Introduction to Cryptology

3.2 - Stream Ciphers

Federico Pintore

Mathematical Institute, University of Oxford (UK)

Michaelmas term 2020

Stream Ciphers

Terminology is not standard: either used to refer to a practical instantiation of PRGs or to the encryption scheme which uses it.

Each stream cipher gives rise to a family of PRGs (one for each expansion factor $\ell(n)$).

Stream Ciphers

A stream cipher consists of two deterministic algorithms:

- $\mathrm{st}_0 \leftarrow \mathrm{Init}(s, IV)$: on input a seed s and an optional initialisation vector IV, it outputs an initial state st_0.
- $(y, \mathrm{st}_{i+1}) \leftarrow \mathrm{GetBits}(\mathrm{st}_i)$: it takes the i-th state st_i and outputs a bit y and an updated state, i.e. st_{i+1}.

Stream Ciphers and PRGs

Construction of a PRG $G_{\ell(n)}$:

$\mathrm{st}_0 \leftarrow \mathrm{Init}(s, IV)$

for $i = 1, \cdots, \ell(n)$;

$\quad (y_i, \mathrm{st}_i) \leftarrow \mathrm{GetBits}(\mathrm{st}_{i-1})$

return $y_1, \cdots, y_{\ell(n)}$

Stream Ciphers and PRGs

Construction of a PRG $G_{\ell(n)}$:

$\mathrm{st}_0 \leftarrow \mathrm{Init}(s, IV)$
for $i = 1, \cdots, \ell(n)$;
  $(y_i, \mathrm{st}_i) \leftarrow \mathrm{GetBits}(\mathrm{st}_{i-1})$
return $y_1, \cdots, y_{\ell(n)}$

A stream cipher is secure if:

- it takes no IV,
- for any expansion factor $\ell(n)$, $G_{\ell(n)}$ is a PRG.

Examples of Stream Ciphers

- Linear-Feedback Shift Registers (LFSRs)
- RC4, proposed by Ron Rivest in 1987 (it should no longer be used)
- eStream competition:
  - Salsa20 (and ChaCha)
  - SOSEMANUK

RC4 - Efficiency/security trade-off

RC4 became extremely popular for SSL/TLS connections:

- immune to some attacks;
- it is pretty fast.

RC4 - Efficiency/security trade-off

RC4 became extremely popular for SSL/TLS connections:

- immune to some attacks;
- it is pretty fast.

Security-wise, it has been less successfull:

- its outputs have several biases (e.g. the probability that the second byte is 0 is 1/128 instead of 1/256);
- these biases can be used to recover a message when encrypted several times with different keys (AlFardan *et al.*, 2013).

Salsa20

Salsa20 works with 4-byte words to expand

- a 32-byte key $k = (k_1, k_2, \ldots, k_8)$ and
- a 8-byte nonce[1] $IV = (IV_1, IV_2)$

into a 2^{70}-byte string (the stream).

[1] A random string, shared with the receiver.

Salsa20

Salsa20 works with 4-byte words to expand

- a 32-byte key $k = (k_1, k_2, \ldots, k_8)$ and
- a 8-byte nonce[1] $IV = (IV_1, IV_2)$

into a 2^{70}-byte string (the stream).

The stream is generated in 64-byte blocks:

- each block is derived from the key, the nonce, and a 8-byte block number $\mathrm{bc} = (\mathrm{bc}_1, \mathrm{bc}_2)$;
- therefore and any number of blocks can be computed in parallel.

[1] A random string, shared with the receiver.

Salsa20 - Notation

- The sum of two 4-byte words is denoted by $+$.
- The xor of two 4-bytes words is denoted by $\oplus$.
- The operator $<<< t$ rotates of t positions to the left the bits of a 4-byte word.
- The quarterround operator takes four 4-byte words (y_1, y_2, y_3, y_4) and returns four 4-byte words (z_1, z_2, z_3, z_4):

$$
\begin{aligned}
z_1 &= y_1 \oplus ((y_0 + y_3) <<< 7) \\
z_2 &= y_2 \oplus ((z_1 + y_0) <<< 9) \\
z_3 &= y_3 \oplus ((z_2 + z_1) <<< 13) \\
z_4 &= y_0 \oplus ((z_3 + z_2) <<< 18).
\end{aligned}
$$

Salsa20

The block indexed by $\mathrm{bc} = (\mathrm{bc}_1, \mathrm{bc}_2)$ takes the matrix

$$A = \begin{pmatrix} \lambda_1 & k_1 & k_2 & k_3 \\ k_4 & \lambda_2 & IV_1 & IV_2 \\ \mathrm{bc}_1 & \mathrm{bc}_1 & \lambda_3 & k_5 \\ k_6 & k_7 & k_8 & \lambda_4 \end{pmatrix}$$

as input ($\lambda_1, \lambda_2, \lambda_3, \lambda_4$ are constant words). It returns the 4x4 matrix B obtained as follows:

$B = A$

for $i = 1, \cdots, 20$;

  for $j = 1, \cdots, 4$;

    $(B_{1,j}, B_{2,j}, B_{3,j}, B_{4,j}) \leftarrow \mathrm{quarterround}(B_{1,j}, B_{2,j}, B_{3,j}, B_{4,j})$

  $B \leftarrow \mathrm{Transpose}(B)$

$B \leftarrow A + \mathrm{Transpose}(B)$

return B

Salsa20

To encrypt a b-byte plaintext m, the first b bytes of the 2^{70}-byte stream are xor'ed with m (the rest of the stream is discarded).

The decryption of a b-byte ciphertext c consists in xor'ing c with the first b bytes of the 2^{70}-byte stream.

Further Reading I

Nadhem J AlFardan, Daniel J Bernstein, Kenneth G Paterson, Bertram Poettering, and Jacob CN Schuldt.
On the security of RC4 in TLS.
In 22nd USENIX Security Symposium (USENIX Security 13), pages 305–320, 2013.

Boaz Barak and Shai Halevi.
A model and architecture for pseudo-random generation with applications to/dev/random.
In Proceedings of the 12th ACM conference on Computer and communications security, pages 203–212. ACM, 2005.

Daniel J Bernstein.
The Salsa20 Family of Stream Ciphers.
In New stream cipher designs, pages 84–97. Springer, 2008.

Further Reading II


Lenore Blum, Manuel Blum, and Mike Shub.
A simple unpredictable pseudo-random number generator.
SIAM Journal on computing, 15(2):364–383, 1986.


Christian Cachin.
Entropy measures and unconditional security in cryptography.
PhD thesis, ETH Zurich, 1997.


Scott Fluhrer, Itsik Mantin, and Adi Shamir.
Weaknesses in the key scheduling algorithm of RC4.
In Selected areas in cryptography, pages 1–24. Springer, 2001.

Further Reading III

Christina Garman, Kenneth G Paterson, and Thyla Van der Merwe.
Attacks only get better: Password recovery attacks against RC4 in TLS.
In 24th USENIX Security Symposium (USENIX Security 15), pages 113–128, 2015.

Itsik Mantin and Adi Shamir.
A practical attack on broadcast RC4.
In Fast Software Encryption, pages 152–164. Springer, 2002.