|   end
|   foreach Bottom as i ⇒ v do
|   |   newS[Top.size + i] := S[v]
|   end
|   S := newS;
end
```

Original RC4 = RC4(N, T) = RC4(256,256) is:

1. $S := \mathsf{KSA}_k(\mathsf{N, N})$
2. $outputStream \leftarrow \mathsf{PRGA}_S(\mathsf{N})$

RC4-RS(N, T) is:

1. $S := \mathsf{KSA\text{-}RS}_k(\mathsf{N,T})$
2. $outputStream \leftarrow \mathsf{PRGA}_S(\mathsf{N})$

Function RC4-drop[D] drops first D bytes of PRGA output.
Function RC4-SST repeats the loop of KSA (lines 5-8 as long as SST marking is done, see: `https://eprint.iacr.org/2016/1049.pdf` – it is StoppingRuleKLZ from page 15).

Assignment 1 (10 pts.) – security track Implement above algorithms and test the quality of generated random bits depending on the parameters:

1. RC4(16, 16)
2. RC4(16, 16)-drop[48]
3. RC4(16, 64)

Repeat experiments for different key lengths: $8, 16, 24, 32, 40$ and 64 bits.

For statistical tests use any of: TestU01, DieHard, Dieharder.

Assignment 2 (10 pts.) – algorithmic track Implement above algorithms and test the quality of generated random bits depending on the parameters:

1. RC4(16, 16)
2. RC4-RS(16, 64)
3. RC4-RS(16, 92)
4. RC4-SST(16)

Repeat experiments for different key lengths: $8, 16, 24, 32, 40$ and 64 bits.

For statistical tests use any of: TestU01, DieHard, Dieharder.

Assignment 3 (10 pts.) – algorithmic track $RandomWalker(N, d, l)$ is defined for the following parameters:

N - number of vertices of the directed (multi)graph $V = \{0, 1, \ldots, N-1\}$ where $N = 2^n$,

d - the out-degree of each vertex,

l - the number of steps a pseudo-random walk performs between times when it announces where it is.

Let S_j denote a permutation of N elements (for $j = 0, \ldots, d-1$). Then the set of (directed-, multi-) edges is defined as:

$$E = \{(i, S_j(i)) : i = 0, \ldots, N-1; j = 0, \ldots, d-1\}$$

The random walk starts at $v_0 = 0$ and performs l steps by walking at step k from a vertex v_k to the vertex $v_{k+1} = S_{k \bmod d}(v_k)$. The output of the generator is a sequence of n-bit numbers: $v_l, v_{2l}, v_{3l}, \ldots$.

For $n = 4, 6, 8 (N = 16, 64, 256)$, $d = n, 2n$ and $l = n, 2n, 3n$ run statistical tests (TestU01 or Diehard or Dieharder) of the generated output.

Initialize $RandomWalker(N, d, l)$ with d instances of $RC4 - SST(N)$.

Consider the following extensions:

- at each step the permutation is changed (you may use $PRGA_{S_j}(N)$ from RC4),
- think about using an additional (d+1) instance of RC4 which would be used to decide which edge to choose *i.e.,* if the kth byte of the $d+1$'s instance is equal to b_k then the walk goes from v_k to $v_{k+1} = S_{b_k}(v_k)$.