Boolean Cryptanalysis: Breaking Bivium

Shravani Shahapure[1] Virendra Sule[2]

[1]Data Security Council of India (DSCI)
New Delhi, India
(shravani.shahapure@dsci.in)
[2]Department of Electrical Engineering
Indian Institute of Technology Bombay, India
(vrs@ee.iitb.ac.in)

Industrial Computeralgebra Conference with focus
Cryptography, September 29, 2021
Carl von Ossietzky University, Oldenburg, Germany

What is Boolean Cryptanalysis

- **Algebraic Cryptanalysis:** Solving multivariate polynomial equations arising in the Cryptanalysis of Block and stream ciphers using algebraic methods such as Grobner basis, Extended linearization (XL) and their combinations.
- **Boolean Cryptanalysis:** Only for Boolean systems of equations. Treating the equations in Boolean functions instead of polynomials. Using Boolean solution methods.
- **Boolean solution methods:** SAT. Not suitable for Cryptanalysis becuase SAT solvers mainly solve the decision problem of existence. Produce only one certificate (one solution). In Cryptanalysis existence is already known. Due to insufficient data it is necessary to find all solutions to confirm the key with extra encryptions.
- **Boolean elimination:** Different from algebraic elimination. Requires symbolic Boolean computation. Not explored much from the point of view of scalability [2, 1].

Boolean Cryptanalysis by Implicant based solver

- Implicant based Boolean solver: Implicant based, parallel, all solution solver [4]. Represents all satisfying assignments in terms of a set of *orthogonal implicants* of the system of equations.
- Inherently parallel method. Does not require backtracking. Starts with a small set of feasible solutions containing all partial satisfying assignments. Hence theoretically better than brute force search. Several advantages in scalability of the method for realistic size Cryptanalysis as shown in case study [3].
- Symbolic Boolean model: Boolean system with smallest number of unknown variables. MQ Algebraic system for $AES128$ more than 4000 variables and 9000 equations. The Boolean model in 256 input variables and 128 equations.

Orthogonal Boolean functions

- $B_0(n)$ the set of all Boolean functions in n-Boolean variables $X = \{x_1, \ldots, x_n\}$ taking values in the two element Boolean algebra $B_0 = \{0, 1, +, ., ' \}$, $+$ denotes OR, . denotes AND, $'$ denotes complement. The associated Boolean ring is $\{0, 1, \oplus, .\}$ is same as the field $\mathbb{F}_2$. $(a \oplus b = ab' + a'b)$.
- $B_0(n)$ is itself a Boolean algebra using the Boolean algebra structure of its values B_0. Hence $f + g = f(X) + g(X)$, $fg = f(X)g(X)$ and $f' = f(X)'$ for all assignments of X, are defined naturally so as the inequalities

$$f \leq g \Leftrightarrow f(X) \leq g(X) \Leftrightarrow f(X)g(X)' = 0$$

for all assignments of X. The XOR operation is also defined by $f \oplus g = f(X) \oplus g(X)$ and $f' = f(X) \oplus 1$.

Implicants of functions

- A set of non-zero Boolean functions $\{f_1, \ldots, f_m\}$ is said to be *orthogonal* (OG) in $B_0(n)$ if $f_i f_j = 0$ for $i \neq j$ and is called *orthonormal* (ON) if it is OG and satisfies

$$\sum_i f_i = f_1 + \ldots + f_m = 1$$

- Terms t in $B_0(n)$ are elementary conjunctions such as for example $x_3 x_6' x_7 x_9'$ and are also called product of literals.
- A term $t(X)$ defined over variables X is called an *implicant* of a function $f(X)$ if $t(a) = 1$ for an assignment $a = (a_1, \ldots, a_n)$ for x where a_i are 0 or 1, then $f(x = a) = 1$.
- A set of implicants $T = \{t_1, t_2, \ldots, t_r\}$ of a function $f(X)$ is said to be *complete* if given $f(a) = 1$ for an assignment $x = a$ there is at least one implicant t_i in T such The $t_i(a) = 1$.

Representation of f in OG implicants I

- Given any complete set T of implicants of a function f, the function itself has a representation

$$f(X) = \sum_{t_i \in T} t_i(X)$$

- If T is an OG complete set of implicants of f then

$$f(X) = \bigoplus_{t_i \in T} t_i(X)$$

is an SOP representation of f in OG implicants. Unique only when T consists of minterms in variables X of f.

Representation of f in OG implicants II

- If $t(a) = 1$ for an implicant then a gives a partial assignment of variables arising as literals in t hence when t belongs to a complete set of implicants of f then $f(a) = 1$ iff $t(a) = 1$ for some t in T.
- This gives rise to a representation of all solutions of the Boolean equation $f(X) = 1$ (or all *satisfying assignments* of f as partial assignments) in terms of implicants

$$S(f) = \bigsqcup_{t_i \in T} \{a \in B_0^n | t_i(a) = 1\}$$

- this representation is compact because each implicant shows only partial assignments, it does not consider free assignments which are logical dont cares.

Examples of representation and all solutions

- In variables $X = \{x, y, z\}$, $\{xz, x'y, x'z'\}$ is an OG set of terms, $\{y', x'y, xyz', xyz\}$ is ON set as the terms are OG and

$$y' + x'y + xyz' + xyz = y' + x'y + xy = y' + y = 1$$

  Minterms in X is an exponential size set of ON terms in X, while

$$\{x_1, x_1'x_2, x_1'x_2'x_3, \ldots, x_1'x_2' \ldots x_{n-1}'x_n, x_1'x_2' \ldots x_{n-1}'x_n'\}$$

  is an ON set in $X = \{x_1, \ldots, x_n\}$ of polynomial size $(n+1)$.
- If $T = \{xy'z, xy, x'z'\}$ is a complete set of implicants of f which is OG, then the set of all satisfying assignments of f, (all solutions of $f(X) = 1$ are represented by the set T in 3 partial assignments

$$\{(1, 0, 1), (1, 1, *), (0, *, 0)\}$$

  where $*$ are dont cares denoting free assignments.

Computation of OG complete set of implicants $I(f)$ of one function

- The set of all minterms in n-variables X, $m(X) = \{X^A | A \subset \{0,1\}^n\}$ is an ON set.
- If $f(X)$ is a function over variables X the set of all $t \in m(X)$ such that $f/t = 1$ is an ON complete set of implicants of f.
- Example: Consider $f(x,y,z) = 1 \oplus x \oplus y \oplus xz \oplus xyz$. Then an ON complete set of implicants can be written down in minterms in x, y, z by brute force search. These are all minterms m such that $f/m = 1$

$$I(f) = \{x'yz, xy'z, xyz', x'y'z'\}$$

- Another alternative is to write implicants in t_i where t_i are homogeneous polynomials of degree i such that $f = 1 \oplus t_1 \oplus t_2 \oplus t_3$ for above example.
- A procedure to compute $I(f) = \text{Implicant}(f)$ is used in the algorithm.

Simultaneous satisfying assignments in terms of implicants of two functions

- Representation of satisfying assignments $S(f)$ by implicants is compact since free variables are ignored. With OG property the set $S(f)$ is a disjoint union.
- Ratio of a Boolean function function and a term: Let t be a term in X and $f(X)$ a Boolean function, then f/t called the *ratio* of f by t is the Boolean function defined by $F = f(x)|_{(t(x)=1)}$ by substitution of partial assignments.

Lemma on simultaneous implicants

- **Lemma**: If f and g are two functions in $B_0(n)$ and $I(f)$, $I(g)$ denote complete sets of OG implicants of f and g respectively, then complete sets of OG implicants of the product $h = fg$ can be represented by two possible ways

$$\begin{aligned} I_1(h) &= \bigsqcup_{t \in I(f), s \in I(g/t)} \{ts\} \\ I_2(h) &= \bigsqcup_{s \in I(g), t \in I(f/s)} \{st\} \end{aligned}$$

where $I(g/t)$ (respectively $I(f/s)$) denote complete sets of OG implicants of g/t (respectively f/s).

Idea of the Implicant based solver

- Above lemma leads to an algorithm for computation of a complete OG set of implicants $I(F)$ of a product of Boolean functions

$$F = f_1 f_2 \dots f_m$$

  by recursively computing implicants of factors f_i.
- $I(F)$ represents all satisfying assignments of F compactly.
- Algorithm executes in parallel threads adding implicant terms to a starting set of implicants of the first factor. A thread resulting in a term t for which $f/t = 0$ is terminated. Longest thread gives time complexity. Number of threads give space complexity.

Boolean solver algorithm I

1: **procedure** IMPLICANT(S)
2: Input: Boolean system $S = \{f_1, \ldots, f_m\}$, $F = \prod f_i$
3: Output: A complete set of orthogonal implicants $I(S)$
4: $I(S) = \emptyset$
5: Select one pivot function f in S, $X = I(f)$.
6: % X a complete set of OG implicants of f computed by a
7: % subroutine
8: **repeat**
9: **for** t in X **do**
10: Compute $S/t = \{f_i/t\}$.
11: % Substitute t in the set S
12: **if** There is i such that $f_i/t = 0$ **then**
13: End the thread started with t.
14: Choose next t in X.

Boolean solver algorithm II

15: **else**
16: Reduce S/t by deleting $f_i/t = 1$.
17: Update $S \leftarrow S/t$ after deleting $f_i/t = 1$.
18: **end if**
19: **while** S has still nonzero functions left **do**
20: $I = \text{IMPLICANT}(S)$
21: % Recursive call to the procedure
22: Return $I(S) = I \cup \{t \times I\}$
23: % $\{t \times I\} = \{ts\}$ for all s in I
24: **end while**
25: **end for**
26: **until** All t in X are processed
27: **end procedure**

An example of solving implicants

- Notation: Boolean function in ANF $f(x_1, x_2, x_3)$

$f(x_1, x_2, x_3)$	Notation
$1 \oplus x_1 \oplus x_2 \oplus x_1x_3 \oplus x_1x_2x_3$	$[1, [1], [2], [13], [123]]$
$x_2 \oplus x_3 \oplus x_1x_2 \oplus x_2x_3$	$[[1], [3].[12], [13]]$

- Notation: Terms $t = x_1x_2'x_3'x_5$ as $(1 - 2 - 35)$, $s = x_1'x_3x_4'$ as $(-13 - 4)$
- Example: Find all satisfying implicants of the function $F = f_1f_2f_3$ where

$$\begin{aligned} f_1 &= [[1], [2], [12]] \\ f_2 &= [[2], [3], [23]] \\ f_3 &= [[1], [3], [13]] \end{aligned}$$

Illustrative example

- Brute force search requires search over 8 minterms.
- Compute implicants of f_1. Brute force search over 4 minterms in x_1, x_2. $I(f_1) = \{(12), (-12), (1-2)\}$. Hence 3 independent threads to search all solutions.
- Thread $t = (12)$: $f_2/t = 0$ end t cannot be a part of an implicant.
- Thread $t = (-12)$: $f/2 = 1, f_3/t = [3]$. Implicant $\{(-123)$.
- Thread $t = (1-2)$: $f_2/t = [3]$, $f_3/t = 1$. Implicant $(1-23)$.
- Hence $I(F) = \{(-123), (1-23)\}$

Heuristics for decomposition I

- The Boolean solver is inherently parallel as all threads are independent of eachother and hence allows decomposition of large systems S into smaller subsystems on which it can operate very efficiently. Especially when the factors are sparse in variables.
- Decomposition also allows control over memory required for independent parallel processing.
- Heuristic 1 (Independent implicants). Choose small subsets S_1, S_2 of factors in S which dont have overlapping variables. Compute $I(S_1)$, $I(S_2)$. Then

$$I(S_1 \cup S_2) = \{ts, t \in I(S_1), s \in I(S_2)\}$$

Substitution of implicants in remaining functions in S reduces number of variables and functions drastically. Search is then

Heuristics for decomposition II

over those implicants of $I(S_1 \cup I(S_2)\ldots)$ which do not result into $f/t = 0$.

- Heuristic 2 (Top ranked variables). Choose a small enough group X_1 of variables in X such that X_1 variables occur in a subset of S most often. Choose minterms $t \in M(X_1)$ in X_1. Substitution (brute force in variables X_1) of minterms cause drastic reduction and increased sparsity of S/t. Implicants of S are obtained as ts where $s \in I(S/t)$.
- Complexity of the algorithm is in general exponential. However the algorithm works efficiently if the system S is sparse, number of variables ν per factor on average is a small percent of total number of variables n. For computation of $I(f)$ of a pivot the maximum computation is of order $O(2^{\nu})$. Assume this as a constant since for sparse systems $\nu << n$.

Heuristics for decomposition III

- If memory is allotted for all parallel threads, longest thread has m steps where m is the number of equations. At least one variable assignment is fixed in one step by solving an implicant of the pivot, free variables are not discovered explicitly. Hence the time complexity is $O(m)$ under complete parallel computation with sparseness assumption.
- Detailed analysis of complexity is still an unfinished problem. The number of threads initially increase exponentially and half way they decrease exponentially finally giving the implicant set. Extent of parallelism reduces the longest thread time.

Bivium: Symbolic Boolean model I

- Bivium belongs to the class of stream ciphers which have following model, x denotes the vector of state variables in $\mathbb{F}_2^{177}$, w denotes the binary output:

$$\begin{array}{lllll} \text{State update map} & x(k+1) & = & F(x(k)) \\ \text{Output function} & w(k) & = & f(x(k)) \end{array}$$

- Initial condition $x(0)$ consists of, symmetric key K first 80 bits $x_1, \ldots, x80$, IV next 80 bits $x_{81}, \ldots, x_{160}$, last bits $x_{161}, \ldots, x_{177}$ set to zero.
- Output stream $w(k)$ used for encryption for $k \geq 709$.

Bivium: Symbolic Boolean model II

- State update map F:

$$\begin{aligned}
x_1(k+1) &= x_{162}(k) \oplus x_{177}(k) \oplus x_{175}(k)x_{176}(k) \oplus x_{69}(k) \\
x_2(k+1) &= x_1(k) \\
\vdots &= \vdots \\
x_{93}(k+1) &= x_{92}(k) \\
x_{94}(k+1) &= x_{66}(k) \oplus x_{93}(k) \oplus x_{91}(k)x_{92}(k) \oplus x_{171}(k) \\
x_{95}(k+1) &= x_{94}(k) \\
\vdots &= \vdots \\
x_{177}(k+1) &= x_{176}(k)
\end{aligned} \tag{1}$$

- Output function f:

$$w(k) = x_{66}(k) \oplus x_{93}(k) \oplus x_{162}(k) \oplus x_{177} \tag{2}$$

Bivium: Symbolic Boolean model III

- Inverse of state update map $G = F^{-1}$: $x(k) = G(x(k+1))$

$$
\begin{array}{lcl}
x_1(k) & = & x_2(k+1) \\
\vdots & = & \vdots \\
x_{92}(k) & = & x_{93}(k+1) \\
x_{93}(k) & = & x_{94}(k+1) \oplus x_{66}(k+1) \oplus \\
 & & x_{91}(k+1)x_{92}(k+1) \oplus x_{177}(k+1) \\
x_{94}(k) & = & x_{95}(k+1) \\
\vdots & = & \vdots \\
x_{176}(k) & = & x_{177}(k+1) \\
x_{177}(k) & = & x_1(k+1) \oplus x_{162}(k+1) \oplus \\
 & & x_{175}(k+1)x_{176}(k+1) \oplus x_{69}(k+1)
\end{array}
\tag{3}
$$

Cryptanalysis: Internal state recovery I

- Using the inverse map G initial condition can be computed from internal state at any future time as fast as forward computation.
- Cryptanalysis was performed by solving the internal states $x(709)$ from a given output stream $w(709), w(710), \ldots, w(885)$ using 177 equations.
- The Boolean equations are

$$\begin{aligned} w(709) &= f(x(709)) \\ w(k) &= F^*(f(x(k-1)) = f(F(x(k-1))) \text{ for } k \geq 710 \end{aligned}$$

Cryptanalysis: Internal state recovery II

- First 15 equations are linear each with 4 variables and each have non-overlapping variables. Hence implicants of any collection of these are just products of implicants of each function. Heuristic 1 is applied for searching over such implicant sets to reduce the remaining system. Results in Table 1.
- 44 top ranked variables were selected and search over minterms in these variables by substitution in all equations was performed to solve the reduced systems. Results in Table 2.

Performance results I

Table: Implicants with non-overlapping equations

# eqns.	# var	# Impl	Av. Time of thread	TB
13	52	$8^{13} = 2^{39}$	$>$ 2 days	3.95
14	56	$8^{14} = 2^{42}$	61503 Sec.	30.68
15	60	$8^{15} = 2^{45}$	4926 Sec.	261.16

Table: Brute force assignment of top ranked variables each thread parallely solved

# top ranked variables	Av. Time taken to solve	Memory in terabytes
40	49.4 Hrs.	5.4
44	28 Hrs.	96.7
45	14 Hrs.	197.9
48	1.8 Hrs.	$>$ 1000

Conclusions

- The Implicant based Boolean solver successfully solved Boolean equations of Bivium in 177 variables with Heuristic 2 in practically feasible time and memory.
- This solution results in breaking of Bivium with 80 bits key without brute force search. Boolean solver gives an alternate Boolean cryptanalysis methodology than Algebraic Cryptanalysis.
- One of the poorly researched aspect of Boolean Cryptanalysis is computation of Boolean Symbolic model in smallest number of variables. If such a model can be built efficiently for $AES128$, it can be represented by 128 Boolean equations in 256 variables. Such a system may be solvable by Boolean solver in known 128 plaintext bits in practically feasible time and memory.

References

[1] Frank M. Brown. *Boolean Reasoning, The Logic of Boolean Equations, 2nd Edition*. Dover Publications, New York, 2003.

[2] Yves Crama and Peter Hammer (Eds). *Boolean functions. Theory, algorithms and applications. Encyclopedia of Mathematics and its applications*, volume 142. Cambridge University Press, 2011.

[3] Shravani Shahapure, Virendra Sule, and R.D̃. Daruwala. Internal state recovery attack on stream ciphers: Breaking bivium. *S. Bhasin et al. (Eds): SPACE 2019, Lecture Notes in Computer Science*, 11947:1–16, December 2019.

[4] Virendra Sule. An implicant based, parallel, all solution solver for boolean satisfiability. *arxiv.org/cs.DS/1611.09590v3*, March 2017.

Thank you

Thank You