Des. Codes Cryptogr.
DOI 10.1007/s10623-010-9465-x

Sparse Boolean equations and circuit lattices

Igor Semaev

Received: 10 February 2009 / Revised: 17 April 2009 / Accepted: 1 August 2010

Abstract A system of Boolean equations is called sparse if each equation depends on a small number of variables. Finding efficiently solutions to the system is an underlying hard problem in the cryptanalysis of modern ciphers. In this paper we study new properties of the Agreeing Algorithm, which was earlier designed to solve such equations. Then we show that mathematical description of the Algorithm is translated straight into the language of electric wires and switches. Applications to the DES and the Triple DES are discussed. The new approach, at least theoretically, allows a faster key-rejecting in brute-force than with COPACOBANA.

Keywords Sparse Boolean equations · Equation graph · Electrical circuits · Switches

Mathematics Subject Classification (2000) 94A60 · 68Q25 · 94C15 · 94C10

1 Introduction

Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of Boolean variables. By $X_i,\ 1 \le i \le m$ we denote subsets of X of size $l_i \le l$. The system of equations

$$f_1(X_1) = 0, \ldots, f_m(X_m) = 0 \tag{1}$$

is considered, where f_i are Boolean functions (polynomials in algebraic normal form) and they only depend on variables X_i. Such equations are called l-sparse. We look for the set of all 0, 1-solutions to (1). Obviously, the equation $f_i(X_i) = 0$ is determined by the pair $E_i = (X_i, V_i)$, where V_i is the set of 0, 1-vectors in variables X_i, also called X_i-vectors, where f_i is zero. In other words, V_i is the set of all solutions to $f_i = 0$. The function f_i is uniquely defined by V_i. Given f_i, the set V_i is computed with 2^{l_i} trials.

I. Semaev (✉)
Department of Informatics, University of Bergen, Bergen, Norway
e-mail: igor@ii.uib.no; igor.semaev@ii.uib.no

Published online: 01 January 2011

Table 1 Algorithms' running time

l	3	4	5	6
The worst case [12]	1.324^n	1.474^n	1.569^n	1.637^n
Gluing1, expectation [18]	1.262^n	1.355^n	1.425^n	1.479^n
Gluing2, expectation [18]	1.238^n	1.326^n	1.393^n	1.446^n
Agreeing-Gluing1, expectation [19]	1.113^n	1.205^n	1.276^n	1.334^n
Weak IAG, expectation [20]	1.029^n	1.107^n	1.182^n	1.239^n

In [15] Agreeing and Gluing procedures were described. Then they were combined with variables guessing to solve (1). See also related earlier work [23]. Assume uniform distribution on instances (1). Table 1 summarizes expected complexity estimates for simple combinations of the Agreeing and Gluing in case of $m = n$ and a variety of l. Each instance of (1) may be encoded by a CNF formula with clause length l in the same variables. So l-SAT solving algorithms provide with worst case complexity estimates. The table data suggests that Agreeing-Gluing based methods should be very fast in practice. This is the reason why a hardware implementation of the Agreeing Algorithm is here proposed. In spite of relatively high worst case bound on l-SAT problem complexity, there exist a number of efficient l-SAT solvers. They became useful tools in cryptanalysis [4,5]. However, an efficient hardware version of the approach is still unknown.

Conjectured asymptotic bounds on the complexity of the popular Gröbner Basis Algorithm and its variants as XL, see [3,8], are found in [1,21]. They are far worse than the estimates by the brute force approach except for quadratic and very over-defined equation system. It was found in [16] that a linear algebra variant (called MRHS) of the Agreeing-Gluing significantly overcomes (on AES type Boolean equations in around 50 variables) the F4 method, a Gröbner Basis Algorithm implemented in MAGMA.

We first study here a new property of the Agreeing Algorithm. This algorithm implements pairwise simplification to the initial equations after some suitable guess. We will show that the result only depends on a smaller subset of equation pairs. This significantly reduces memory requirements for the Agreeing Algorithm. For example, for DES instead of 3,545 pairs, the algorithm should only run through 1,404 of them with the same output. In case of the Triple DES the figure is 3,929 instead of 16,831, see Table 2.

Then we suggest implementing the Agreeing Algorithm in hardware. The main features of the related device, called Circuit Lattice(CL), are:

- No memory locations are necessary as no one bit is kept by the device in common sense. Solutions to particular equations are circuits with two type of switches and the whole system is a network of connections between them represented as a circuit lattice. See Fig. 6 for instance.
- Voltage is induced by guessing variables. Its expansion is then directed by switches implemented as electronic relays or transistors on a semiconductor chip. The potential difference detected in some particular circuits indicates the system is inconsistent after the guess.
- The number of input contacts is essentially $2s$, where s is the number of variables guessed during the solution of the system. That is at most $2n$ anyway. Some power contacts and one output contact that sends out a signal when the system is found inconsistent should be added.

– The speed of the device is determined by the time of switching, where lots of switches turn simultaneously. Switches are not necessarily synchronized, so the device does not work as a conventional computer.

It is very unlikely to solve the system by Agreeing alone. So some guesses on the variable values should be made. The system is then checked for consistence with the Agreeing Algorithm. As most of the guesses should be incorrect, it is important to have an efficient way to check the system's inconsistence. The suggested Circuit Lattice is designed to achieve this goal. Implementing equations from a cipher, it may be used for a brute force attack. When trying the current key, one introduces the guess into the device, and checks whether the system is inconsistent.

Common approaches to the key search [2,6,7,13,14,17,22] are based on the parallelization of the job to many special purpose chips, which efficiently implement the encryption. The best reported speed for one DES encryption with COPACOBANA is about 0.1 GHz per chip, [13]. Approximately the same speed per each of its SPU is achieved by Cell Processor, see [14]. This results in about 0.034 GHz for the Triple DES. This is the key rejecting rate.

In contrast, our idea is to not implement any encryption. If constructed, the Circuit Lattice might achieve a higher key rejecting rate, see the discussion in Sect. 7. Moreover, depending on the equation system from the cipher, the number of key bits necessary to guess before solving or observing an inconsistence may vary. For instance, in [15] it was reported that 37–38 key variables out of 56 are guessed and the rest of the system from 6 rounds of the DES is solved by the Agreeing Algorithm alone. So it is sometimes not necessary to guess all key bits. There may exist a lot of equation systems describing one particular cipher produced, for instance, with the Gluing procedure. Our approach therefore has more flexibility.

It was also reported in [16] that admitting up to 2^s right hand sides (produced with Gluing during system solution) in MRHS equations for the AES-128, one should only guess 128-s of the key bits before the system is solved. A fast way, based on some physical principle, for checking the system's inconsistence after the guess might result in breaking a real world cipher. Two principles may be in use here: electric potential expansion and the expansion of light. We will presently follow the first principle.

This proposal is different from an independent work by Geiselmann et al., which describes a hardware implementation of main MRHS routines, see [11].

2 Agreeing procedure

For equations $E_1 = (X_1, V_1)$ and $E_2 = (X_2, V_2)$, let $X_{1,2} = X_1 \cap X_2$. Then let $V_{1,2}$ be the set of $X_{1,2}$-subvectors of V_1, that is the set of projections of V_1 to variables $X_{1,2}$. Similarly, the set $V_{2,1}$ of $X_{1,2}$-subvectors of V_2 is defined. We say the equations E_1 and E_2 agree if $V_{1,2} = V_{2,1}$. Otherwise, we apply the procedure called Agreeing. All vectors whose $X_{1,2}$-subvectors are not in $V_{2,1} \cap V_{1,2}$ are deleted from V_1 and V_2. Obviously, we delete V_i-vectors which can't make part of any common solution to the equations. Then we put $E_i \leftarrow (X_i, V_i')$, where $V_i' \subseteq V_i$ consist of the survived vectors.

2.1 Agreeing algorithm

The goal of the Agreeing Algorithm is to identify wrong solutions to equations E_i and remove them from V_i by pairwise application of the Agreeing Procedure. The output doesn't depend on the order of pairwise agreeings, see [16]. Application of the procedure to E_i and E_j where $X_i \cap X_j = \emptyset$ can be avoided. We will show that some pairs E_i, E_j can be avoided too even if

$X_i \cap X_j \neq \emptyset$. This significantly optimizes memory requirement of the Agreeing Algorithm and the hardware implementation described in Sect. 3.

The equations $E_1, \ldots, E_m$ are vertices in an equation graph G. Vertices E_i and E_j are connected by the edge (E_i, E_j) labeled with $X_{i,j} = X_i \cap X_j \neq \emptyset$. There may occur different edges with the same labels. The Agreeing Procedure, being applied to E_i and E_j, implements a kind of information exchange between them through the edge (E_i, E_j). That is for $Y \subseteq X_{i,j}$ the information $Y \neq a$ for some binary string a is transmitted from E_i to E_j or backwards. For simplicity, the same symbol Y also denotes an ordered string of variables Y. We will now show that some of the edges in the graph G are obsolete in this respect.

A subgraph G_m of G is called *minimal* if it is on the same vertices and

1. For any (E_i, E_j) in G, there exists a sequence of vertices
$$E_i, E_k, E_l, \ldots, E_r, E_j, \tag{2}$$
where $(E_i, E_k), (E_k, E_l), \ldots, (E_r, E_j)$ are in G_m and $X_{i,j}$ is a subset in each label $X_{i,k}, X_{k,l}, \ldots, X_{r,j}$.
2. G_m has minimal number of edges among subgraphs that satisfy property 1.

This definition is correct as G itself satisfies property 1. Also it implies that any minimal subgraph has the same number of edges. Therefore our goal is to find such a subgraph, see Algorithm and Lemma 2 below. In particular, the second statement of the Lemma says that any subgraph of G that satisfies property 1 has at least as many edges as the algorithm's output graph. The edges of a minimal subgraph are called *maximal* and denoted A for some fixed G_m. Minimal subgraphs are not uniquely defined.

Lemma 1 *The Agreeing Algorithm output doesn't depend on whether the Agreeing procedure runs through all edges of G or through only maximal edges.*

Proof Let $Y \subseteq X_{i,j}$ for the equations E_i and E_j. Assume we learn, from the equation E_i, that $Y \neq a$ for some string a. The Agreeing procedure expands $Y \neq a$ from E_i to E_j. That is all vectors in V_j whose projection to Y is a are wrong and should be removed from V_j. There exists a path (2) in the minimal graph G_m, where

$$Y \subseteq X_{i,j} \subseteq X_i, X_k, X_l, \ldots, X_r, X_j.$$

So $Y \neq a$ is expanded from E_i to E_j through the path (2) by first agreeing pairwise E_i, E_k. That is one removes all vectors in V_k whose projection to Y is a. Then one agrees E_k, E_l,..., and finally E_r, E_j. All V_j-vectors whose projection to Y is a are found wrong and removed. We therefore see that the results produced by agreeing through (E_i, E_j) and through the path (2) in G_m are the same. This proves the Lemma. □

We now formulate the algorithm to compute a minimal subgraph of G:

1. For every label $Y \subseteq X$ find all edges (E_s, E_r) in G such that $Y \subseteq X_{s,r}$. Denote a subgraph of G with all such edges (E_s, E_r) and related vertices by G_Y. We remark that G_Y is a complete graph.
2. Find the set V_Y of edges (E_s, E_r) in G_Y, where $X_{s,r} = Y$. Find a subset $W_Y \subseteq V_Y$ such that G_Y is still connected after removing the edges W_Y and W_Y has the largest number of edges. We remark that W_Y is not uniquely defined, and taking different W_Y produces different minimal subgraphs.
3. Remove the edges W_Y from G for all Y and get G_m.

Lemma 2 *Let G_m be the algorithm's output graph.*

1. *Then G_m satisfies property* 1 *from the minimal subgraph definition.*
2. *Any subgraph of G that satisfies property* 1 *has at least as many edges as G_m.*

Therefore G_m is minimal.

Proof We prove that for any edge (E_i, E_j) in G there is a path (2) on G_m. That will imply the first claim of the Lemma. Let $Y = X_{i,j}$. If (E_i, E_j) is not in W_Y, then it is nothing to prove as (E_i, E_j) in G_m. Assume $(E_i, E_j) \in W_Y$. Then there is a path on G_Y from E_i to E_j through the edges $(E_r, E_s) \notin W_Y$ and $Y \subseteq X_{r,s}$. This is because G_Y remains connected after removing W_Y. If all such $(E_r, E_s) \notin W_{X_{r,s}}$, then the required path is found, as all these edges are in G_m.

Otherwise, assume some $(E_r, E_s) \in W_Z$, where $Z = X_{r,s}$. Therefore $Y \subsetneq Z$ and the edge (E_r, E_s) was removed from G. Then there is a path on G_Z from E_r to E_s through edges $(E_k, E_l) \notin W_Z$. This is because G_Z is still connected after removing the edges W_Z. Moreover, $Y \subsetneq Z \subseteq X_{k,l}$ for (E_k, E_l). If all such $(E_k, E_l) \notin W_{X_{k,l}}$, then the required path is found, as all these edges are in G_m.

Otherwise, we continue so on and stop at some point as the sequence of the graphs $G_Y \supsetneq G_Z \supsetneq \ldots$ is strictly decreasing.

Let G' be any subgraph that satisfies property 1 from the minimal subgraph definition. The subgraph G' is produced by removing some edges W'_Y from every V_Y, where Y is a label in G. We claim that G_Y should be connected after removing edges W'_Y. Otherwise, there are two disconnected in G_Y vertices E_i and E_j, where $X_{i,j} = Y$ and the edge (E_i, E_j) was removed. Therefore the path (2) on G' does not exist for E_i and E_j. This contradicts with the definition of G'. Therefore $|W_Y| \geq |W'_Y|$ for every label Y as W_Y is the largest in size subset of edges in V_Y such that G_Y remains connected after removing W_Y. One concludes that G' has at least as many edges as G_m. The Lemma is proved. □

Example. Let there be five Boolean equations in five variables, where $X_1 = \{x_1, x_2, x_4\}$, $X_2 = \{x_1, x_2, x_3\}$, $X_3 = \{x_2, x_3, x_5\}$, $X_4 = \{x_3, x_4, x_5\}$ and $X_5 = \{x_1, x_4, x_5\}$. The equation graph G has 5 vertices and 10 edges: (E_1, E_2) labeled with $X_{1,2} = \{x_1, x_2\}$, (E_2, E_3) labeled with $X_{2,3} = \{x_2, x_3\}$, and so on. Five edges

$$(E_1, E_3), (E_1, E_4), (E_2, E_4), (E_2, E_5), (E_3, E_5)$$

are to be removed as they are obsolete for the Agreeing Algorithm; see Fig. 1.

2.2 Agreeing2 algorithm

This is an asymtotically faster variant of the Agreeing Algorithm, see [16].

(Precomputation.) For each maximal edge (E_i, E_j) find the set $X_{i,j}$ and the number $r = |X_{i,j}|$. For each r-bit address b the unordered tuple of lists

$$\{V_{i,j}(b); V_{j,i}(b)\} \tag{3}$$

is precomputed. The lists $V_{i,j}(b)$ and $V_{j,i}(b)$ consist of vectors from V_i and respectively V_j whose projection to variables $X_{i,j}$ is b. The list of tuples is sorted using some linear order. The algorithm marks vectors, which are wrong solutions, in tuples (3). Then all marked vectors are deleted from V_i. We say list $V_{i,j}(b)$ empty if it does not contain any entries or all its entries are marked.

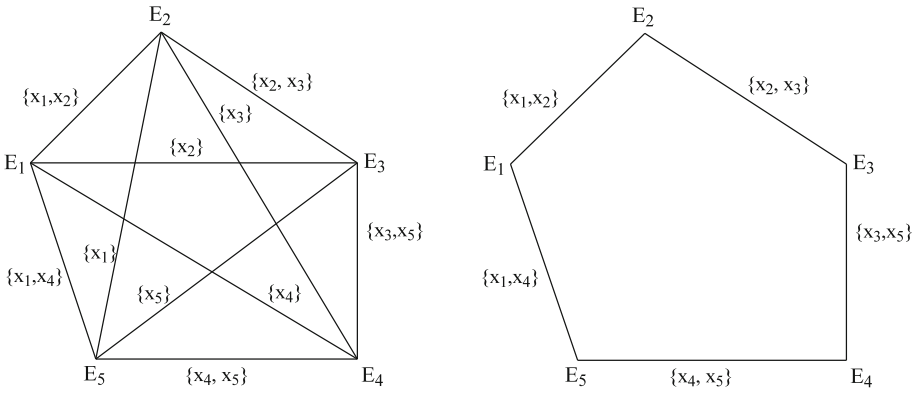

Fig. 1 Edges removing

(Agreeing.) The Algorithm starts with the first tuple $\{V_{i,j}(b); V_{j,i}(b)\}$, where just one of two lists is empty. If no such tuples are found, then a variable guess is necessary to produce some initial marking in tuples (see example below). Another option is to glue, see [18], some of the equations and recompute tuples. The Algorithm follows the rules:

1. Let the current tuple be $\{V_{i,j}(b); V_{j,i}(b)\}$, where $V_{i,j}(b)$ is empty, while $V_{j,i}(b)$ is not.
2. For every unmarked a in $V_{j,i}(b)$ do: mark a in $V_{j,i}(b)$, for every maximal edge (E_j, E_k) do: compute the projection d of a to variables $X_{j,k}$, mark a in $V_{j,k}(d)$, the tuple $\{V_{j,k}(d); V_{k,j}(d)\}$ is now current.
3. If just one of $V_{j,k}(d)$ or $V_{k,j}(d)$ is found empty, then apply step 1. If not, then take another maximal edge (E_j, E_k) or another unmarked a in $V_{j,i}(b)$. If $V_{j,i}(b)$ is already empty, then backtrack to the tuple last to $\{V_{i,j}(b); V_{j,i}(b)\}$. If the former was the starting tuple, then start a new walk with the next tuple, where just one of the lists is empty or terminate walking.
4. All vectors that have been earlier marked in the tuples are now deleted from V_i.

We see that for each starting tuple the algorithm walks through a search tree with backtracking. If new markings do not occur in the current tree, then the next tuple, where just one list is empty, is taken. The algorithm stops when in all tuples $\{V_{i,j}(b); V_{j,i}(b)\}$ the lists both are empty or both non-empty.

We remark that each tuple $\{a_1, \ldots, a_r; b_1, \ldots, b_s\}$ implements two implications. First, marking all $\{a_1, \ldots, a_r\}$ implies marking all $\{b_1, \ldots, b_s\}$, which can be denoted $\bar{a}_1, \ldots, \bar{a}_r \Rightarrow \bar{b}_1, \ldots, \bar{b}_s$, and vice versa $\bar{b}_1, \ldots, \bar{b}_s \Rightarrow \bar{a}_1, \ldots, \bar{a}_r$. Agreeing2 Algorithm simply expands marking through these implications.

Lemma 3 *Equation 1 are pairwise agreed if and only if in all* $\{V_{i,j}(b); V_{j,i}(b)\}$ *defined for maximal edges* (E_i, E_j) *the lists both are empty or both non-empty.*

Lemma 4 *Let for at least one edge* (E_i, E_j) *the lists* $V_{i,j}(b)$ *be empty for all* b*. Then the system is inconsistent.*

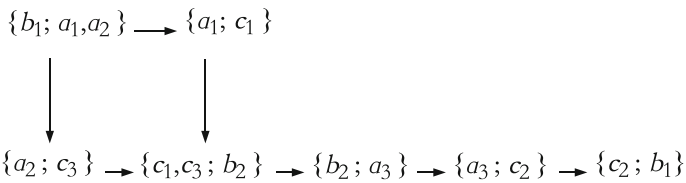

Fig. 2 The marking expansion

2.3 Example

Let three Boolean equations E_1, E_2, E_3 be given in algebraic normal form:

$$\begin{aligned} 1 + x_3 + x_1x_2 + x_1x_3 + x_1x_2x_3 &= 0, \\ 1 + x_1 + x_4 &= 0, \\ 1 + x_3 + x_2x_4 + x_3x_4 + x_2x_3x_4 &= 0. \end{aligned}$$

Represent them as lists of solutions:

	x_1	x_2	x_3
a_1	0	0	1
a_2	0	1	1
a_3	1	1	0

,

	x_1	x_4
b_1	0	1
b_2	1	0

,

	x_2	x_3	x_4
c_1	0	1	0
c_2	1	0	1
c_3	1	1	0

. (4)

The list of tuples: $P = \{a_1, a_2; b_1\}$, $Q = \{a_3; b_2\}$, $R = \{b_1; c_2\}$, $T = \{b_2; c_1, c_3\}$, $U = \{a_1; c_1\}$, $V = \{a_2; c_3\}$, $W = \{a_3; c_2\}$. As there are no tuples with just one list empty, a guess is necessary to start marking. We mark with a bar.

Assume $x_4 = 0$. So b_1 should be marked. We now have two tuples, where just one of the lists is empty: $\{\bar{b}_1; a_1, a_2\}$ and $\{\bar{b}_1; c_2\}$. According to the algorithm, take the first of two. Then a_1 get marked in $\{\bar{b}_1; a_1, a_2\}$ and $\{a_1; c_1\}$. Therefore, c_1 get marked in $\{\bar{a}_1; c_1\}$ and then in $\{c_1, c_3; b_2\}$. Now backtrack and mark a_2 in $\{\bar{b}_1; \bar{a}_1, a_2\}$ and $\{a_2; c_3\}$, and so on. The sequence of marking is represented in Fig. 2. Instances in all tuples have been marked. The guess was wrong. We alternatively could add a new tuple $\{b_1; \emptyset\}$ to the tuple list and start marking. Similarly, all tuple lists become empty in case $x_4 = 1$. The system has no solution.

3 Agreeing with a circuit lattice

Switches. Circuit lattice is a combination of switches and wires. There are two types of switches as in Fig. 3. Type 1 switch(1-switch) controls vertical circuits connected in parallel and powered by the same battery by any of horizontal circuits also connected in parallel and powered by another battery. So that voltage detected in at least one horizontal circuit makes the switch close. That may induce voltage in all vertical circuits simultaneously. Similarly, type 2 switch(2-switch) controls horizontal circuits connected in parallel by any of vertical circuits; voltage detected in a vertical circuit makes the switch close. That may induce voltage in all horizontal circuits. Only switches with one vertical and one horizontal input circuits are used in this Section in order to construct Circuit Lattice. Later, in Sect. 4 we will see that using switches with multiple horizontal and vertical input circuits enables constructing Reduced Circuit Lattices with a much lower number of switches.

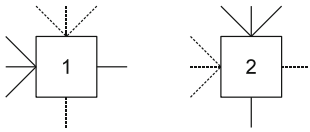

Fig. 3 The type 1 and 2 switches

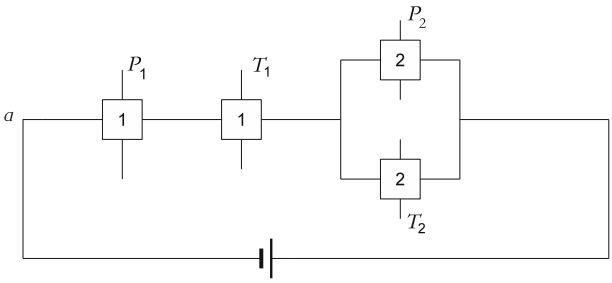

Fig. 4 The horizontal circuit for a particular solution a

Circuit lattice construction. Assume the list of tuples (3) is precomputed. The device is a lattice of horizontal and vertical circuits with intersections at switches of two types as in Fig. 6. The horizontal circuits are in one-to-one correspondence with solutions $a \in V_i$ to equations E_i in (1). So

1. each $a \in V_i$ defines the horizontal circuit labeled a as in Fig. 4. 1-switches on the horizontal circuit a are connected either in series or in parallel. We choose here series connection. 2-switches should be connected in parallel.
2. each tuple $\{a_1, \ldots, a_r; b_1, \ldots, b_s\}$ defines two vertical circuits, see Fig. 5. They implement two related implications. The left crosses horizontal circuits $a_1, \ldots, a_r$ at switches of type 1 and $b_1, \ldots, b_s$ at switches of type 2. Therefore it implements implication $\bar{a}_1, \ldots, \bar{a}_r \Rightarrow \bar{b}_1, \ldots, \bar{b}_s$. That means potential in all horizontal circuits $a_1, \ldots, a_r$ implies potential in all horizontal circuits $b_1, \ldots, b_s$ simultaneously. Similarly, the right circuit in Fig. 5 implements another implication $\bar{b}_1, \ldots, \bar{b}_s \Rightarrow \bar{a}_1, \ldots, \bar{a}_r$. Also see Fig. 6, which represents circuit lattice for Eq. 4.

The number of 1-switches equals the number of 2-switches on each horizontal circuit. This is the number of tuples (3), where a occurs. As the horizontal circuits are labeled by vectors $a \in V_i$, there are $\sum_i |V_i|$ horizontal circuits. Assume voltage (potential) is detected in a horizontal circuit. That is due to one of 2-switches on that circuit being closed. Then all 1-switches on this circuit get closed too. This may imply voltage in vertical circuits, e.g. in circuits P_1 and T_1 in Fig. 4. That happens if all other 1-switches on these vertical circuits (e.g. on P_1 and T_1) are closed. Then their 2-switches get closed. That affects new horizontal circuits and voltage expands, and so on. We remark that all horizontal circuits consume power from the same battery. All vertical circuits may be powered from another battery.

Solving. Solving starts with inducing potential into the circuit lattice. The potential may appear due to the tuples with just one of the lists empty. That is similar to Agreeing2 method explained before, as we start the algorithm with such tuples. So potential appears in one of two vertical circuit constructed from $\{\emptyset; b_1, \ldots, b_s\}$ as soon as the battery is switched on. This induces voltage in the horizontal circuits $b_1, \ldots, b_s$. Voltage may be then induced in some new vertical and horizontal circuits, and so on. One easily sees that potential is detected in a horizontal circuit labeled a if and only if a is marked by Agreeing2 algorithm. That is

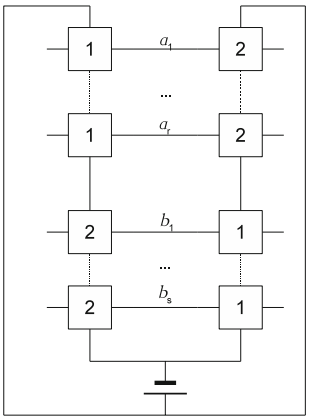

Fig. 5 The vertical circuits defined by $\{a_1, \ldots, a_r; b_1, \ldots, b_s\}$

a can't be a part of any common solution to Eq. 1. Therefore, the following statement is obvious.

Lemma 5 *Assume that after inducing potential in the circuit lattice, it is detected in each horizontal circuit $a_j \in V_i$ for at least one V_i. Then the system is inconsistent.*

If there are no tuples with just one empty list, then the device won't start. So variable guesses are to be introduced to start voltage expansion. Assume we are to guess the value $x \in X_i$ for some equation E_i. Let $a_1, \ldots, a_t$ be all vectors in V_i, where $x = 0$, and $a_{t+1}, \ldots, a_r$ all vectors in V_i, where $x = 1$. Each horizontal circuit $a \in V_i$ is provided with one additional 2-switch. It is connected in parallel with other 2-switches. Two vertical circuits are constructed: S_1 and S_2 by connecting new 2-switches above on horizontal circuits $a_{t+1}, \ldots, a_r$ and $a_1, \ldots, a_t$ respectively. It is not necessary to use 1-switches here as they won't play any role. To guess $x = 0$ one switches on the vertical circuit S_1, while S_2 is off. To guess $x = 1$ one switches on another vertical circuit S_2 with S_1 is off. See Fig. 6 for an example. Remark that S_1 and S_2 are there constructed for guessing the value x_4 in E_2.

Example. Circuit lattice for Eq. 4 is represented in Fig. 6. Two vertical circuits related to each tuple $P, Q \ldots$ are denoted $P_1, P_2, Q_1, Q_2 \ldots$. So that for the tuple $P = \{a_1, a_2; b_1\}$ we define two circuits, where P_1 implements implication $\bar{a}_1, \bar{a}_2 \Rightarrow \bar{b}_1$ and P_2 implements implication $\bar{b}_1 \Rightarrow \bar{a}_1, \bar{a}_2$. Similarly for the remaining tuples. There are two additional circuits S_1 and S_2 used for introducing guesses on x_4. Each of these two circuits incorporates one additional 2-switch. So the device is composed of 34 switches on the whole. In order to check $x_4 = 0$, one turns the circuit S_1 on, while S_2 is off. This results in 2-switch on the circuit S_1 get close and voltage appears in the horizontal circuit b_1. Two 1-switches on b_1 get closed and therefore voltage appears in two vertical circuits R_2 and P_2. All 2-switches on them become closed and voltage expands to the horizontal circuits a_1, a_2, c_2, and so on. Finally, after a number of simultaneous switch turns, voltage is detected in all horizontal circuits. The guess was wrong. Similarly, the circuit S_2 is switched on, S_1 is off, in order to check $x_4 = 1$. All horizontal circuits get voltage. The guess was wrong too. The system is therefore inconsistent.

The number of switches. The main characteristic of the device is the number of switches. This is twice the number of vectors in all tuples (3) for maximal edges and computed by the formula

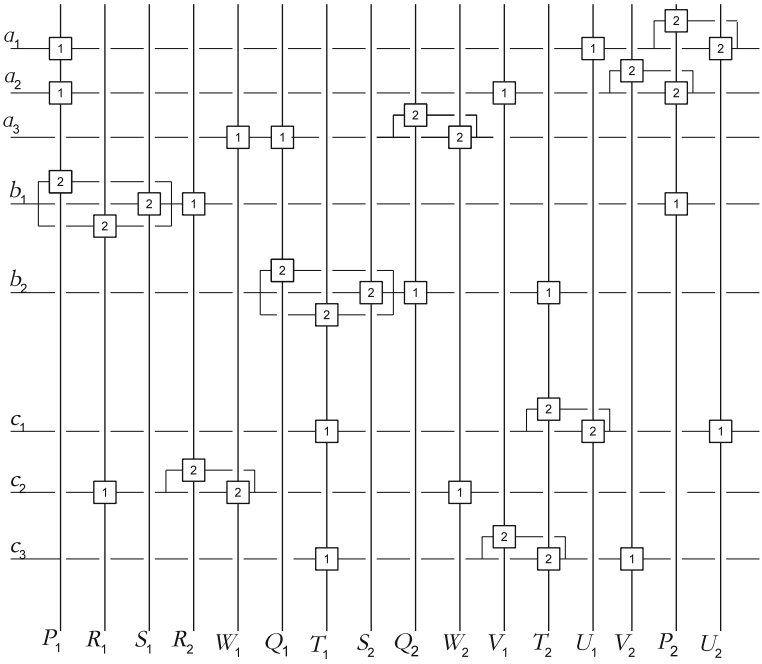

Fig. 6 The circuit lattice for Eq. 4

$$2\sum_{A}\sum_{b}(|V_{i,j}(b)|+|V_{j,i}(b)|)=2\sum_{A}(|V_i|+|V_j|). \tag{5}$$

The external sum is over all maximal edges $(E_i, E_j) \in A$ in G. For guessing s variables $x_1 \in X_{i_1}, \ldots, x_s \in X_{i_s}$ there should be also $|V_{i_1}|+\cdots+|V_{i_s}|$ additional switches.

The number of wires. We also count the number of wires necessary to connect switches in the circuit lattice. The number of wires in all vertical circuits is obviously the number of the lattice switches (5) plus the number of vertical circuits themselves. The latter value equals twice the number of tuples. In a horizontal circuit the type 2 switches are connected in parallel. So the number of wires is the number of type 1 switches plus twice the number of type 2 switches plus two. Therefore, the number of wires in all horizontal circuits is $3\sum_A(|V_i|+|V_j|)+2\sum_i|V_i|$. So the total number of wires should be

$$5\sum_{A}(|V_i|+|V_j|)+2\sum_{i}|V_i|+2\sum_{\text{tuples}}1. \tag{6}$$

For guessing s variables $x_1 \in X_{i_1}, \ldots, x_s \in X_{i_s}$ there should be also $|V_{i_1}|+\cdots+|V_{i_s}|+2s$ additional wires.

4 Reduced circuit lattices

In this section we briefly discuss how to reduce the parameters of the device(the number of switches and wires) through using switches that control several circuits connected in parallel, and controlled themselves by any of several parallel circuits, see Fig. 3.

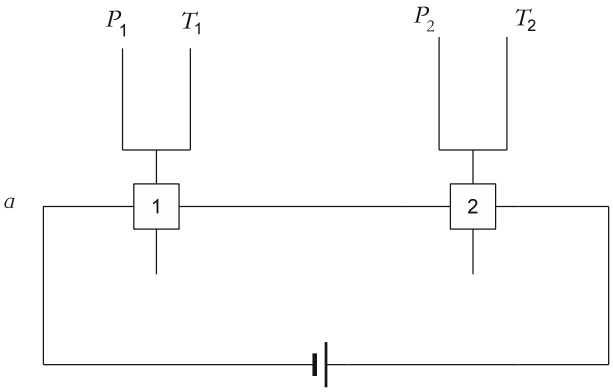

Fig. 7 The reduced horizontal circuit for a particular solution a

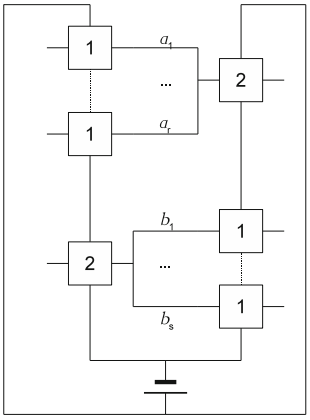

Fig. 8 The reduced vertical circuits defined by $\{a_1, \ldots, a_r; b_1, \ldots, b_s\}$

First, we modify each horizontal circuit so that it now comprises only one 1-switch and one 2-switch. The same 1-switch now controls all vertical circuits that passed via 1-switches on a horizontal circuit in above Circuit Lattice(CL). Then the same 2-switch controls that horizontal circuit by any of vertical circuits passing via 2-switches in CL. So the horizontal circuit in Fig. 4 now transforms into that in Fig. 7. We keep all vertical circuits intact. The number of switches becomes $2\sum_i |V_i|$, while the number of wires is essentially $2\sum_A(|V_i| + |V_j|)$. We call the described device Reduced Circuit Lattice 1(RCL1). It operates similarly to how CL operates.

We will further reduce the device parameters by observing that one 2-switch can control several horizontal circuits. We keep one type 1 switch on each horizontal circuit as above. Particular $a \in V_i$ are in one-to-one correspondence with 1-switches. So we say there is voltage in the horizontal circuit a if the related 1-switch is closed. However the connections in the vertical circuits related to a tuple are now as in Fig. 8, compare with that in Fig. 5. The number of switches now becomes $\sum_i |V_i| + 2\sum_{\text{tuples}} 1$, while the number of wires is essentially $2\sum_A(|V_i| + |V_j|)$. We call the described device Reduced Circuit Lattice 2(RCL2).

5 Guessing the variable values

Equations from a cipher. The number of key variables is commonly very small if compared with all system variables. Guessing all key variables results in the whole system collapsing by any of the Agreeing Algorithms. This is a variant of the brute force attack. If Agreeing works faster than this cipher encryption, then an advantage over common brute force attack is observed. It might well be that a proper subset of key variables should be guessed before the system is solved with Agreeing, see this paper Introduction, where the issue is briefly discussed.

Random equations. Generally, s-variable guesses result in 2^s trials(Agreeing runs). However, in randomly generated sparse equations there is a more efficient approach based on Gluing [15]. Assume that an s-bit guess is enough for solving (1) or finding it inconsistent with Agreeing. Look at the gluing of some t equations:

$$(X(t), U_t) = (X_{i_1}, V_{i_1}) \circ (X_{i_2}, V_{i_2}) \circ \cdots \circ (X_{i_t}, V_{i_t}),$$

where $s = |X(t)|$ and $X(t) = X_{i_1} \cup X_{i_2} \cup \ldots \cup X_{i_t}$. In other words, U_t is the set of all common solutions to the equations $E_{i_1}, \ldots, E_{i_t}$. The number of vectors in U_t is 2^{s-t} on the average, see Lemma 4 in [18]. The vectors U_t are produced one after the other as in [18] with the cost per vector proportional to t. So it is not necessary to keep the whole set U_t. This is true for t smaller than some critical value $\frac{\alpha_0 n}{l}$, where $\alpha_0 = 2^{1/l} \ln\left(\frac{1-1/2}{1-(1/2)^{1/l}}\right)$, see [18]. So the total complexity of solving is roughly proportional to 2^{s-t} of Agreeing runs.

6 DES and triple DES equations

The DES and the Triple DES equation systems are constructed in Appendix, where input/output 64-bit blocks are considered variables too. So each equation comprises 20 variables and admits 2^{16} solutions. Table 2 provides with the equation system parameters as the number of equations, the number of variables, the number of edges of the adjacent graph with nonempty labels, the number of maximal edges and the number of tuples (3). Any of Circuit Lattices may be used to compute the key for any given plain-texts and related cipher-texts. These are introduced into a Circuit Lattice similarly to the guessed key-bits. However plain-text, cipher-text bits are not changing during the whole computation. So any CL should have $2 \times 56 + 2 \times 128 = 368$ input contacts for the DES and $2 \times 112 + 2 \times 128 = 480$ input contacts for the Triple DES.

Tables 3 and 4 show main characteristics of Circuit Lattices for DES and Triple DES: the number of necessary switches, wires and input contacts, which are computed by formulas (5) and (6) and in Sect. 4.

Two plain-text, cipher-text 64-bit blocks uniquely define 112-bit key in the Triple DES. So for the key search there should be two above described devices working in parallel. The speed

Table 2 DES and triple DES equations

Nmbr of	Eqns	Vrbls	Edges	Mx.edges	Tuples
DES	128	632	3,545	1,409	16,636
TDES	384	1,712	16,831	3,929	71,320

Table 3 DES circuit lattice implementations

Nmbr of	Switches	Wires	Input contacts
CL	3.9×10^8	9.5×10^8	368
RCL1	1.7×10^7	3.9×10^8	368
RCL2	8.5×10^6	3.9×10^8	368

Table 4 TDES circuit lattice implementations

Nmbr of	Switches	Wires	Input contacts
CL	1.1×10^9	2.7×10^9	480
RCL1	5.1×10^7	1.1×10^9	480
RCL2	2.6×10^7	1.1×10^9	480

of computation is determined by the time that a switch takes to turn. However, how many switch turns are necessary before the system is found inconsistent looks generally difficult to estimate. This is an open problem. Voltage expands in a highly parallel manner through several circuits which affect each other and many switches turn simultaneously. Fortunately, this is easy for round ciphers like DES or Triple DES. Assume guessing all key variables at once. Then all Type 1 switches in tuples related to pairs of equations in subsequent rounds turn simultaneously when voltage expands from one round to another. That makes related Type 2 switches turn too. This is so even if the Agreeing only runs through maximal edges of the adjacent graph. Therefore the time measured in switch turns that the solver takes to agree pairwise all equations is twice the number of rounds. In particular, to reject one wrong key in the Triple DES takes at most 2×48 switch turns.

7 Conclusion, open problems and discussion

The paper describes a hardware implementation of the Agreeing Algorithm aimed to find solutions to a system of sparse Boolean equations, e.g. coming from ciphers. Some variables guess is introduced into the device which signals out if the system is inconsistent after that guess. The device architecture implemented with a lattice of circuits is transparent. However, this is an open problem whether the circuit lattice for a real world cipher like DES or Triple DES is implementable within the current technology in computer industry.

There are several related problems:

1. The number of switches is the most important parameter of the solver. Table 3 and 4 data shows that the equation systems for the DES and the Triple DES require the number of switches which is within the number of transistors now available on one semiconductor crystal. For instance, Intel announced Dual-Core Itanium2 processor with more than 1.7 billion transistors, see [9]. Obviously, a transistor is able to work as a switch.
2. Special purpose hardware to supply one after the other guesses on fixed variables is to be devised. Its speed should be comparable with that of the solver. The device is similarly constructed in wires and switches and controlled by the output signal from the solver. We do not discuss this in detail as its construction is rather obvious. It is easy to understand that its speed should be only 2 switch turns on the average.
3. The transistor speed(the speed of a turn) is constantly increasing. E.g., historical 17% year performance improvement is also predicted in [24] for the next decade. Then a new

speed record for the world fastest transistor which is more than 1THz(1,000 GHz), see [10], was reported. However, to be on the safe side we assume available transistors with speed about 100 GHz. Assume it is feasible to integrate one billion or so such transistors on one semiconductor chip as a Triple DES Circuit Lattice. We remark that Reduced Circuit Lattices require a much lower number of transistors; see Table 4. Then average time for producing a guess on 112 key variables and finding the system's inconsistence is approximately $2 \times 48 + 2 = 98$ switch turns. So the key rejecting rate is approximately 1 GHz in this case. It is compared favorably with what is currently achieved, about 0.034 GHz.

Acknowledgments The author is grateful to Håvard Raddum and Valerij Kutuzov for useful discussions, Thorsten Schilling for indicating a flaw in the first version of Lemma 2, anonymous referees from WCC'09 and "Designs, Codes and Cryptography" for thoroughly reading the paper and numerous suggestions on improving the presentation. The author was partially supported by the grant NIL-I-004 from Iceland, Lichtenstein and Norway through the EEA Financial Mechanism and the Norwegian Financial Mechanism.

8 Appendix

In this Appendix we describe how to make the equation system from the DES algorithm. The similar equations are constructed for the Triple DES. The input and output applications of the permutation IP are ignored as well as the final swap between 32-bit sub-blocks. The 64-bit internal state of the cipher after the i-th round is denoted by (R_{i-1}, R_i). In particular, (R_{-1}, R_0) denotes the 64-bit plain-text block and (R_{15}, R_{16}) is the related cipher-text block. All these 128 bits are generally considered known constants. But we write them variables. So that when the Agreeing algorithm is being run, these 128 variables are substituted by constants as if for guessing. Therefore, 576 state variables are bits of $R_{-1}, R_0, R_1, \dots, R_{15}, R_{16}$. They are numbered $-63, -62, \dots, 512$. 56 key variables are numbered by $512 + j$, where $1 \le j \le 64$ and $j \ne 8, 16, \dots, 64$.

At every round $i = 1, 2, \dots, 16$, sub-blocks R_i are related as

$$R_i \oplus R_{i-2} = PS(\overline{R_{i-1}} \oplus K_i), \tag{7}$$

where $\overline{R_{i-1}}$ is the 48-bit expansion of the 32-bit R_{i-1} and K_i is the round key. P denotes the fixed permutation on 32 symbols and S is the transform implemented by 8 S-boxes. The Eq. 7 is equivalent to 8 equations related to each of the S-boxes S_j:

$$(P^{-1}(R_i))_j \oplus (P^{-1}(R_{i-2}))_j = S_j((\overline{R_{i-1}})_j \oplus K_{i,j}), \tag{8}$$

where $R_{i,j}$ is a 4-bit sub-block of R_i, and $K_{i,j}$ is a 6-bit sub-block of K_i and $(T)_j$ denotes a 6(or 4)-bit sub-block of T. The Eq. 8 is denoted by $E_{i,j} = E_{j+8(i-1)}$. The full system of the DES equations consists of 128 equations E_t, $t = 1, 2, \dots, 128$. One equation incorporates 20 variables. For instance, $E_{8,4} = E_{60}$ depends on 20 variables:

$$\begin{aligned}
(P^{-1}(R_6))_4 &= (x_{161}, x_{170}, x_{180}, x_{186}),\\
(\overline{R_7})_4 &= (x_{204}, x_{205}, x_{206}, x_{207}, x_{208}, x_{209}),\\
(P^{-1}(R_8))_4 &= (x_{225}, x_{234}, x_{244}, x_{250}),\\
K_{8,4} &= (x_{514}, x_{529}, x_{538}, x_{539}, x_{556}, x_{561}).
\end{aligned}$$

These variables compose the set X_{60}. For any values of the following 16 variables:

$$x_{204}, x_{205}, x_{206}, x_{207}, x_{208}, x_{209}, x_{225}, x_{234},$$
$$x_{244}, x_{250}, x_{514}, x_{529}, x_{538}, x_{539}, x_{556}, x_{561},$$

the values of $x_{161}, x_{170}, x_{180}, x_{186}$ are uniquely defined by (8). So 2^{16} vectors of length 20 compose the list V_{60}. That is all equations have 2^{16} solutions. Let $m \to E_K(m)$ denote the encryption function on plain-text blocks with the DES algorithm. Then the Triple DES implements the mapping:

$$m \to E_{K_1}(E_{K_2}(E_{K_1}(m))).$$

Therefore Triple DES equations are determined similarly to those for the DES.

References

1. Bardet M., Faugére J.-C., Salvy B.: Complexity of Gröbner basis computation for semi-regular overdetermined sequences over F_2 with solutions in F_2, Research report RR–5049, INRIA (2003).
2. Clayon R., Bond M.: Experience using a low-cost FPGA design to crack DES keys. In: CHES 2002, LNCS 2523, pp. 579–592, Springer-Verlag (2002).
3. Courtois N., Klimov A., Patarin J., Shamir A.: Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In: Eurocrypt 2000, LNCS 1807, pp. 392–407, Springer-Verlag (2000).
4. Courtois N.T., Bard G.V.: Algebraic Cryptanalysis of the Data Encryption Standard, Cryptology ePrint Archive: Report 2006/402.
5. Courtois N.T., Bard G.V., Wagner D.: Algebraic and Slide Attacks on KeeLoq, Cryptology ePrint Archive: Report 2007/062.
6. Diffie W., Hellman M.: Exhaustive cryptanalysis of the NBS Data Encryption Standard. Computer **10**(6), 74–84 (1977).
7. Electronic Frontier Foundation: Cracking DES: Secrets of Encryption Research, Wiretap Politics and Chip Design, O'Reilly and Assotiates Inc. (1998).
8. Faugère, J.-C.: A new efficient algorithm for computing Gröbner bases without reduction to zero (F5), Proc. of ISSAC 2002, pp. 75–83, ACM Press (2002).
9. http://www.intel.com.
10. http://www.semiconductor.net/article/CA6514491.html.
11. Geiselmann W., Matheis K., Steinwandt R.: PET SNAKE: A Special Purpose Architecture to Implement an Algebraic Attack in Hardware, Cryptology ePrint Archive, 2009/222.
12. Iwama K.: Worst-case upper bounds for kSAT, The Bulletin of the EATCS, **82**, 61–71 (2004).
13. Kumar S., Paar C., Pelzl J., Pfeiffer G., Schimmler M.: Breaking ciphers with Copacobana-a cost-optimized parallel code breaker. In: CHES2006, LNCS 4249, pp. 101–118, Springer-Verlag(2006).
14. Osvik D.A., Tromer E.: Cryptologic applications of the Playstation3: Cell SPEED, http://www.hyperelliptic.org/SPEED/slides/Osvik_cell-speed.pdf.
15. Raddum H., Semaev I.: New technique for solving sparse equation systems, Cryptology ePrint Archive, 2006/475.
16. Raddum H., Semaev I.: Solving Multiple Right Hand Sides linear equations, Designs, Codes and Cryptography, vol. **49**, pp. 147–160 (2008), extended abstract in Proceedings of WCC'07, 16-20 April 2007, Versailles, France, INRIA (2007).
17. Rouvroy G., Standaert F.-X., Quisquater J.-J., Legat, J.-D.: Design strategies and modified desciptions to optimize cipher FPGA implementations: fast and compact results for DES and Triple-DES. In: FPL2003, LNCS 2778, pp. 181–193, Springer-Verlag (2003).
18. Semaev I.: On solving sparse algebraic equations over finite fields, Designs, Codes and Cryptography, vol. **49**, pp. 47–60,(2008), extended abstract in Proceedings of WCC'07, 16-20 April 2007, Versailles, France, INRIA(2007).
19. Semaev I.: Sparse algebraic equations over finite fields. SIAM J. Comput. **39**, 388–409 (2009).
20. Semaev I.: Improved Agreeing-Gluing algorithm. In: Proceedings of SCC'10, 23–25 June 2010, Royal Holloway, University of London, UK, pp. 73–88.
21. Yang B.-Y., Chen J-M., Courtois, N.: On asymptotic security estimates in XL and Gröbner bases-related algebraic cryptanalysis. In: ICICS 2004, LNCS 3269, pp. 401–413, Springer-Verlag (2004).

22. Wiener M.J.: Efficient DES key search. In: Stalling, W.R. (ed.) Practical Cryptography for Data Interworks, pp.31–79. IEEE Computer Society Press (1996).
23. Zakrevskij A., Vasilkova I.: Reducing large systems of Boolean equations. In: 4th Int. Workshop on Boolean Problems, Freiberg University, September, pp. 21–22 (2000).
24. Zeitzoff P.: 2007 International Technology Roadmap: MOSFET scaling challenges, Solid State Technology Magazine, February (2008).