

BDD-Attacks on Stream Ciphers

Dirk Stegemann

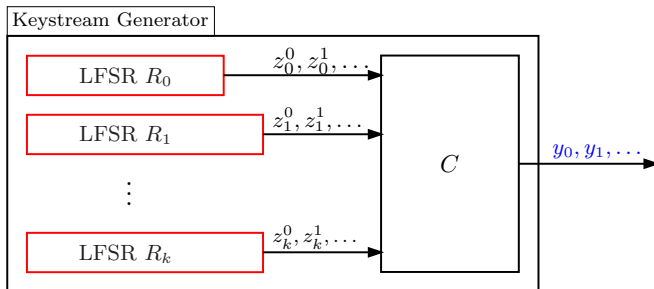
Theoretical Computer Science
University of Mannheim, Germany

GI-Dagstuhl Seminar on Symmetric Cryptography
January 08-12, 2007
Schloss Dagstuhl, Germany

Overview

- 1 Introduction
- 2 Binary Decision Diagrams
- 3 BDD-based Attacks and Improvements
- 4 Conclusion

LFSR-based Keystream Generators

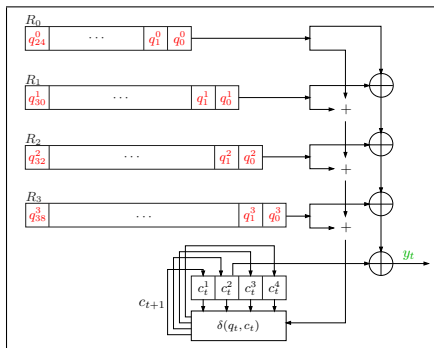


secret key = initial states of the LFSRs

Practical Examples:

- Bluetooth keystream generator E_0
- GSM keystream generator A5/1

The E_0 Keystream Generator

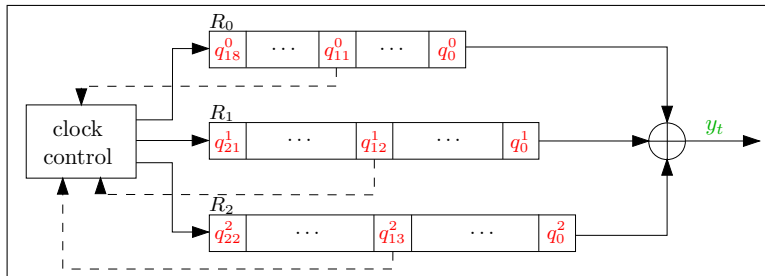


In each iteration t :

- Produce keystream bit $y_t = q_t^0 \oplus \dots \oplus q_t^3 \oplus c_t^2$
- Update memory bits $c_{t+1} = \delta(q_t, c_t)$
- Clock all 4 LFSRs

The A5/1 Keystream Generator

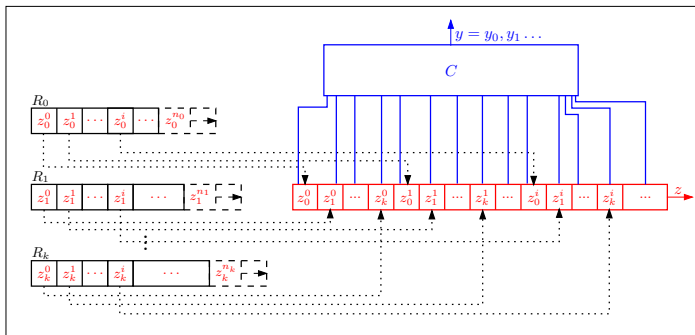
LFSRs R_0 , R_1 , R_2 with control positions N_0 , N_1 , N_2



In each iteration t :

- Produce keystream bit $y_t = q_t^0 \oplus q_t^1 \oplus q_t^2$
- Clock R_j iff $q_{N_j+t} = \text{maj}_3(q_{N_0+t}^0, q_{N_1+t}^1, q_{N_2+t}^2)$

Generic Representation



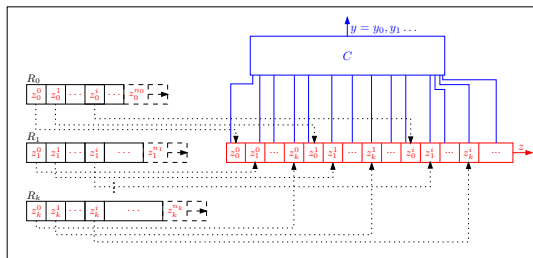
- LFSRs produce **internal bitstream z**
- *Compression function C* computes keystream $y = C(z)$

Cryptanalysis

Problem: Given a prefix of the **keystream** y , compute the **initial states** of the LFSRs.

Observation

The **initial states** are contained in the first bits of the **internal bitstream** z .

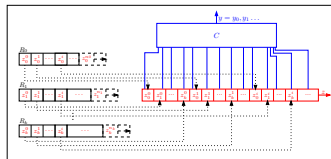


→ Reconstruct the **internal bitstream** z from y .

Reconstruction of z

Restrictions on z

- (1) LFSR-relations hold for the z_i .
- (2) $C(z)$ is a prefix of y .



Generic Algorithm

Compute for $m = 1, 2, \dots, m^*$ the candidate sets

$$\begin{aligned}
 \mathcal{P}_m &= \{z \in \{0, 1\}^m \mid z \text{ fulfills (1) and (2)}\} \\
 &= \mathcal{P}_{m-1} \times \{0, 1\} \\
 &\quad \cap \{(z_0, \dots, z_{m-1}) \in \{0, 1\}^m \mid z_{m-1} \text{ fulfills (1)}\} \\
 &\quad \cap \{z \in \{0, 1\}^m \mid z \text{ fulfills (2)}\}
 \end{aligned}$$

until only bitstreams fully consistent with y are left.

Reconstruction of z

Problem: How to maintain the $\mathcal{P}_m \subseteq \{0, 1\}^m$ efficiently?

Represent $S \subseteq \{0, 1\}^m$ by its characteristic Boolean function

$$\begin{aligned} \delta_S : \{0, 1\}^m &\rightarrow \{0, 1\} \\ z &\mapsto \begin{cases} 1 & z \in S \\ 0 & z \notin S \end{cases} \end{aligned}$$

and find an efficient data structure for δ_S .

→ *Binary Decision Diagrams (BDDs)*

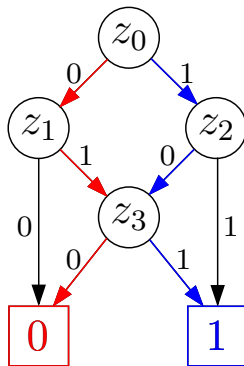
Binary Decision Diagram (BDD)

directed acyclic graph G_f

- 1 source and 2 sinks
- inner nodes have $outdeg = 2$
- node labels
 - sinks: 0 and 1
 - inner nodes: variables from $Z_n = \{z_0, \dots, z_{n-1}\}$
- edge labels from $\in \{0, 1\}$
- represents Boolean function

$$f : \{0, 1\}^m \rightarrow \{0, 1\}$$

$z \mapsto$ end of the z -path



Examples:

$z = (z_0, z_1, z_2, z_3)$	$f(z)$
$(0, 1, 1, 0)$	0
$(1, 0, 0, 1)$	1

Efficiency of BDD-operations

For Boolean functions f, g, h and corresponding BDDs G_f, G_g and G_h , we need to compute

- $G_f \wedge G_g \wedge G_h$ computing $f \wedge g \wedge h$,
- the set of satisfying inputs $G_f^{-1}(1)$ of G_f

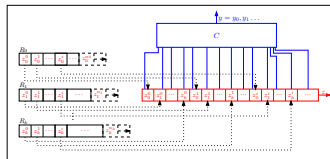
If each variable is read at most once on each path in the BDD, we have:

operation	runtime and memory
$G_f \wedge G_g \wedge G_h$	$O(G_f \cdot G_g \cdot G_h)$
$G_f^{-1}(1)$	$O(G_f + n \cdot G_f^{-1}(1))$

Reconstruction of z with BDDs

Restrictions on z

- (1) LFSR-relations hold for the z_i .
- (2) $C(z)$ is a prefix of y .



BDD-Algorithm

Compute for $m = 1, 2, \dots, m^*$ the **BDDs**

$$P_m = P_{m-1} \wedge \underbrace{\{z \in \{0, 1\}^m \mid z_{m-1} \text{ fulfills (1)}\}}_{BDD_m^{(1)}} \\ \wedge \underbrace{\{z \in \{0, 1\}^m \mid z \text{ fulfills (2)}\}}_{BDD_m^{(2)}}$$

until only bitstreams fully consistent with y are left.

Reconstruction of z with BDDs

Runtime of the algorithm depends on

- number of iterations m^* and $\max_m \{|P_m|\}$
- parameters of the keystream generator
 - information rate $\alpha \in (0, 1]$
 - (best case) compression rate $\gamma \in (0, 1]$

Theorem (Krause 2002)

For keystream generators of keylength n and $|BDD_m^{(2)}| \in m^{O(1)}$,

- $m^* = \lceil \alpha^{-1} n \rceil$
- $|P_m| \leq n^{O(1)} 2^{\frac{1-\alpha}{1+\alpha} n}$ for $m \leq m^*$

→ z and therefore x can be reconstructed in time and with space

$$n^{O(1)} 2^{\frac{1-\alpha}{1+\alpha} n}$$

from the first $\lceil \gamma \alpha^{-1} n \rceil$ consecutive bits of y .

Application to E_0 and A5/1

	keylength n	α	γ	# required keystream bits
E_0	128	0.25	0.25	n
A5/1	64	0.2193	0.25	$1.14n$

Performance of the BDD-attack:

	theory		simulations
	time and space	q	time and space
E_0	$2^{76.8}$	0.85	$2^{65.28}$
A5/1	2^{41}	0.9	$2^{36.9}$

Simulations on reduced instances indicate that

- practical BDD-sizes only deviate from theory by a constant 2^q
- memory is the main bottleneck

Reducing the Memory Requirements

One approach: Divide-and-Conquer strategies (DCS) [KS 2006]

- 1 Divide the key space into segments $t_i \in T$
- 2 BDD-attack the segments individually

Recover the secret key in

- time $\leq n^{O(1)} \cdot |T| \cdot 2^{w^*(n)}$
- memory $n^{O(1)} 2^{w^*(n)}$

for a generator-specific parameter $w^*(n)$

In order to gain from the DCS, we need

- $|T|$ not too large
- $w^* < \frac{1-\alpha}{1+\alpha} n$

Example: Guess the shortest LFSR

Guessing the shortest LFSR of E_0 and A5/1

Performance of the modified BDD-attack

	theory			simulations	
	time	space	q	time	space
E_0	$2^{76.8}$	$2^{76.8}$	0.85	$2^{65.28}$	$2^{65.28}$
E_0^{DCS}	$2^{76.5}$	$2^{51.5}$	0.95	$2^{72.68}$	$2^{48.93}$
A5/1	2^{41}	2^{41}	0.9	$2^{36.9}$	$2^{36.9}$
A5/1 ^{DCS}	$2^{51.86}$	$2^{32.85}$	0.77	$2^{39.93}$	$2^{25.3}$

Observations

- time requirements slightly increase
- memory requirements significantly decrease

What can be done with standard hardware

Our simulation environment:

- Standard Linux PC
- 2.7 GHz Intel Xeon Processor
- 4 GB of main memory
- BDD library CUDD (written in C)

	max. keylength	runtime
E_0	37	87 minutes
E_0^{DCS}	46	60.5 hours
A5/1	30	54 minutes
A5/1 ^{DCS}	37	25.1 hours

→ No practical danger for the real-life versions.

Conclusion

BDD-Attack on Stream Ciphers

- generic, short keystream attack
- works also for irregularly clocked combiners
- relies on BDDs for storing intermediate results
- needs a large amount of memory
- memory consumption can be reduced by divide-and-conquer strategies

Experimental results indicate that

- practical time and memory requirements do not seem to deviate from theory by more than a constant
- memory is the main bottle neck

Further Research

- Attack other ciphers
 - Self Shrinking Generator [Krause 2002], [KS 2006]
 - Bit Search Generator and variants [Gouget et al. 2005]
- Use other divide-and-conquer strategies
 - first half of each register [KS 2006]
 - two entire registers [Shaked, Wool 2006]
- Implement better memory management
- Use other BDD-subclasses
 - Multi-Terminal BDDs [Stegemann 2004]

The End.

`dstegema@th.informatik.uni-mannheim.de`