CDCL(Crypto) SAT Solvers for Cryptanalysis

Saeed Nejati and Vijay Ganesh

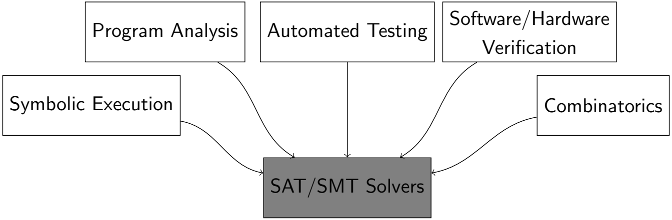
November 4th

CASCON 2019

University of Waterloo

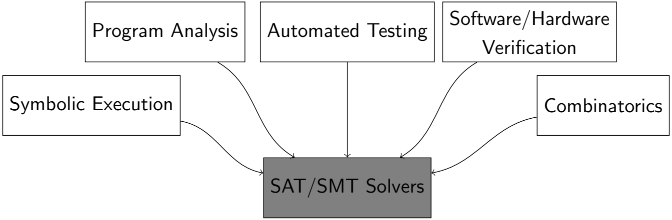
Motivation

- **SAT Solvers**: Powerful general purpose search tools



Motivation

- **SAT Solvers**: Powerful general purpose search tools
- **Cryptanalysis**: Searching a huge search space for a secret key/value



Motivation

- SAT/SMT solvers have increasingly been used in Cryptographic tasks

Motivation

- SAT/SMT solvers have increasingly been used in Cryptographic tasks
  - Finding cryptographic keys [Mas99, MM00]
  - Modular root finding [FMM03]
  - A collision attack [MZ06]
  - Preimage attacks [MS13], [Nos12]
  - Differential cryptanalysis [Pro16]
  - RX-differentials [Ashur2017], [DW17]
  - Verification of cryptographic primitives [Tom15]

Motivation

- SAT/SMT solvers have increasingly been used in Cryptographic tasks
  - Finding cryptographic keys [Mas99, MM00]
  - Modular root finding [FMM03]
  - A collision attack [MZ06]
  - Preimage attacks [MS13], [Nos12]
  - Differential cryptanalysis [Pro16]
  - RX-differentials [Ashur2017], [DW17]
  - Verification of cryptographic primitives [Tom15]
- Mostly used as a black-box solver for a reduced equation system

Motivation

- SAT/SMT solvers have increasingly been used in Cryptographic tasks
  - Finding cryptographic keys [Mas99, MM00]
  - Modular root finding [FMM03]
  - A collision attack [MZ06]
  - Preimage attacks [MS13], [Nos12]
  - Differential cryptanalysis [Pro16]
  - RX-differentials [Ashur2017], [DW17]
  - Verification of cryptographic primitives [Tom15]
- Mostly used as a black-box solver for a reduced equation system

Question

Can we tailor internals of a SAT solver for a specific cryptographic problem to improve the solving time?

Outline

Boolean SAT Solvers

- CDCL SAT Solvers

The CDCL(Crypto) Framework

- Programmatic SAT Architecture

Case Studies

- Algebraic Fault Attack
- Differential Cryptanalysis

Boolean SAT Solvers

Boolean SATisfiability

- NP-complete problem

Boolean SATisfiability

- NP-complete problem
- Given a Boolean formula, determine if it is satisfiable.

Boolean SATisfiability

- NP-complete problem
- Given a Boolean formula, determine if it is satisfiable.
- Boolean formula: an expression involving Boolean variables and logical connectives $\neg, \wedge, \vee$.

Boolean SATisfiability

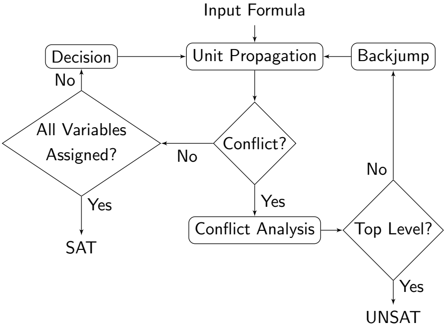
- NP-complete problem
- Given a Boolean formula, determine if it is satisfiable.
- Boolean formula: an expression involving Boolean variables and logical connectives $\neg, \wedge, \vee$.
- A formula is satisfiable, if there exists an assignment to the variables which the formula true.

Boolean SATisfiability

- NP-complete problem
- Given a Boolean formula, determine if it is satisfiable.
- Boolean formula: an expression involving Boolean variables and logical connectives $\neg, \wedge, \vee$.
- A formula is satisfiable, if there exists an assignment to the variables which the formula true.
- Example: $(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y) \wedge z$

CDCL SAT Solver

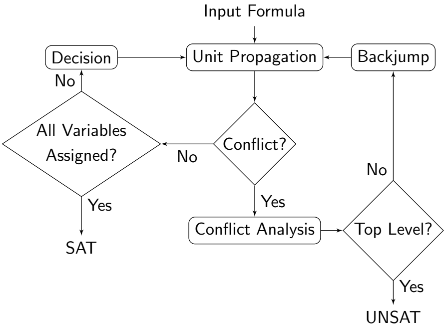
Unit propagation



CDCL SAT Solver

Unit propagation

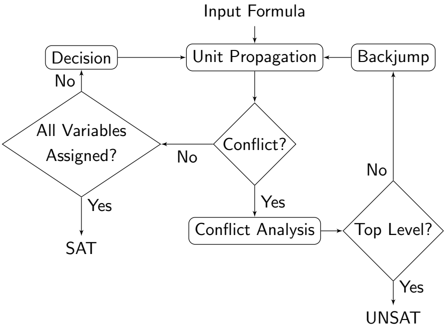
- $(x \vee z) \wedge (\neg x \vee \neg y) \wedge (\neg z)$



CDCL SAT Solver

Unit propagation

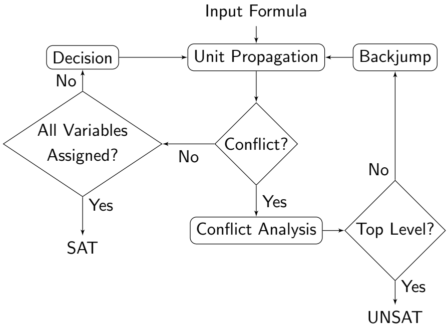
- $(x \vee z) \wedge (\neg x \vee \neg y) \wedge (\neg z)$
- $(x) \wedge (\neg x \vee \neg y)$



CDCL SAT Solver

Unit propagation

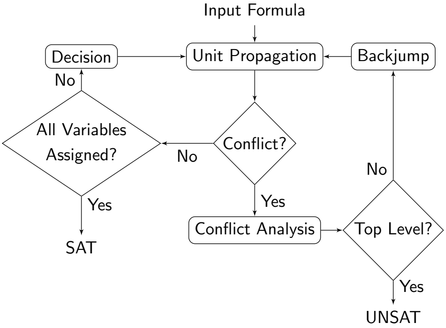
- $(x \vee z) \wedge (\neg x \vee \neg y) \wedge (\neg z)$
- $(x) \wedge (\neg x \vee \neg y)$
- (y)



CDCL SAT Solver

Unit propagation

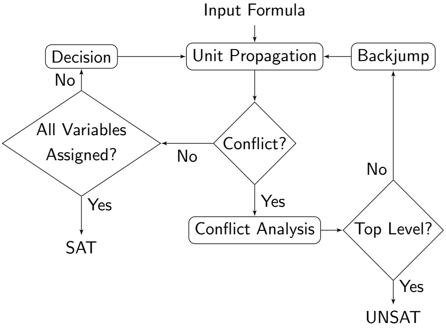
- $(x \vee z) \wedge (\neg x \vee \neg y) \wedge (\neg z)$
- $(x) \wedge (\neg x \vee \neg y)$
- (y)
- $\neg z, \quad \neg z \rightarrow x, \quad x \rightarrow \neg y$



CDCL SAT Solver

Decision/Branching heuristics

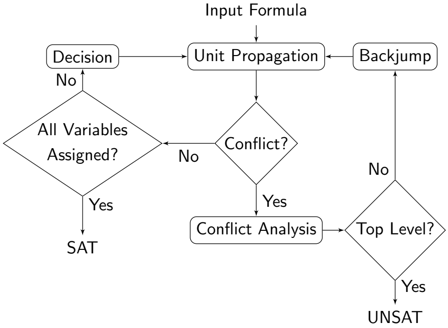
- Pick an unassigned variable and set it to False/True



CDCL SAT Solver

Conflict analysis

- Find the root cause of conflict
- Encode it as a clause and add it back to the formula



The CDCL(Crypto) Framework

Lost in Translation

- When encoding a constraint into SAT, some higher level properties might be lost
- Example: consider a pseudo-Boolean constraint $C : x + y \leq 0$
  - We trivially know: $C \rightarrow \neg x$ and $C \rightarrow \neg y$.

Lost in Translation

- When encoding a constraint into SAT, some higher level properties might be lost
- Example: consider a pseudo-Boolean constraint $C : x + y \leq 0$
  - We trivially know: $C \to \neg x$ and $C \to \neg y$.
  - We can encode it using a half-adder

Lost in Translation

- When encoding a constraint into SAT, some higher level properties might be lost
- Example: consider a pseudo-Boolean constraint $C : x + y \leq 0$
  - We trivially know: $C \rightarrow \neg x$ and $C \rightarrow \neg y$.
  - We can encode it using a half-adder
  - $sum \leftrightarrow x \oplus y$, $carry \leftrightarrow x \wedge y$, and adding constraints $sum = 0, carry = 0$.

Lost in Translation

- When encoding a constraint into SAT, some higher level properties might be lost
- Example: consider a pseudo-Boolean constraint $C : x + y \leq 0$
  - We trivially know: $C \rightarrow \neg x$ and $C \rightarrow \neg y$.
  - We can encode it using a half-adder
  - $sum \leftrightarrow x \oplus y$, $carry \leftrightarrow x \wedge y$, and adding constraints $sum = 0, carry = 0$.
  - Resultant CNF: $(\neg x \vee \neg y) \wedge (\neg x \vee y) \wedge (x \vee y)$

Lost in Translation

- When encoding a constraint into SAT, some higher level properties might be lost
- Example: consider a pseudo-Boolean constraint $C : x + y \leq 0$
  - We trivially know: $C \rightarrow \neg x$ and $C \rightarrow \neg y$.
  - We can encode it using a half-adder
  - $sum \leftrightarrow x \oplus y$, $carry \leftrightarrow x \wedge y$, and adding constraints $sum = 0, carry = 0$.
  - Resultant CNF: $(\neg x \vee \neg y) \wedge (\neg x \vee y) \wedge (x \vee y)$
  - No unit clause to propagate!

Lost in Translation

- When encoding a constraint into SAT, some higher level properties might be lost
- Example: consider a pseudo-Boolean constraint $C : x + y \leq 0$
  - We trivially know: $C \rightarrow \neg x$ and $C \rightarrow \neg y$.
  - We can encode it using a half-adder
  - $sum \leftrightarrow x \oplus y$, $carry \leftrightarrow x \wedge y$, and adding constraints $sum = 0, carry = 0$.
  - Resultant CNF: $(\neg x \vee \neg y) \wedge (\neg x \vee y) \wedge (x \vee y)$
  - No unit clause to propagate!

- "Better" encoding vs. "Better" Propagation

Programmatic SAT

- Instrumenting a SAT solver with *callbacks*

Programmatic SAT

- Instrumenting a SAT solver with *callbacks*
- Extending functionality of propagation and conflict analysis

Programmatic SAT

- Instrumenting a SAT solver with *callbacks*
- Extending functionality of propagation and conflict analysis
- Programmatic callbacks analyze the partial assignment

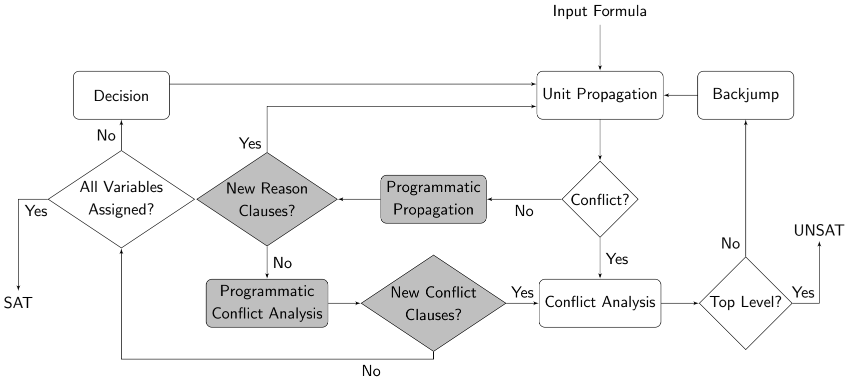
Programmatic SAT

- Instrumenting a SAT solver with *callbacks*
- Extending functionality of propagation and conflict analysis
- Programmatic callbacks analyze the partial assignment
- Propagation callback
  - Called after unit propagation
  - Checks for implied literals that are missed by unit propagation

Programmatic SAT

- Instrumenting a SAT solver with *callbacks*
- Extending functionality of propagation and conflict analysis
- Programmatic callbacks analyze the partial assignment
- Propagation callback
  - Called after unit propagation
  - Checks for implied literals that are missed by unit propagation
- Conflict analysis callback
  - Called after *propagation* is done
  - Checks if partial assignment cannot be extended to a full solution

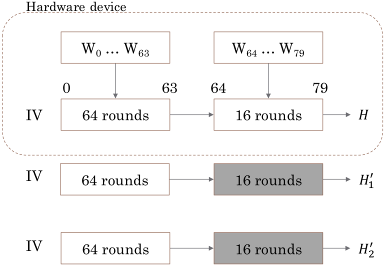
Programmatic SAT



Case Studies

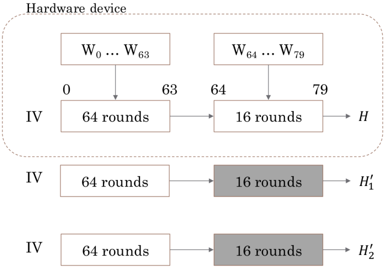
Hardware Fault Injection

- SHA-1 hash function



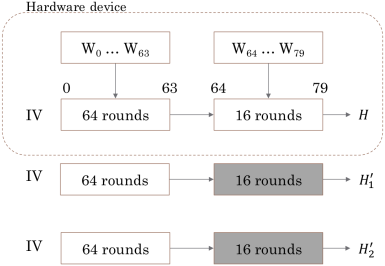
Hardware Fault Injection

- SHA-1 hash function
- Induce a fault in a target register



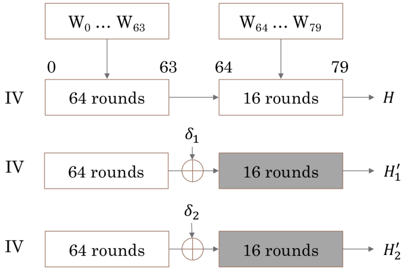
Hardware Fault Injection

- SHA-1 hash function
- Induce a fault in a target register
- Using heat, EM, laser, ...



Algebraic Fault Analysis

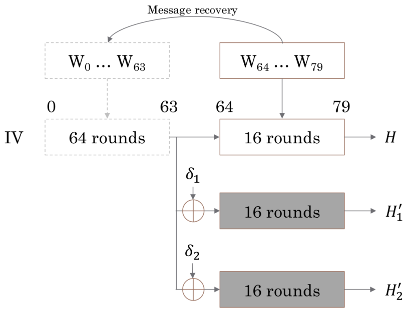
- $H = f_{0..79}(IV, W_{0..79})$
- This equation system will be encoded into CNF.
- Fault model: Constraints on δ_i.



$$H'_i = f_{64..79}(f_{0..63}(IV, W_{0..63}) \oplus \delta_i, W_{64..79})$$

Algebraic Fault Analysis

- Abstract away the common parts
- Verification of the solution will be needed

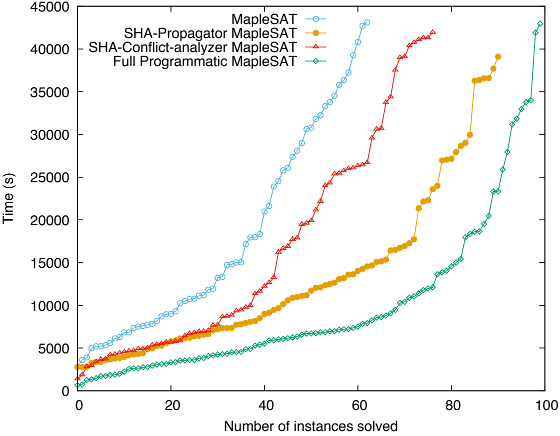


Algebraic Fault Analysis - Programmatic Approach

- Base SAT solver: MapleSAT
- Programmatic conflict analyzer
  - Embedding the verification loop
  - As soon as message word variables are set, they are ready to be verified
  - Early embedded check vs. Straightforward check after solving completely
- Programmatic propagator
  - Improving the propagation flow of multi-operand additions
  - Generating *reason clauses* in each column addition when output bits are missed

Algebraic Fault Analysis - Results

- Recovering SHA-256 message bits
- 14.3x speed-up on average
- 17 fewer faults were needed compared to the previous works



Differential Cryptanalysis

- Analyzing how a difference at the input propagates to a difference at the output.
- $\Delta x = x \oplus x' \rightarrow \Delta y = y \oplus y'$

Differential Cryptanalysis

- Analyzing how a difference at the input propagates to a difference at the output.
- $\Delta x = x \oplus x' \to \Delta y = y \oplus y'$
- Gathering statistical information about the differences (finding bias/non-randomness).

Differential Cryptanalysis

- Analyzing how a difference at the input propagates to a difference at the output.
- $\Delta x = x \oplus x' \to \Delta y = y \oplus y'$
- Gathering statistical information about the differences (finding bias/non-randomness).
- Differential Path: A trace of differentials over smaller steps in the function

Differential Cryptanalysis

- Analyzing how a difference at the input propagates to a difference at the output.
- $\Delta x = x \oplus x' \rightarrow \Delta y = y \oplus y'$
- Gathering statistical information about the differences (finding bias/non-randomness).
- Differential Path: A trace of differentials over smaller steps in the function
- Collision: a differential path with final difference equal to zero.

- Guess-and-determine solvers:
    - Very similar search approach to SAT solvers
    - Dedicated propagators for differential propagation rules
    - Dedicated branching heuristics
    - State-of-the-art results on SHA-256 collision (31 steps)

Differential Cryptanalysis - Collision on SHA-256

- Guess-and-determine solvers:
  - Very similar search approach to SAT solvers
  - Dedicated propagators for differential propagation rules
  - Dedicated branching heuristics
  - State-of-the-art results on SHA-256 collision (31 steps)
- SAT-based approaches:
  - Encoding bitwise differential behaviour only
  - Prioritizing difference variables
  - Limited representation power hence limited propagation power
  - State-of-the-art: 24 steps

Differential Cryptanalysis - Collision on SHA-256

- Guess-and-determine solvers:
  - Very similar search approach to SAT solvers
  - Dedicated propagators for differential propagation rules
  - Dedicated branching heuristics
  - State-of-the-art results on SHA-256 collision (31 steps)
- SAT-based approaches:
  - Encoding bitwise differential behaviour only
  - Prioritizing difference variables
  - Limited representation power hence limited propagation power
  - State-of-the-art: 24 steps

- Example rule: $r = IF(x, y, z)$, `-` $\leftarrow$ `---`, `x` $\leftarrow$ `-xx`.

Differential Cryptanalysis - Collision on SHA-256

- Guess-and-determine solvers:
  - Very similar search approach to SAT solvers
  - Dedicated propagators for differential propagation rules
  - Dedicated branching heuristics
  - State-of-the-art results on SHA-256 collision (31 steps)
- SAT-based approaches:
  - Encoding bitwise differential behaviour only
  - Prioritizing difference variables
  - Limited representation power hence limited propagation power
  - State-of-the-art: 24 steps

- Example rule: $r = IF(x, y, z)$, `-` $\leftarrow$ `---`, `x` $\leftarrow$ `-xx`.
- Full representation yields a blow up in size of the encoding
- An opportunity for Programmatic propagation

Differential Cryptanalysis - Results

- Collision on round-reduced SHA-256
- Modified the starting differential path of [Pro16]
- Base solver: MapleSAT
- Programmatic Propagator: Implemented a subset of differential propagation rules
- Programmatic conflict analyzer: Detects impossible differentials

- Found collision for 25 rounds of SHA-256 using MapleSAT(Crypto) in $\sim$3.5 hours

Conclusions

- A framework on top of CDCL SAT solvers to implement cryptographic reasonings.
- Showcased the power of the framework in two cryptanalysis tasks.
- A bridge between two ends of a spectrum:
    - Performance of dedicated cryptanalysis tools
    - Flexibility and search power of SAT solvers
- Beating state-of-the-art in some cases but still a long way to match state-of-the-art in other cases

Thanks!

Questions?

Glenn De Witte.
Automatic sat-solver based search tools for cryptanalysis.
2017.

Claudia Fiorini, Enrico Martinelli, and Fabio Massacci.
How to Fake an RSA Signature by Encoding Modular Root Finding as a SAT Problem.
Discrete Applied Mathematics, 130(2):101–127, 2003.

Fabio Massacci.
Using Walk-SAT and Rel-SAT for Cryptographic Key Search.
In *IJCAI*, volume 1999, pages 290–295, 1999.

Fabio Massacci and Laura Marraro.
Logical Cryptanalysis as a SAT Problem.
Journal of Automated Reasoning, 24(1-2):165–203, 2000.

Paweł Morawiecki and Marian Srebrny.
A SAT-based Preimage Analysis of Reduced KECCAK Hash Functions.
Information Processing Letters, 113(10):392–397, 2013.

Ilya Mironov and Lintao Zhang.
Applications of SAT Solvers to Cryptanalysis of Hash Functions.
Theory and Applications of Satisfiability Testing-SAT 2006, pages 102–115, 2006.

Vegard Nossum.
SAT-based Preimage Attacks on SHA-1.
2012.

Lukas Prokop.
Differential cryptanalysis with SAT solvers.
PhD thesis, University of Technology, Graz, 2016.

Aaron Tomb.
Applying Satisfiability to the Analysis of Cryptography.
https://github.com/GaloisInc/sat2015-crypto/blob/master/slides/talk.pdf, 2015.

CDCL(T): SAT-Modulo-Theories

- Satisfiability for higher level logical formulas

CDCL(T): SAT-Modulo-Theories

- Satisfiability for higher level logical formulas
- Abstract the given formula into a propositional one

CDCL(T): SAT-Modulo-Theories

- Satisfiability for higher level logical formulas
- Abstract the given formula into a propositional one
- $\underbrace{x^2 < 0}_{A} \vee \underbrace{x^2 > 1}_{B}$ becomes $A \vee B$.

CDCL(T): SAT-Modulo-Theories

- Satisfiability for higher level logical formulas
- Abstract the given formula into a propositional one
- $\underbrace{x^2 < 0}_{A} \vee \underbrace{x^2 > 1}_{B}$ becomes $A \vee B$.
- Solve it using a SAT solver (e.g. set A to true).

CDCL(T): SAT-Modulo-Theories

- Satisfiability for higher level logical formulas
- Abstract the given formula into a propositional one
- $\underbrace{x^2 < 0}_{A} \vee \underbrace{x^2 > 1}_{B}$ becomes $A \vee B$.
- Solve it using a SAT solver (e.g. set A to true).
- A *theory solver* checks if the assignment is a solution to the original formula (and if not, why not).

CDCL(T): SAT-Modulo-Theories

- Satisfiability for higher level logical formulas
- Abstract the given formula into a propositional one
- $\underbrace{x^2 < 0}_{A} \vee \underbrace{x^2 > 1}_{B}$ becomes $A \vee B$.
- Solve it using a SAT solver (e.g. set A to true).
- A *theory solver* checks if the assignment is a solution to the original formula (and if not, why not).
- Here the T-solver can return the clause $\neg A$ (i.e. $x^2 \geq 0$).