SAT Solvers in the Context of Stream Ciphers

Presentation for Jurnées C2

M. Soos, K. Nohl, C. Castelluccia

PLANETE team INRIA

8th of October 2009

Table of Contents

Outline

Motivations

Wide usage of cryptography

- Authentication (e.g. NaviGO)
- Privacy protection (Tor)

Previous work on SAT solver-based analysis

- Solving Crypto-1 in 200 s
- Solving Bivium B in 2^{43} s

Black-box usage

- Representation not well-optimised for SAT solvers
- Solving not well-examined through statistics
- Solver not optimised for the problem

Goals

Understand the bottlenecks

- Through statistics
- Through visualisations

Remove the bottlenecks

- Adapting the solver to the problem
- Adapting the problem representation to the solver

What is a SAT solver

Solves a problem in CNF

CNF is an "and of or-s"

$$\neg x_1 \vee \neg x_3 \qquad \neg x_2 \vee x_3 \qquad x_1 \vee x_2$$

Uses DPLL(φ) algorithm

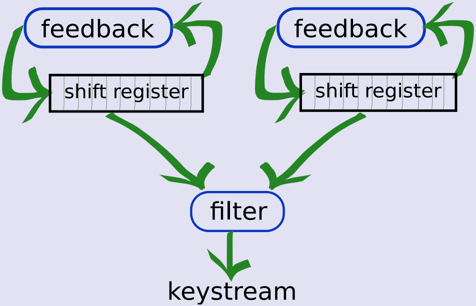
1. If formula φ is trivial, return SAT/UNSAT
2. Picks a variable v to branch on
3. $v \leftarrow$ value
4. Simplifies formula to φ' and calls DPLL(φ')
5. if SAT, output SAT
6. if UNSAT, $v \leftarrow$ opposite value
7. Simplifies formula to φ'' and calls DPLL(φ'')
8. if SAT, output SAT
9. if UNSAT, output UNSAT

Stream ciphers

Shift register-based stream ciphers

- Use 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. . .

Example cipher



Outline

Problem with XOR-s

The truth

$$a \oplus b \oplus c$$

must be put into the solver as

$$a \vee \neg b \vee \neg c \qquad (1) \qquad\qquad \neg a \vee \neg b \vee c \qquad (3)$$
$$a \vee b \vee c \qquad (2) \qquad\qquad \neg a \vee b \vee \neg c \qquad (4)$$

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

Solution until now

Example

$$x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5 \oplus x_6 \oplus x_7 \oplus x_8 \oplus x_9$$

Modelled in CNF:

$$\neg i_1 \oplus x_1 \oplus x_2 \oplus x_3 \oplus x_4$$
$$\neg i_2 \oplus x_5 \oplus x_6 \oplus x_7 \oplus x_8 \oplus x_9$$
$$i_1 \oplus i_2$$

Problems

- Still very long to model
- Needs extra vars

Solution to XOR: xor-clause

Example

$$a \oplus b \oplus c$$

Represents regular clauses

$$a \vee \neg b \vee \neg c \qquad (1) \qquad\qquad \neg a \vee \neg b \vee c \qquad (3)$$
$$a \vee b \vee c \qquad (2) \qquad\qquad \neg a \vee b \vee \neg c \qquad (4)$$

changes appearance to match the situation

Example set-up

$$a = \texttt{true} \quad b = \texttt{true} \quad c = \texttt{false}$$

$$\Rightarrow \neg a \vee \neg b \vee c$$

Solution to XOR: xor-clause

Example

$$a \oplus b \oplus c$$

Represents regular clauses

$$a \vee \neg b \vee \neg c \qquad (1) \qquad\qquad \neg a \vee \neg b \vee c \qquad (3)$$
$$a \vee b \vee c \qquad (2) \qquad\qquad \neg a \vee b \vee \neg c \qquad (4)$$

changes appearance to match the situation

Results

- 2.2x speed
- Order of magnitude savings in memory

Solution to XOR: xor-clause

Example

$$a \oplus b \oplus c$$

Represents regular clauses

$$a \vee \neg b \vee \neg c \qquad (1) \qquad\qquad \neg a \vee \neg b \vee c \qquad (3)$$

$$a \vee b \vee c \qquad (2) \qquad\qquad \neg a \vee b \vee \neg c \qquad (4)$$

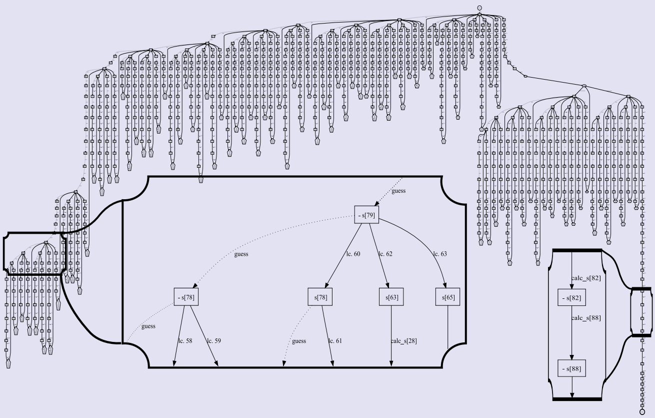
changes appearance to match the situation

Challenges overcome

- MiniSat is complex, we needed to completely understand it
- Design choices were difficult: e.g. we use special memory alloc. to maximise cache-hit

Dynamic behaviour analysis

Example search tree



Visualised

- Guesses
- Propagations
- Generated learnt clauses
- Clause group causing the propagation

Calculated stats

- Average depth
- Most conflicted clauses
- No. of guess/branch
- Most guessed vars
- Most propagated vars

Gaussian elimination

Reasoning

- Gaussian elimination is efficient for solving systems of linear equations
- xor-clause is a linear equation $\rightarrow$ use Gauss elim. to solve them

Implementation

A-matrix

$$\begin{array}{c} \begin{array}{ccccc} v10 & v8 & v9 & v12 & \text{aug} \end{array} \\ \left[\begin{array}{cccc|c} 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{array}\right] \end{array}$$

N-matrix

$$\begin{array}{c} \begin{array}{ccccc} v10 & v8 & v9 & v12 & \text{aug} \end{array} \\ \left[\begin{array}{cccc|c} 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{array}\right] \end{array}$$

Gaussian elimination

Reasoning

- Gaussian elimination is efficient for solving systems of linear equations
- xor-clause is a linear equation $\rightarrow$ use Gauss elim. to solve them

Implementation

A-matrix
with $v8$ assigned to `true`

$$\begin{array}{ccccc} v10 & v8 & v9 & v12 & \text{aug} \end{array}$$
$$\left[\begin{array}{cccc|c} 1 & - & 1 & 1 & 1 \\ 0 & - & 1 & 1 & 1 \\ 0 & - & 0 & 1 & 0 \\ 0 & - & 0 & 0 & 0 \end{array}\right]$$

N-matrix

$$\begin{array}{ccccc} v10 & v8 & v9 & v12 & \text{aug} \end{array}$$
$$\left[\begin{array}{cccc|c} 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{array}\right]$$

Gaussian elimination

Reasoning

- Gaussian elimination is efficient for solving systems of linear equations
- xor-clause is a linear equation $\rightarrow$ use Gauss elim. to solve them

Implementation

A-matrix with $v8$ assigned to `true`

$$\begin{array}{ccccc} v10 & v8 & v9 & v12 & \text{aug} \end{array}$$
$$\left[\begin{array}{cccc|c} 1 & - & 1 & 1 & 1 \\ 0 & - & 1 & 1 & 1 \\ 0 & - & 0 & 1 & 0 \\ 0 & - & 0 & 0 & 0 \end{array}\right]$$

N-matrix

$$\begin{array}{ccccc} v10 & v8 & v9 & v12 & \text{aug} \end{array}$$
$$\left[\begin{array}{cccc|c} 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{array}\right]$$

Resulting xor-clause:

$$v8 \oplus v12$$

Reasoning

- Gaussian elimination is efficient for solving systems of linear equations
- xor-clause is a linear equation $\rightarrow$ use Gauss elim. to solve them

Implementation

A-matrix with $v8$ assigned to `true`	N-matrix
$\begin{array}{ccccc} v10 & v8 & v9 & v12 & \text{aug} \end{array}$ $\left[\begin{array}{cccc\|c} 1 & - & 1 & 1 & 1 \\ 0 & - & 1 & 1 & 1 \\ 0 & - & 0 & 1 & 0 \\ 0 & - & 0 & 0 & 0 \end{array}\right]$	$\begin{array}{ccccc} v10 & v8 & v9 & v12 & \text{aug} \end{array}$ $\left[\begin{array}{cccc\|c} 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{array}\right]$

Resulting xor-clause:

$$v12 = \texttt{false} \quad \leftarrow \quad v8 \oplus v12$$

Gaussian elimination results

	No. help bits	Gaussian elimination active until level		
		Inactive	2	3
Crypto-1	12	27.0 s	25.8 s(4%)	26.5 s(2%)
HiTag2	18	34.8 s	33.9 s(3%)	29.5 s(15%)
Bivium B	60	174.0 s	165.1 s(5%)	171.1 s(2%)

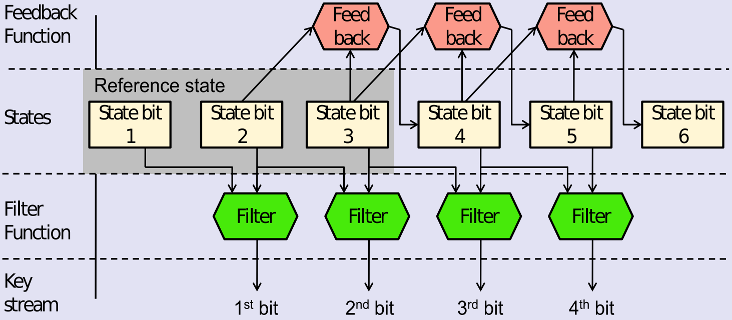
Highlights

- Search space reduced by up to 87%
- Speedup between 0-15%
- A mix of linear and non-linear methods
- Adds possibility to add other algebraic tools $\longrightarrow$ potentially major speedup

Outline

Logical circuit representation

Example



Legend

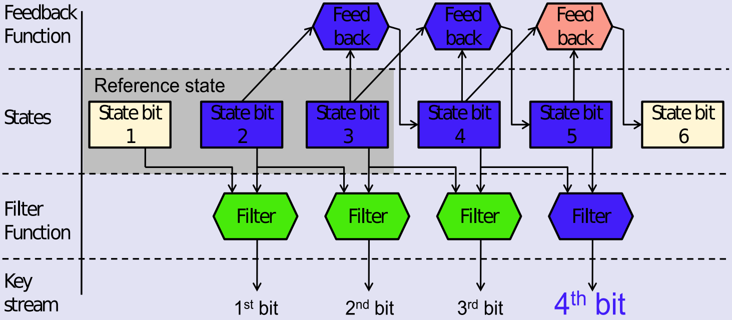
- Variables → boxes
- Functions → hexagons

Complexity measures

- *Depth* of keystream bit
- *Dependency no.*: state ↔ keystream
- *Difficulty of functions*: representation

Logical circuit representation

Example



Legend

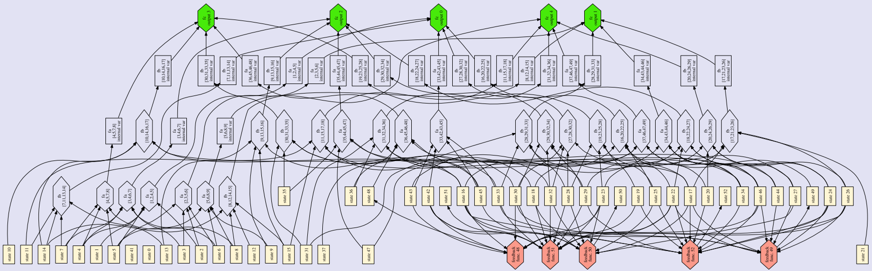
- Variables → boxes
- Functions → hexagons

Complexity measures

- *Depth* of keystream bit
- *Dependency no.*: state ↔ keystream
- *Difficulty of functions*: representation

Dependency graph generator

Example HiTag2 logical circuit

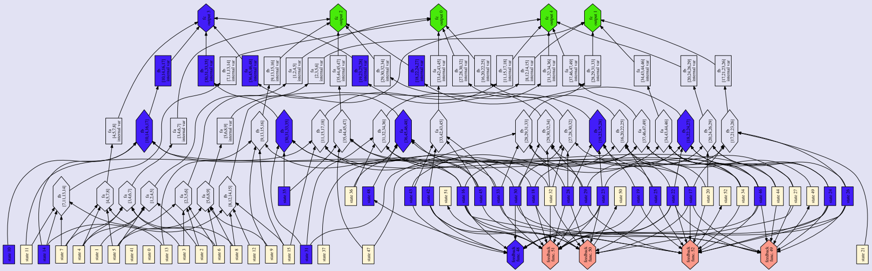


Usage

- Calculate mentioned statistics
- Visual clue

Dependency graph generator

Example HiTag2 logical circuit



Usage

- Calculate mentioned statistics
- Visual clue

Optimising representation of non-linear functions

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

$$x_1 + x_1x_2 + x_2x_3 + x_1x_3$$

Usual representation

$$x_1 + i_1 + i_2 + i_3$$

- No. of clauses: 3×3 regular + 1 xor-clause
- $\sum$ clause length: 31
- 2 extra variables

Karnaugh-table representation

$$\neg x_1 \vee \neg x_3 \quad \neg x_2 \vee x_3 \quad x_1 \vee x_2$$

- No. of clauses: 3 regular
- $\sum$ clause length: 6
- No extra variables

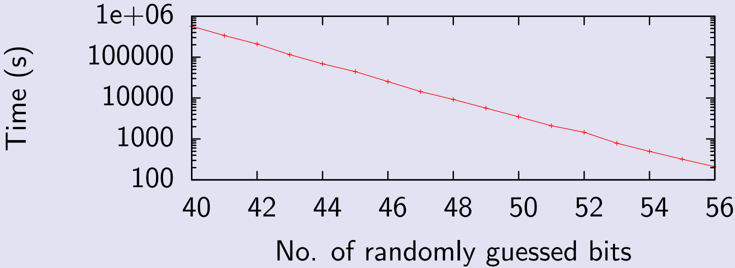
Outline

Bivium B

Background

- Simplified version of Trivium eSTREAM candidate
- Best SAT solver-based attack against it takes 2^{43} s
- Non-SAT solver-based attack: $2^{64.5}$ s

Our techniques



Find its secret state in approx. $2^{36.5}$ s

Outline

Conclusions

Conclusions

- SAT solvers have large potential for cryptanalysis
- For best results we need to adapt the problem and solver to each other
- Such a system is able to break certain ciphers

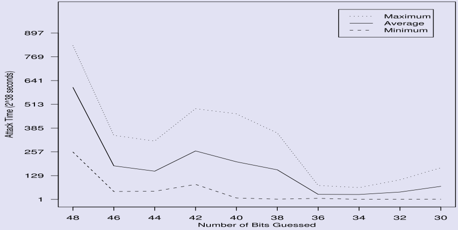
Possible future work

- Further enhance SAT solvers for stream ciphers
- Better understand the solving process to arrive at better problem representation
- Use generated statistics for understanding the cipher

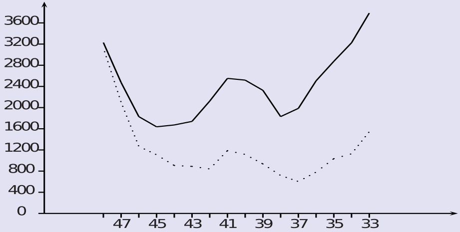
Thank you for your time

Research results until now

"Attacking Bivium with MiniSat" by (McDonald et al.)



"Attacking Bivium Using SAT Solvers" by (Eibach et al.)



Research results until now

We introduce more randomness

- Reference state bits to assign are picked randomly
- The picked bits are assigned randomly `true` or `false`
- Clauses are randomly permutated inside MiniSat
- MiniSat's internal seed (used to randomly explore the search space) is set randomly
- MiniSat's random number generator has been replaced