SAT-based Cryptanalysis of Cryptographic Hash Functions

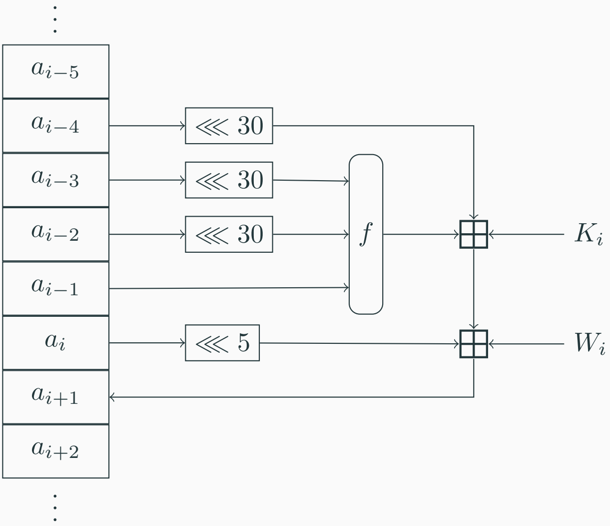
Saeed Nejati
Amazon Web Services

Simons Institute, February 24, 2021

Problem statement

- **Algebraic Fault Attack on SHA-1 function**: Find an input message given the hash output and extra constraints extracted from hardware implementation of the hash
- Core problem:
  - **Preimage attack**: Given H, find x such that $Hash(x) = H$.
  - Encode SHA-1 function in CNF, fix the output variables to H

SHA-1's Step Function



a_i, K_i, W_i: 32-bit vectors; f: 32-bit bitwise function

SHA-1 Bitwise function

$$f(x,y,z) = \begin{cases} IF(x,y,z) = (x \wedge y) \vee (\neg x \wedge z), & 0 \leq r \leq 19 \\ XOR(x,y,z) = x \oplus y \oplus z, & 20 \leq r \leq 39 \\ MAJ(x,y,z) = (x \wedge y) \vee (x \wedge z) \vee (y \wedge z), & 40 \leq r \leq 59 \\ XOR(x,y,z) = x \oplus y \oplus z, & 60 \leq r \leq 79 \end{cases}$$

Encoding Basic Blocks in CNF

$$f = IF(x, y, z) \equiv \bigwedge_{i=0}^{31} f_i \leftrightarrow (x_i \wedge y_i) \vee (\neg x_i \wedge z_i)$$

Encoding Basic Blocks in CNF

$$
\begin{aligned}
f = IF(x, y, z) &\equiv \bigwedge_{i=0}^{31} f_i \leftrightarrow (x_i \wedge y_i) \vee (\neg x_i \wedge z_i) \\
&\equiv \bigwedge_{i=0}^{31} (f_i \vee \neg x_i \neg y_i) \wedge (f_i \vee x_i \vee \neg z_i) \wedge \\
&\qquad (\neg f_i \vee x_i \vee z_i) \wedge (\neg f_i \vee \neg x_i \vee y_i)
\end{aligned}
$$

Encoding Basic Blocks in CNF

$$
\begin{aligned}
f = XOR(x,y,z) &\equiv \bigwedge_{i=0}^{31} f_i \leftrightarrow (x_i \oplus y_i \oplus z_i) \\
&\equiv \bigwedge_{i=0}^{31} (\neg f_i \vee \neg x_i \vee \neg y_i \vee z_i) \wedge (\neg f_i \vee \neg x_i \vee y_i \vee \neg z_i) \wedge \\
&\qquad (\neg f_i \vee x_i \vee \neg y_i \vee \neg z_i) \wedge (\neg f_i \vee x_i \vee y_i \vee z_i) \wedge \\
&\qquad (f_i \vee \neg x_i \vee \neg y_i \vee \neg z_i) \wedge (f_i \vee \neg x_i \vee y_i \vee z_i) \wedge \\
&\qquad (f_i \vee x_i \vee \neg y_i \vee z_i) \wedge (f_i \vee x_i \vee y_i \vee \neg z_i)
\end{aligned}
$$

Encoding Basic Blocks in CNF

$$
\begin{aligned}
f = MAJ(x,y,z) &\equiv \bigwedge_{i=0}^{31} f_i \leftrightarrow (x_i \wedge y_i) \vee (x_i \wedge z_i) \vee (y_i \wedge z_i) \\
&\equiv \bigwedge_{i=0}^{31} (\neg f_i \vee x_i \vee y_i) \wedge (\neg f_i \vee x_i \vee z_i) \wedge (\neg f_i \vee y_i \vee z_i) \wedge \\
&\qquad (f_i \vee \neg y_i \vee \neg z_i) \wedge (f_i \vee \neg x_i \vee \neg z_i) \wedge (f_i \vee \neg x_i \vee \neg y_i)
\end{aligned}
$$

Multi-operand modular addition

$$a_{i+1} = (a_i \lll 5) \boxplus W_i \boxplus K_i \boxplus f(a_{i-1}, a_{i-2}, a_{i-3}) \boxplus a_{i-4}$$

[Nos12]: Nossum, SAT-based Preimage Attacks on SHA-1

Multi-operand modular addition

$$a_{i+1} = (a_i \lll 5) \boxplus W_i \boxplus K_i \boxplus f(a_{i-1}, a_{i-2}, a_{i-3}) \boxplus a_{i-4}$$

$$\begin{array}{ccccc}
c'_2 & c'_1 & c'_0 & & \\
 & c_2 & c_1 & c_0 & \\
\hline
 & x_3 & x_2 & x_1 & x_0 \\
 & y_3 & y_2 & y_1 & y_0 \\
 & z_3 & z_2 & z_1 & z_0 \\
 & t_3 & t_2 & t_1 & t_0 \\
+ & w_3 & w_2 & w_1 & w_0 \\
\hline
= & s_3 & s_2 & s_1 & s_0
\end{array}$$

[Nos12]: Nossum, SAT-based Preimage Attacks on SHA-1

Multi-operand modular addition

$$a_{i+1} = (a_i \lll 5) \boxplus W_i \boxplus K_i \boxplus f(a_{i-1}, a_{i-2}, a_{i-3}) \boxplus a_{i-4}$$

$$\begin{array}{ccccc}
c'_2 & c'_1 & c'_0 & & \\
 & c_2 & c_1 & c_0 & \\
\hline
 & x_3 & x_2 & x_1 & x_0 \\
 & y_3 & y_2 & y_1 & y_0 \\
 & z_3 & z_2 & z_1 & z_0 \\
 & t_3 & t_2 & t_1 & t_0 \\
+ & w_3 & w_2 & w_1 & w_0 \\
\hline
= & s_3 & s_2 & s_1 & s_0
\end{array}$$

- 7-to-3 counter

[Nos12]: Nossum, SAT-based Preimage Attacks on SHA-1

Multi-operand modular addition

$$a_{i+1} = (a_i \lll 5) \boxplus W_i \boxplus K_i \boxplus f(a_{i-1}, a_{i-2}, a_{i-3}) \boxplus a_{i-4}$$

$$\begin{array}{ccccc}
c'_2 & c'_1 & c'_0 & & \\
 & c_2 & c_1 & c_0 & \\
\hline
 & x_3 & x_2 & x_1 & x_0 \\
 & y_3 & y_2 & y_1 & y_0 \\
 & z_3 & z_2 & z_1 & z_0 \\
 & t_3 & t_2 & t_1 & t_0 \\
+ & w_3 & w_2 & w_1 & w_0 \\
\hline
= & s_3 & s_2 & s_1 & s_0
\end{array}$$

- 7-to-3 counter
- espresso

[Nos12]: Nossum, SAT-based Preimage Attacks on SHA-1

Lost in Translation

- Observation in Nossum's encoding: Setting all of the input bits, does not unit propagate to all of the output bits!

Lost in Translation

- Observation in Nossum's encoding: Setting all of the input bits, does not unit propagate to all of the output bits!
- CSP literature: It means that the encoding is not arc-consistent

- Observation in Nossum's encoding: Setting all of the input bits, does not unit propagate to all of the output bits!
- CSP literature: It means that the encoding is not arc-consistent

- Example: consider a pseudo-Boolean constraint
  $C : x + y \leq 0, (x, y \in \{0, 1\})$

Lost in Translation

- Observation in Nossum's encoding: Setting all of the input bits, does not unit propagate to all of the output bits!
- CSP literature: It means that the encoding is not arc-consistent

- Example: consider a pseudo-Boolean constraint $C : x + y \leq 0, (x, y \in \{0, 1\})$
  - We trivially know: $C \to \bar{x}$ and $C \to \bar{y}$.

Lost in Translation

- Observation in Nossum's encoding: Setting all of the input bits, does not unit propagate to all of the output bits!
- CSP literature: It means that the encoding is not arc-consistent

- Example: consider a pseudo-Boolean constraint $C : x + y \leq 0, (x, y \in \{0, 1\})$
  - We trivially know: $C \to \bar{x}$ and $C \to \bar{y}$.
  - We can encode it using a half-adder

Lost in Translation

- Observation in Nossum's encoding: Setting all of the input bits, does not unit propagate to all of the output bits!
- CSP literature: It means that the encoding is not arc-consistent

- Example: consider a pseudo-Boolean constraint $C : x + y \leq 0, (x, y \in \{0, 1\})$
  - We trivially know: $C \rightarrow \bar{x}$ and $C \rightarrow \bar{y}$.
  - 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

- Observation in Nossum's encoding: Setting all of the input bits, does not unit propagate to all of the output bits!
- CSP literature: It means that the encoding is not arc-consistent

- Example: consider a pseudo-Boolean constraint $C : x + y \leq 0, (x, y \in \{0, 1\})$
  - We trivially know: $C \rightarrow \bar{x}$ and $C \rightarrow \bar{y}$.
  - 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

- Observation in Nossum's encoding: Setting all of the input bits, does not unit propagate to all of the output bits!
- CSP literature: It means that the encoding is not arc-consistent

- Example: consider a pseudo-Boolean constraint $C : x + y \leq 0, (x, y \in \{0, 1\})$
  - We trivially know: $C \rightarrow \bar{x}$ and $C \rightarrow \bar{y}$.
  - 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!

Encoding and Propagation

$$\text{Small Size} \xleftrightarrow{\text{Encoding}} \begin{array}{c}\text{Good for}\\ \text{Unit Progpagation}\end{array}$$

Encoding and Propagation

$$\text{Small Size} \xleftrightarrow{\hspace{6em}\text{Encoding}\hspace{6em}} \begin{array}{c}\text{Good for}\\ \text{Unit Progpagation}\end{array}$$

- Ideal: Having "good" propagation while keeping the encoding small

Encoding and Propagation

$$\text{Small Size} \xleftrightarrow{\text{Encoding}} \begin{array}{c}\text{Good for} \\ \text{Unit Progpagation}\end{array}$$

- Ideal: Having "good" propagation while keeping the encoding small
- Extending propagation programmatically

Encoding and Propagation

$$\text{Small Size} \xleftrightarrow{\qquad\qquad\text{Encoding}\qquad\qquad} \begin{array}{c}\text{Good for}\\ \text{Unit Progpagation}\end{array}$$

- Ideal: Having "good" propagation while keeping the encoding small
- Extending propagation programmatically
- Using Programmatic SAT architecture (inspired by the CDCL(T) paradigm)

[GOS$^+$12]: Ganesh et al., Lynx: A Programmatic SAT Solver for the RNA-Folding Problem, SAT12

Encoding and Propagation

$$\text{Small Size} \xleftrightarrow{\text{Encoding}} \begin{array}{c}\text{Good for}\\ \text{Unit Progpagation}\end{array}$$

- Ideal: Having "good" propagation while keeping the encoding small
- Extending propagation programmatically
- Using Programmatic SAT architecture (inspired by the CDCL(T) paradigm)
- Propagation callback:
  - Analyze the partial assignment
  - Add reason clauses for missed implications

[GOS+12]: Ganesh et al., Lynx: A Programmatic SAT Solver for the RNA-Folding Problem, SAT12

Encoding and Propagation

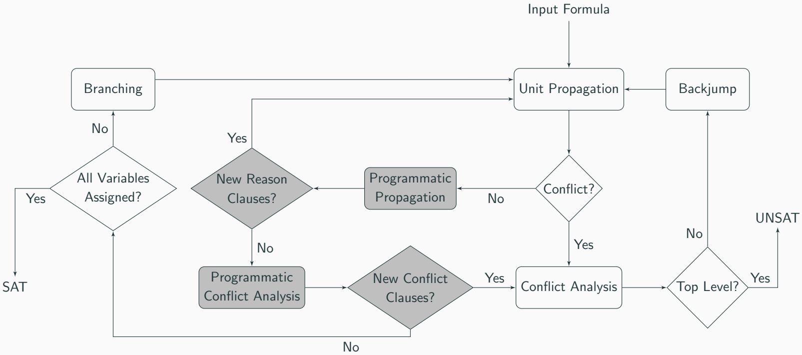
Small Size $\longleftrightarrow$ Good for Unit Progpagation

Encoding

- Ideal: Having "good" propagation while keeping the encoding small
- Extending propagation programmatically
- Using Programmatic SAT architecture (inspired by the CDCL(T) paradigm)
- Propagation callback:
  - Analyze the partial assignment
  - Add reason clauses for missed implications
- Encoding the problem in CNF+C

[GOS+12]: Ganesh et al., Lynx: A Programmatic SAT Solver for the RNA-Folding Problem, SAT12

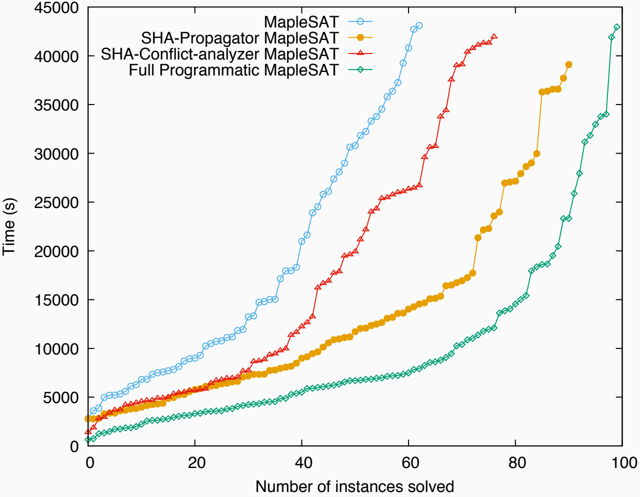
Programmatic SAT



I/O Arc-consistent

- We added reason clauses for input/output relations
- Unit+Programmatic propagation: input/output arc-consistent
- Also called "propagation complete" in some papers

Results



[NHGG18]: Nejati et al., Algebraic fault attack on SHA hash functions using programmatic SAT solvers, CP18

Thanks!

Questions?

Vijay Ganesh, Charles W. O'Donnell, Mate Soos, Srinivas Devadas, Martin C. Rinard, and Armando Solar-Lezama.
Lynx: A programmatic SAT solver for the RNA-folding problem.
In *Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings*, pages 143–156, 2012.

Saeed Nejati and Vijay Ganesh.
Cdcl (crypto) sat solvers for cryptanalysis.
In *Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering*, pages 311–316. IBM Corp., 2019.

Saeed Nejati, Jan Horáček, Catherine Gebotys, and Vijay Ganesh.
Algebraic fault attack on sha hash functions using programmatic sat solvers.
In *International Conference on Principles and Practice of Constraint Programming*, pages 737–754. Springer, Cham, 2018.

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