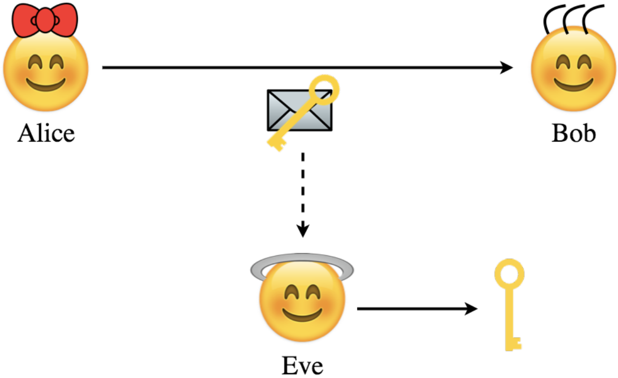
Logical Cryptanalysis with WDSat

Monika Trimoska Gilles Dequen Sorina Ionica

MIS, University of Picardie Jules Verne

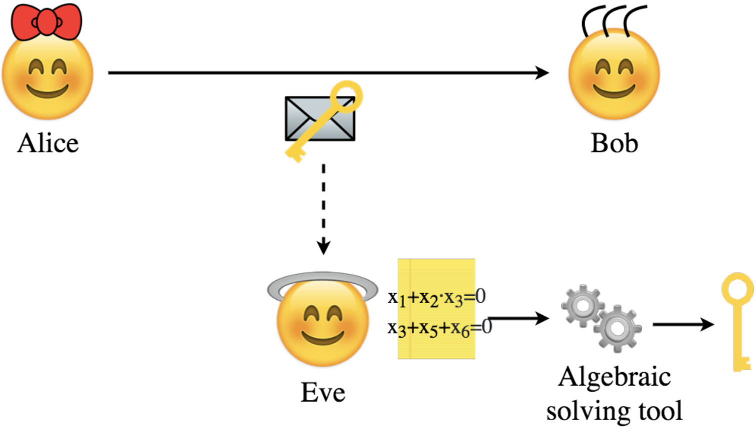
SAT 2021



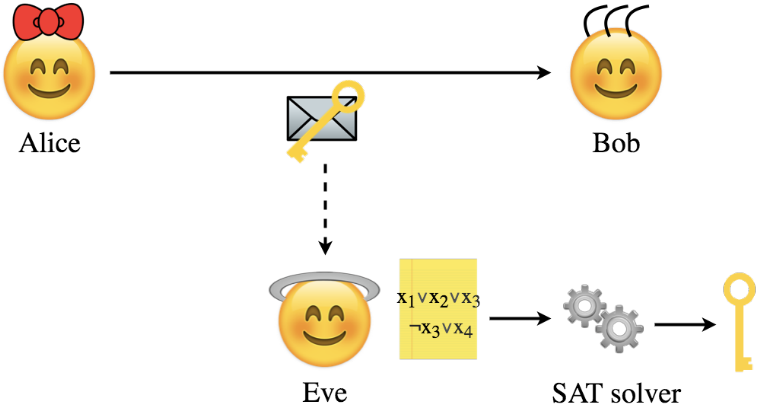
Goal

Determine minimum cryptographic key length requirements.

Algebraic cryptanalysis



Logical cryptanalysis



The multivariate polynomial problem

Example. A multivariate polynomial system of three equations in three variables

$$\mathbf{x}_1 + \mathbf{x}_2 \cdot \mathbf{x}_3 = 0$$
$$\mathbf{x}_1 \cdot \mathbf{x}_2 + \mathbf{x}_2 + \mathbf{x}_3 = 0$$
$$\mathbf{x}_1 + \mathbf{x}_1 \cdot \mathbf{x}_2 \cdot \mathbf{x}_3 + \mathbf{x}_2 \cdot \mathbf{x}_3 = 0.$$

At the core of algebraic cryptanalysis: finding a solution to the multivariate polynomial system results in recovering the secret key or the plaintext.

The degree-two case is the underlying problem in one of the five families of post-quantum cryptographic schemes.

From the algebraic model to the CNF-XOR model

Variables in $\mathbb{F}_2$:
$\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6$.

Propositional variables:
$x_1, x_2, x_3, x_4, x_5, x_6$ with truth values in $\{\text{TRUE}, \text{FALSE}\}$

$$\mathbf{x}_1 + \mathbf{x}_2 \cdot \mathbf{x}_4 + \mathbf{x}_5 \cdot \mathbf{x}_6 + 1 = 0$$
$$\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_4 + \mathbf{x}_5 + 1 = 0$$
$$\mathbf{x}_3 + \mathbf{x}_4 + \mathbf{x}_2 \cdot \mathbf{x}_4 = 0$$
$$\mathbf{x}_2 + \mathbf{x}_5 + \mathbf{x}_2 \cdot \mathbf{x}_4 + \mathbf{x}_5 \cdot \mathbf{x}_6 + 1 = 0$$
$$\mathbf{x}_3 + \mathbf{x}_4 + \mathbf{x}_6 + 1 = 0$$

$$(x_1 \oplus (x_2 \wedge x_4) \oplus (x_5 \wedge x_6)) \wedge$$
$$(x_1 \oplus x_2 \oplus x_4 \oplus x_5) \wedge$$
$$(x_3 \oplus x_4 \oplus (x_2 \wedge x_4) \oplus \top) \wedge$$
$$(x_2 \oplus x_5 \oplus (x_2 \wedge x_4) \oplus (x_5 \wedge x_6)) \wedge$$
$$(x_3 \oplus x_4 \oplus x_6)$$

Multiplication in $\mathbb{F}_2$ $(\cdot)$ becomes the logical AND operation $(\wedge)$ and addition in $\mathbb{F}_2$ $(+)$ becomes the logical XOR $(\oplus)$.

Add new variable $x_{2,4}$ to substitute the conjunction $x_2 \wedge x_4$.

Transform the constraint

$$x_{2,4} \Leftrightarrow (x_2 \wedge x_4)$$

into CNF.

From the algebraic model to the CNF-XOR model

Propositional variables:

$x_1, x_2, x_3, x_4, x_5, x_6, x_{2,4}, x_{5,6}$ with truth values in $\{\text{TRUE}, \text{FALSE}\}$

$$(x_1 \oplus (x_2 \wedge x_4) \oplus (x_5 \wedge x_6)) \wedge$$
$$(x_1 \oplus x_2 \oplus x_4 \oplus x_5) \wedge$$
$$(x_3 \oplus x_4 \oplus (x_2 \wedge x_4) \oplus \top) \wedge$$
$$(x_2 \oplus x_5 \oplus (x_2 \wedge x_4) \oplus (x_5 \wedge x_6)) \wedge$$
$$(x_3 \oplus x_4 \oplus x_6)$$

$$(\neg x_{2,4} \vee x_2) \wedge$$
$$(\neg x_{2,4} \vee x_4) \wedge$$
$$(\neg x_2 \vee \neg x_4 \vee x_{2,4}) \wedge$$
$$(\neg x_{5,6} \vee x_5) \wedge$$
$$(\neg x_{5,6} \vee x_6) \wedge$$
$$(\neg x_5 \vee \neg x_6 \vee x_{5,6}) \wedge$$
$$(x_1 \oplus x_{2,4} \oplus x_{5,6}) \wedge$$
$$(x_1 \oplus x_2 \oplus x_4 \oplus x_5) \wedge$$
$$(x_3 \oplus x_4 \oplus x_{2,4} \oplus \top) \wedge$$
$$(x_2 \oplus x_5 \oplus x_{2,4} \oplus x_{5,6}) \wedge$$
$$(x_3 \oplus x_4 \oplus x_6)$$

The WDSat solver

WDSat algorithm

Based on the Davis-Putnam-Logemann-Loveland (DPLL) algorithm.

Three reasoning modules

- **CNF module** : Performs unit propagation on CNF-clauses.
- **XORSET module** : Performs unit propagation on the parity constraints. When all except one literal in a XOR clause is assigned, we infer the truth value of the last literal according to parity reasoning.
- **XORGAUSS module** : Performs Gaussian elimination on the XOR system.

OR-clauses are stored as bit-vectors comprised of three parts.

$$x_1 \vee x_3$$

0	0	1	0	0	0	1	0	0
value					*weight*			*sat*

Value

The arithmetic sum of the literals in the clause in their dimacs representation.

Weight

The number of unassigned literals left in the clause.

Sat slot

Set to 1 when the clause is already satisfied by one of its assigned literals, and to 0 otherwise.

Example.

$\neg x_1 \vee \neg x_2 \vee x_4$ $((-1-2+4 \ll 3)+3) \ll 1$	0	0	0	0	1	0	1	1	0
$x_1 \vee x_3$ $((1+3 \ll 3)+2) \ll 1$	0	0	1	0	0	0	1	0	0

Compressed CNF Reasoning

Example.

$\neg x_1 \vee \neg x_2 \vee x_4$ $((-1-2+4 \ll 3)+3) \ll 1$	0	0	0	0	1	0	1	1	0
$x_1 \vee x_3$ $((1+3 \ll 3)+2) \ll 1$	0	0	1	0	0	0	1	0	0

Set x_1 to FALSE.

$\neg x_1 \vee \neg x_2 \vee x_4$ $\vert 1$	0	0	0	0	1	0	1	1	1
x_3 $-(((1 \ll 3)+1) \ll 1)$	0	0	0	1	1	0	0	1	0

Example.

$\neg x_1 \vee \neg x_2 \vee x_4$ $((-1-2+4 \ll 3)+3) \ll 1$	0	0	0	0	1	0	1	1	0
$x_1 \vee x_3$ $((1+3 \ll 3)+2) \ll 1$	0	0	1	0	0	0	1	0	0

Set x_1 to FALSE.

$\neg x_1 \vee \neg x_2 \vee x_4$ $\|1$	0	0	0	0	1	0	1	1	1
x_3 $-(((1 \ll 3)+1) \ll 1)$	0	0	0	1	1	0	0	1	0

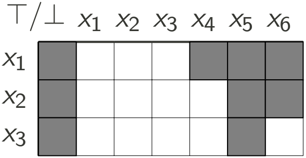
Propagation x_3 is set to TRUE.

WDSat - XORGAUSS module

- All variables in an XOR-clause belong to the same equivalence class.
- We choose one literal from the equivalence class to be the representative.
- Property: a representative of an equivalence class will never be present in another equivalence class.

XOR-clauses	Equivalence classes
$x_1 \oplus x_4 \oplus x_5 \oplus x_6$	$x_1 \Leftrightarrow x_4 \oplus x_5 \oplus x_6 \oplus \top$
$x_1 \oplus x_2 \oplus x_4 \oplus \top$	$x_2 \Leftrightarrow x_5 \oplus x_6 \oplus \top$
$x_2 \oplus x_3 \oplus x_6 \oplus \top$	$x_3 \Leftrightarrow x_5 \oplus \top$

- Implementation: A compact *EC* structure.



WDSat - XORGAUSS module

- All variables in an XOR-clause belong to the same equivalence class.
- We choose one literal from the equivalence class to be the representative.
- Property: a representative of an equivalence class will never be present in another equivalence class.

	XOR-clauses	Equivalence classes
	$x_1 \oplus x_4 \oplus x_5 \oplus x_6$	$x_1 \Leftrightarrow x_4 \oplus x_5 \oplus x_6 \oplus \top$
$x_2 \oplus x_5 \oplus x_6$	~~$x_1 \oplus x_2 \oplus x_4 \oplus \top$~~	$x_2 \Leftrightarrow x_5 \oplus x_6 \oplus \top$
	$x_2 \oplus x_3 \oplus x_6 \oplus \top$	$x_3 \Leftrightarrow x_5 \oplus \top$

- Implementation: A compact *EC* structure.

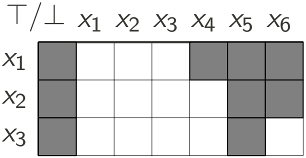
	$\top/\bot$	x_1	x_2	x_3	x_4	x_5	x_6
x_1	■				■	■	■
x_2	■					■	■
x_3	■					■	

WDSat - XORGAUSS module

- All variables in an XOR-clause belong to the same equivalence class.
- We choose one literal from the equivalence class to be the representative.
- Property: a representative of an equivalence class will never be present in another equivalence class.

	XOR-clauses	Equivalence classes
	$x_1 \oplus x_4 \oplus x_5 \oplus x_6$	$x_1 \Leftrightarrow x_4 \oplus x_5 \oplus x_6 \oplus \top$
$x_2 \oplus x_5 \oplus x_6$	~~$x_1 \oplus x_2 \oplus x_4 \oplus \top$~~	$x_2 \Leftrightarrow x_5 \oplus x_6 \oplus \top$
$x_3 \oplus x_5$	~~$x_2 \oplus x_3 \oplus x_6 \oplus \top$~~	$x_3 \Leftrightarrow x_5 \oplus \top$

- Implementation: A compact *EC* structure.



Setting x_6 to TRUE

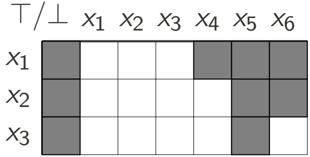
Algorithm 1 Function INFER_NON_REPRESENTATIVE(ul, tv, F)

Input: Propositional variable ul, truth value tv, the propositional formula F

Output: The EC structure is modified.

```
add ul to R.
if tv = TRUE then
    FLIP_CONSTANT(EC[ul]).
end if
set ul to 1 in EC[ul].
for each r in R do
    if ul is set to 1 in EC[r] then
        EC[r] ← EC[r] ⊕ EC[ul].
        if all variable bits in EC[r] are set to 0 then
            if the constant bit in EC[r] is set to 1 then
                add r to XG_propagation_stack.
            else
                add ¬r to XG_propagation_stack.
            end if
        end if
    end if
end for
set ul to 0 in EC[ul].
```

Before execution:



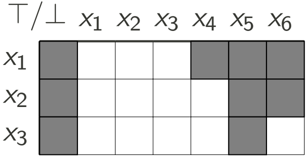
Setting x_6 to TRUE

Algorithm 2 Function INFER_NON_REPRESENTATIVE(ul, tv, F)

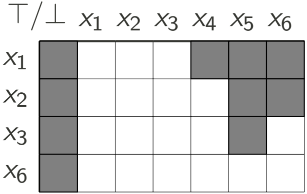
Input: Propositional variable ul, truth value tv, the propositional formula F
Output: The EC structure is modified.

1: add ul to R.
2: **if** tv =TRUE **then**
3: FLIP_CONSTANT($EC[ul]$).
4: **end if**
5: set ul to 1 in $EC[ul]$.
6: **for each** r in R **do**
7: **if** ul is set to 1 in $EC[r]$ **then**
8: $EC[r] \leftarrow EC[r] \oplus EC[ul]$.
9: **if** all variable bits in $EC[r]$ are set to 0 **then**
10: **if** the constant bit in $EC[r]$ is set to 1 **then**
11: add r to $XG\_propagation\_stack$.
12: **else**
13: add $\neg r$ to $XG\_propagation\_stack$.
14: **end if**
15: **end if**
16: **end if**
17: **end for**
18: set ul to 0 in $EC[ul]$.

Before execution:



After line 3:



Setting x_6 to TRUE

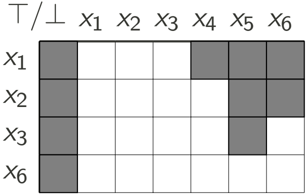
Algorithm 3 Function INFER_NON_REPRESENTATIVE(ul, tv, F)

Input: Propositional variable ul, truth value tv, the propositional formula F

Output: The EC structure is modified.

1: add ul to R.
2: **if** tv =TRUE **then**
3: FLIP_CONSTANT($EC[ul]$).
4: **end if**
5: set ul to 1 in $EC[ul]$.
6: **for each** r in R **do**
7: **if** ul is set to 1 in $EC[r]$ **then**
8: $EC[r] \leftarrow EC[r] \oplus EC[ul]$.
9: **if** all variable bits in $EC[r]$ are set to 0 **then**
10: **if** the constant bit in $EC[r]$ is set to 1 **then**
11: add r to $XG\_propagation\_stack$.
12: **else**
13: add $\neg r$ to $XG\_propagation\_stack$.
14: **end if**
15: **end if**
16: **end if**
17: **end for**
18: set ul to 0 in $EC[ul]$.

After line 3:



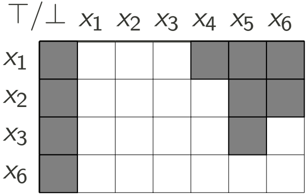
Setting x_6 to TRUE

Algorithm 4 Function INFER_NON_REPRESENTATIVE(ul, tv, F)

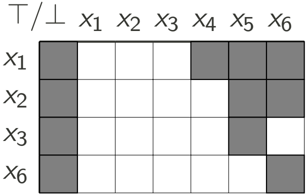
Input: Propositional variable ul, truth value tv, the propositional formula F
Output: The EC structure is modified.

```
1:  add ul to R.
2:  if tv = TRUE then
3:      FLIP_CONSTANT(EC[ul]).
4:  end if
5:  set ul to 1 in EC[ul].
6:  for each r in R do
7:      if ul is set to 1 in EC[r] then
8:          EC[r] ← EC[r] ⊕ EC[ul].
9:          if all variable bits in EC[r] are set to 0 then
10:             if the constant bit in EC[r] is set to 1 then
11:                 add r to XG_propagation_stack.
12:             else
13:                 add ¬r to XG_propagation_stack.
14:             end if
15:         end if
16:     end if
17: end for
18: set ul to 0 in EC[ul].
```

After line 3:



After line 5:



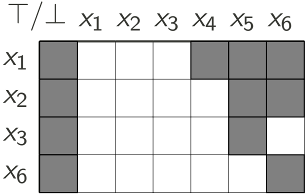
Setting x_6 to TRUE

Algorithm 5 Function INFER_NON_REPRESENTATIVE(*ul*, *tv*, *F*)

Input: Propositional variable *ul*, truth value *tv*, the propositional formula *F*
Output: The *EC* structure is modified.

1: add *ul* to *R*.
2: **if** *tv* = TRUE **then**
3: FLIP_CONSTANT($EC[ul]$).
4: **end if**
5: set *ul* to 1 in $EC[ul]$.
6: **for each** *r* in *R* **do**
7: **if** *ul* is set to 1 in $EC[r]$ **then**
8: $EC[r] \leftarrow EC[r] \oplus EC[ul]$.
9: **if** all variable bits in $EC[r]$ are set to 0 **then**
10: **if** the constant bit in $EC[r]$ is set to 1 **then**
11: add *r* to *XG_propagation_stack*.
12: **else**
13: add $\neg r$ to *XG_propagation_stack*.
14: **end if**
15: **end if**
16: **end if**
17: **end for**
18: set *ul* to 0 in $EC[ul]$.

After line 5:



XORGAUSS infer algorithm

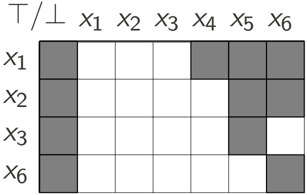
Setting x_6 to TRUE

Algorithm 6 Function INFER_NON_REPRESENTATIVE(ul, tv, F)

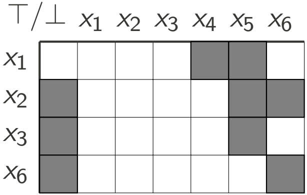
Input: Propositional variable ul, truth value tv, the propositional formula F
Output: The EC structure is modified.

```
 1: add ul to R.
 2: if tv = TRUE then
 3:     FLIP_CONSTANT(EC[ul]).
 4: end if
 5: set ul to 1 in EC[ul].
 6: for each r in R do
 7:     if ul is set to 1 in EC[r] then
 8:         EC[r] ← EC[r] ⊕ EC[ul].
 9:         if all variable bits in EC[r] are set to 0 then
10:             if the constant bit in EC[r] is set to 1 then
11:                 add r to XG_propagation_stack.
12:             else
13:                 add ¬r to XG_propagation_stack.
14:             end if
15:         end if
16:     end if
17: end for
18: set ul to 0 in EC[ul].
```

After line 5:



After line 8:



XORGAUSS infer algorithm

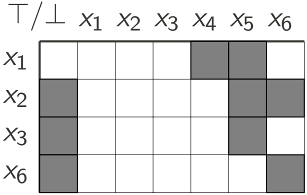
Setting x_6 to TRUE

Algorithm 7 Function INFER_NON_REPRESENTATIVE(ul, tv, F)

Input: Propositional variable ul, truth value tv, the propositional formula F
Output: The EC structure is modified.

1: add ul to R.
2: **if** tv = TRUE **then**
3:     FLIP_CONSTANT($EC[ul]$).
4: **end if**
5: set ul to 1 in $EC[ul]$.
6: **for each** r in R **do**
7:     **if** ul is set to 1 in $EC[r]$ **then**
8:         $EC[r] \leftarrow EC[r] \oplus EC[ul]$.
9:         **if** all variable bits in $EC[r]$ are set to 0 **then**
10:             **if** the constant bit in $EC[r]$ is set to 1 **then**
11:                 add r to *XG_propagation_stack*.
12:             **else**
13:                 add $\neg r$ to *XG_propagation_stack*.
14:             **end if**
15:         **end if**
16:     **end if**
17: **end for**
18: set ul to 0 in $EC[ul]$.

After line 8:



XORGAUSS infer algorithm

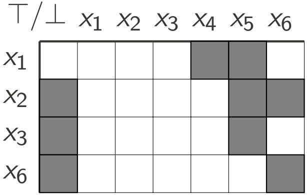
Setting x_6 to TRUE

Algorithm 8 Function INFER_NON_REPRESENTATIVE(ul, tv, F)

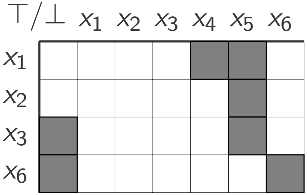
Input: Propositional variable ul, truth value tv, the propositional formula F
Output: The EC structure is modified.

```
1: add ul to R.
2: if tv = TRUE then
3:     FLIP_CONSTANT(EC[ul]).
4: end if
5: set ul to 1 in EC[ul].
6: for each r in R do
7:     if ul is set to 1 in EC[r] then
8:         EC[r] ← EC[r] ⊕ EC[ul].
9:         if all variable bits in EC[r] are set to 0 then
10:            if the constant bit in EC[r] is set to 1 then
11:                add r to XG_propagation_stack.
12:            else
13:                add ¬r to XG_propagation_stack.
14:            end if
15:        end if
16:    end if
17: end for
18: set ul to 0 in EC[ul].
```

After line 8:



After line 8:



XORGAUSS infer algorithm

Setting x_6 to TRUE

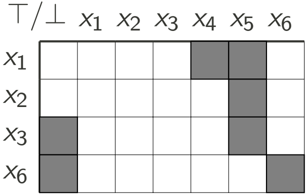
Algorithm 9 Function INFER_NON_REPRESENTATIVE(ul, tv, F)

Input: Propositional variable ul, truth value tv, the propositional formula F

Output: The EC structure is modified.

```
1: add ul to R.
2: if tv = TRUE then
3:     FLIP_CONSTANT(EC[ul]).
4: end if
5: set ul to 1 in EC[ul].
6: for each r in R do
7:     if ul is set to 1 in EC[r] then
8:         EC[r] ← EC[r] ⊕ EC[ul].
9:         if all variable bits in EC[r] are set to 0 then
10:            if the constant bit in EC[r] is set to 1 then
11:                add r to XG_propagation_stack.
12:            else
13:                add ¬r to XG_propagation_stack.
14:            end if
15:        end if
16:    end if
17: end for
18: set ul to 0 in EC[ul].
```

After line 8:



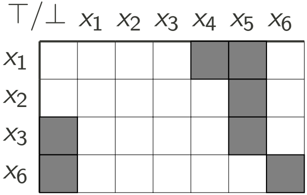
Setting x_6 to TRUE

Algorithm 10 Function INFER_NON_REPRESENTATIVE(ul, tv, F)

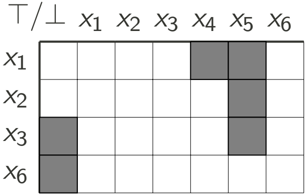
Input: Propositional variable ul, truth value tv, the propositional formula F
Output: The EC structure is modified.

1: add ul to R.
2: **if** tv =TRUE **then**
3: FLIP_CONSTANT($EC[ul]$).
4: **end if**
5: set ul to 1 in $EC[ul]$.
6: **for each** r in R **do**
7: **if** ul is set to 1 in $EC[r]$ **then**
8: $EC[r] \leftarrow EC[r] \oplus EC[ul]$.
9: **if** all variable bits in $EC[r]$ are set to 0 **then**
10: **if** the constant bit in $EC[r]$ is set to 1 **then**
11: add r to $XG\_propagation\_stack$.
12: **else**
13: add $\neg r$ to $XG\_propagation\_stack$.
14: **end if**
15: **end if**
16: **end if**
17: **end for**
18: set ul to 0 in $EC[ul]$.

After line 8:



After line 18.



Comparing different SAT approaches for solving Boolean polynomial systems with 50 quadratic equations over 25 variables.

- Results show an average of 100 runs.
- Running times are in seconds.

Input form	#Vars	#Clauses	Solver	Runtime	#Conflicts
CNF	8301	33006	MiniSat	11525.24	40718489
			Glucose	2384.99	10982657
			Kissat	2118.52	6622284
			Relaxed	3014.22	10353009
CNF-XOR	325	920	CryptoMiniSat	2870.81	9197978
			CryptoMiniSat + GE	594.48	2407635
			WDSat	57.85	14177200
			WDSat + GE	23.77	1046328
ANF	25	50	WDSat + XG-EXT	**0.82**	**21140**

Conclusion

- WDSat outperforms state-of-the-art SAT solvers for instances derived from dense Boolean polynomial systems.
- The compressed CNF reasoning module allows WDSat to handle polynomial systems of higher degree without compromising its performance.

WDSat on github

https://github.com/mtrimoska/WDSat