Extending SAT Solvers to Cryptographic Problems

Mate Soos, Karsten Nohl, Claude Castelluccia

INRIA Rhône-Alpes, University of Virginia

July 1, 2009

Table of Contents

Outline

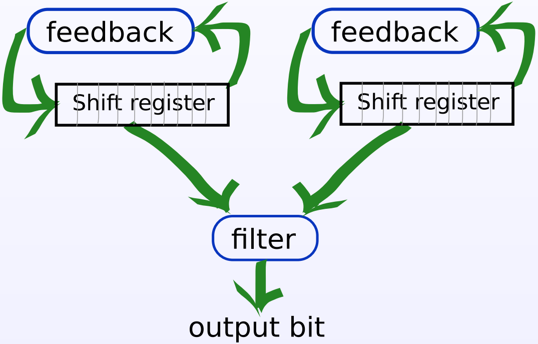
1. Background
   - DPLL-based SAT solvers
   - Stream ciphers
2. Adapting the SAT solver
   - XOR-support
   - Gaussian elimination
   - Dynamic behaviour analysis
3. Adapting the cipher representation
   - Logical circuit representation
   - Representation of non-linear functions
4. Implemented attacks
   - Crypto-1 and HiTag2
   - Bivium

DPLL-based SAT solvers

- A tool to solve a problem given in clauses ('and' of 'or'-s)
- Performs unit propagation
- Picks a variable to branch on, works on the two sub-problems
- Optimisations:
  - learning
  - non-chronological backjumping
  - restarting
  - variable choice
  - implementation details
- We used MiniSat2

Stream ciphers

- Uses a set of *shift registers*
- Shift registers' *feedback function* is either linear or non-linear
- Uses a *filter function* to generate 1 secret bit from the state
- Working: clock-filter-clock-filter. . . = feedback-filter-feedback-filter. . .



Outline

Problem with XOR-s

The truth

$$a \oplus b \oplus c$$

must be put into the solver as

$$a \vee \overline{b} \vee \overline{c} \qquad (1) \qquad\qquad \overline{a} \vee \overline{b} \vee c \qquad (2)$$
$$a \vee b \vee c \qquad (3) \qquad\qquad \overline{a} \vee b \vee \overline{c} \qquad (4)$$

So, it takes 2^{n-1} clauses to model an n-long XOR

Problem with XOR-s

To model the truth

$$x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5 \oplus x_6 \oplus x_7 \oplus x_8$$

the following truths are put into the SAT solver (*cutting*)

$$\overline{y_1} \oplus x_1 \oplus x_2 \oplus x_3 \oplus x_4$$
$$\overline{y_2} \oplus x_5 \oplus x_6 \oplus x_7 \oplus x_8 \qquad y_1 \oplus y_2$$

Problems: still too long, extra vars

Solution to XOR-s

Xor-clauses [Massacci00Taming]:

$$a \oplus b \oplus c$$

represents all the regular clauses

$$a \vee \overline{b} \vee \overline{c} \qquad (1) \qquad\qquad \overline{a} \vee \overline{b} \vee c \qquad (2)$$
$$a \vee b \vee c \qquad (3) \qquad\qquad \overline{a} \vee b \vee \overline{c} \qquad (4)$$

and changes appearance to match the regular clause that is the most pertinent to the situation. Gives this changed appearance to the `analyze()` method

Uses a *watched variable* scheme instead of a watched literal scheme

Gain:

- 2.2x in speed
- order of magnitude in memory

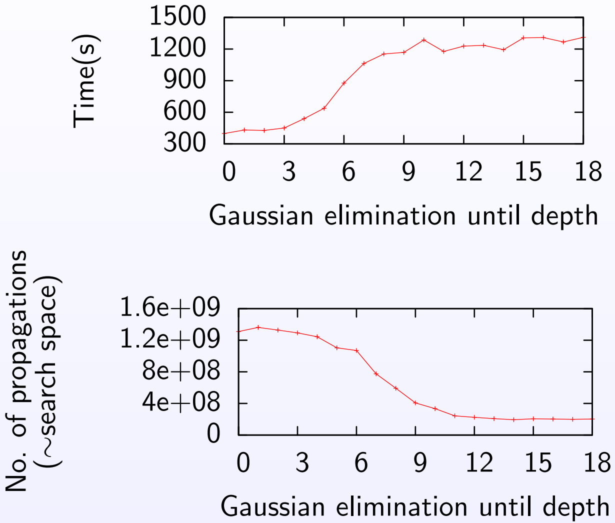
Gaussian elimination

- Gaussian elimination is an efficient algorithm for solving systems of linear equations
- XOR-clause is a linear equation → use Gauss elim. to solve the system of XORs-clauses

xor-clauses with $v8$ assigned to true	actual xor-clauses
$\begin{array}{ccccc} v10 & v8 & v9 & v12 & \text{const} \end{array}$ $\left[\begin{array}{cccc\|c} 1 & - & 1 & 1 & 1 \\ 0 & - & 1 & 1 & 1 \\ 0 & - & 0 & 1 & 0 \\ 0 & - & 0 & 0 & 0 \end{array}\right]$	$\begin{array}{ccccc} v10 & v8 & v9 & v12 & \text{const} \end{array}$ $\left[\begin{array}{cccc\|c} 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{array}\right]$

- make temp. XOR-clause out of the interesting clauses found
- given prop. row 3, save temp. XOR-clause for a short while
- given a conflict, give it to analyze() and delete it

Gaussian elimination results



Visual representation

It's hard to follow how a solver operates. So we implemented dynamic behaviour analysis

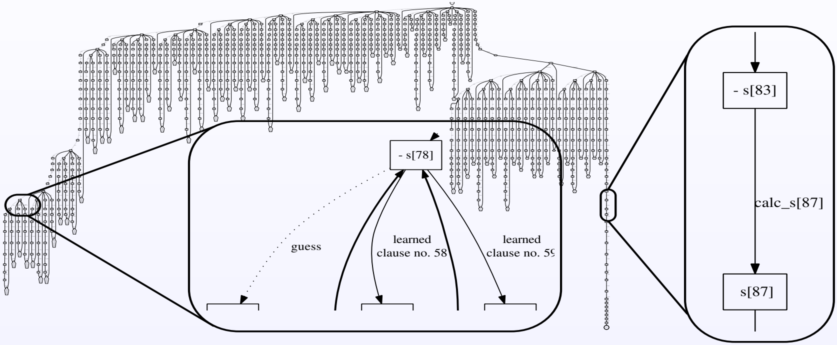


Figure: Graphviz visualisation of an example search for the Crypto-1 cipher's states. The tree is read from left to right, top to bottom: the left- and bottommost pentagon is the first conflict clause, the right- and bottommost circle is the satisfying assignment.

Detailed statistics

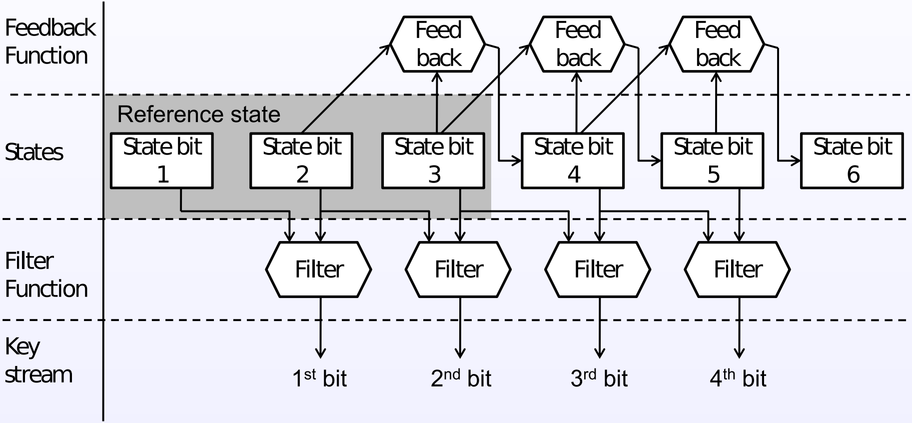
Statistics generated:

- No. times variable branched upon
- Number of conflicts made by clause groups
- Propagation depth order of clause groups
- Avg. conflict depth order of clause groups

Outline

Logical circuit representation

Best to look at the cipher as a logical circuit inside the solver. The logical circuit has variables (boxes), functions (hexagons) and the known keystream.



Measures of the logical circuit representation

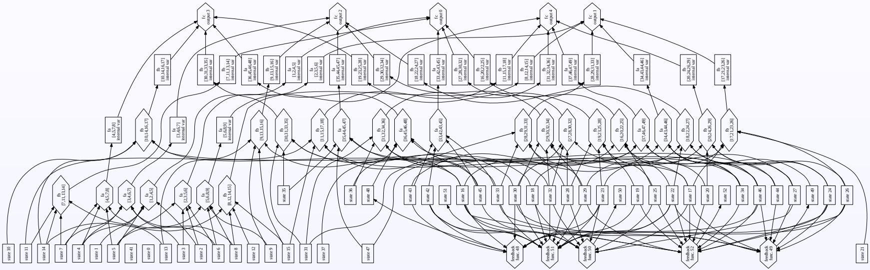
Measures of this logical circuit representation:

- *Depth* of each keystream bit is the number of functions traversed from the reference state
- Reference state *dependency numbers*: no. bits each keystream bit depends on. A large part of these must be guessed before evaluation can take place
- *Function difficulty*. When traversed, these must be calculated

Goal: minimise all of these

Generate logical circuit from CNF

We wrote an extension to MiniSat to visualise the logical circuit. Example HiTag2 logical circuit:



Optimising representation of non-linear functions

Simple $\mathbb{GF}(2)$ polynomial

$$x_1 + x_1x_2 + x_2x_3 + x_1x_3$$

It is usually represented with each non-single monomial expressed as a set of clauses, setting additional variables $i_1 \ldots i_3$. The polynomial then becomes

$$x_1 + i_1 + i_2 + i_3$$

With this representation, no. of clauses is 3×3 regular $+$ 1 xor-clause, avg. clause length 4.14. Three extra variables also needed

However, representation using a Karnaugh table is

$$\bar{x}_1 \vee \bar{x}_3 \qquad \bar{x}_2 \vee x_3 \qquad \bar{x}_1 \vee \bar{x}_2$$

Outline

Crypto-1&HiTag2

Crypto-1

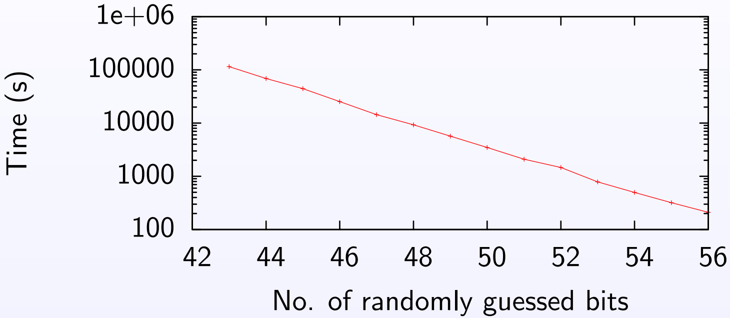
- Best attack with SAT-solvers[Courtois08Algebraic]: 200 seconds, but this uses mathematical means to bring down the complexity (simple, as Crypto-1 uses only an LFSR)
- We break it in 40 seconds.

HiTag2

- Without our optimisation: 2^{21} s to break
- Takes $2^{14.5}$ s to break with our technique

Bivium

Bivium is a simplified version of Trivium. Best attack against it takes 2^{43} s.



We break it in $2^{36.5}$ s.

Thank you for your time

Thank you for your time!