Sparse Boolean equations and circuit lattices

Igor Semaev[*]

Department of Informatics, University of Bergen, Norway
igor@ii.uib.no

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.

Key words: Sparse Boolean equations, equations graph, electrical circuits, switches

1 Introduction

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

$$f_1(X_1) = 0, \ldots, f_m(X_m) = 0 \qquad (1)$$

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.

In [13] Agreeing and Gluing procedures were described. Then they were combined with variables guessing to solve (1). See also earlier work [20]. 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 and the same variables. So l-SAT solving algorithms provide with worst case complexity estimates. The table data

Table 1. Algorithms' running time.

l	3	4	5	6
the worst case,[11]	1.324^n	1.474^n	1.569^n	1.637^n
Gluing1, expectation,[16]	1.262^n	1.355^n	1.425^n	1.479^n
Gluing2, expectation,[16]	1.238^n	1.326^n	1.393^n	1.446^n
Agreeing-Gluing1, expectation,[17]	1.113^n	1.205^n	1.276^n	1.334^n

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 tool 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 [8, 3], are found in [1, 18]. 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 [14] that a linear algebra variant of the Agreeing-Gluing significantly overcomes(on AES type Boolean equations in around 50 variables) 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. E.g. for the DES instead of 3545 pairs, the algorithm should only run through 1404 of them with the same output. In case of the Triple DES the figure is 3929 instead of 16831, see Table 2.

Then we suggest implementing the Agreeing Algorithm in hardware. The main features of the related device 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. 4 for instance.
- Voltage is induced by variables guess. 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.

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 device 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 [6, 19, 2, 7, 15, 12] 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 is 0.1 GHz per chip, [12]. Therefore 0.034 GHz for the Triple DES. This is the key rejecting rate.

In contrast, our idea is to not implement any encryption. Having been constructed, the device might achieve a higher key rejecting rate, see the discussion in Section 6. 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 [13] 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 [14] that admitting up to 2^s right hand sides(produced with Gluing) 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.

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 Agreing 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 [14]. 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 Section 3.

The equations $E_1, \ldots, E_m$ are vertices in an equations 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. We will now show that some of the edges in the graph G are obsolescent 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.

The edges of a minimal subgraph are called maximal and denoted A for some fixed G_m. They 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. Therefore, there exists a path (2), 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 agreeing pairwise E_i, E_k, then $E_k, E_l, \ldots$ and E_r, E_j. This proves the Lemma.

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

1. For any $Y = X_{i,j}$ find all edges (E_s, E_r) in G such that $Y \subseteq X_{s,r}$. Denote a subgraph of G on the vertices $E_s, E_r, \ldots$ with all such edges (E_s, E_r) by G_Y. 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 largest subset $W_Y \subseteq V_Y$ such that G_Y is connected after removing the edges W_Y.
3. Remove the edges W_Y from G for all $Y = X_{i,j}$ and get G_m.

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

Proof. We first prove that for any edge (E_i, E_j) in G there is a path (2) on G_m. 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) is 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) not in 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) are not in $W_{X_{r,s}}$, 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 (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) not occurring in W_Z. This is because G_Z is still connected after removing the edges W_Z. Moreover, $Y \subseteq Z \subseteq X_{k,l}$ for such (E_k, E_l). If all such (E_k, E_l) are not in $W_{X_{k,l}}$, then the required path is found, as all these edges are in G_m.

Otherwise, we continue so on. One stops at some point as the sequence of the graphs $G_Y \supset G_Z \supset \ldots$ is strictly decreasing.

The resulting graph G_m is with minimal number of edges. Otherwise, let be possible to remove one more edge (E_r, E_s) from G_m and still have some path (2) for any (E_i, E_j). Then one finds a bigger set W_Z, where $Z = X_{r,s}$, such that removing W_Z from G_Z keeps this graph connected. That is impossible by the definition of W_Z. This proves the Lemma.

Example. Let there be five Boolean equations in four variables, where $X_1 = \{x_1, x_2\}, X_2 = \{x_2, x_3\}, X_3 = \{x_3, x_4\}, X_4 = \{x_1, x_3\}$ and $X_5 = \{x_2, x_4\}$. The graph G has 5 vertices and 7 edges: (E_1, E_2) labeled with $X_{1,2} = \{x_2\}$, (E_2, E_3) labeled with $X_{2,3} = \{x_3\}$, and so on. Two edges (E_1, E_2) and (E_2, E_4) are to be removed as they are obsolescent for the Agreeing Algorithm.

2.2 Agreeing2 Algorithm

This is an asymtotically faster variant of the Agreeing Algorithm, see [14].

(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 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 set of tuples is sorted using some linear order. The algorithm marks vectors in tuples (3), then deletes all marked vectors from V_i. We say list $V_{i,j}(b)$ empty if it does not contain entries or all they are marked.

(Agreeing.) The Algorithm starts with the first tuple $\{V_{i,j}(b); V_{j,i}(b)\}$, where just one list is empty and 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. Then all the vectors a in $V_{j,i}(b)$ are made marked one after one.

2. For a in $V_{j,i}(b)$ the projection d of a to variables $X_{j,k}$ is computed, where (E_j, E_k) is a maximal edge. Then a in $V_{j,k}(d)$ is made marked. 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 mark another a in $V_{j,i}(b)$. If $V_{j,i}(b)$ is already empty, then backtrack to the tuple last to $\{V_{i,j}(b); V_{j,i}(b)\}$.
4. For each starting tuple the algorithm walks through a search tree with backtracking. If new deletions do not occur in the current tree, then the next tuple, where just one list is empty, is taken.
5. The algorithm stops when in all tuples $\{V_{i,j}(b); V_{j,i}(b)\}$ the lists both are empty or both non-empty. Then all vectors that have been earlier marked in the tuples are now deleted from V_i.

Lemma 3. *Equations (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 maximal edge (E_i, E_j) the lists $V_{i,j}(b)$ be empty for all b. Then the system is inconsistent.*

2.3 Example

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

$$x_3 + x_1 x_2 + x_1 x_3 + x_1 x_2 x_3 = 0,$$
$$1 + x_1 + x_4 = 0,$$
$$x_3 + x_2 x_4 + x_3 x_4 + x_2 x_3 x_4 = 0.$$

Represent them as lists of solutions:

$$
\begin{array}{c|ccc}
 & x_1 & x_2 & x_3 \\
\hline
a_1 & 0 & 0 & 1 \\
a_2 & 0 & 1 & 1 \\
a_3 & 1 & 1 & 0
\end{array}
,\quad
\begin{array}{c|cc}
 & x_1 & x_4 \\
\hline
b_1 & 0 & 1 \\
b_2 & 1 & 0
\end{array}
,\quad
\begin{array}{c|ccc}
 & x_2 & x_3 & x_4 \\
\hline
c_1 & 0 & 1 & 0 \\
c_2 & 1 & 0 & 1 \\
c_3 & 1 & 1 & 0
\end{array}
. \tag{4}
$$

The list of tuples is: $T_1 = \{a_1, a_2; b_1\}$, $T_2 = \{a_3; b_2\}$, $T_3 = \{b_1; c_2\}$, $T_4 = \{b_2; c_1, c_3\}$, $T_5 = \{a_1; c_1\}$, $T_6 = \{a_2; c_3\}$, $T_7 = \{a_3; c_2\}$. 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 got marked in $\{\bar{b}_1; a_1, a_2\}$ and in $\{a_1; c_1\}$. Therefore, c_1 got 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 in $\{a_2; c_3\}$, and so on. This implies one search tree:

$$
\begin{array}{l}
\qquad\qquad \nearrow \{\bar{a}_1; \bar{c}_1\} \rightarrow \{\bar{c}_1, c_3; b_2\} \\
\{\bar{b}_1; \bar{a}_1, \bar{a}_2\} \\
\qquad\qquad \searrow \{\bar{a}_2; \bar{c}_3\} \rightarrow \{\bar{c}_1, \bar{c}_3; \bar{b}_2\} \rightarrow \{\bar{b}_2; \bar{a}_3\} \rightarrow \{\bar{a}_3; \bar{c}_2\} \rightarrow \{\bar{c}_2; \bar{b}_1\}.
\end{array}
$$

The vectors in all tuples have been marked. The guess was wrong. We alternatively could add $\{b_1; \emptyset\}$ 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. 1. Type 1 switch controls the vertical circuit by the horizontal circuit, which means that voltage detected in the horizontal circuit makes the switch close. That may induce voltage in the vertical circuit if other type 1 switches on that circuit are closed too. Similarly, type 2 switch controls the horizontal circuit by the vertical circuit, which means that voltage detected in the vertical circuit makes the switch close. That may induce voltage in the horizontal circuit.


Fig. 1. Type 1 and 2 switches.

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. 4. The horizontal circuits are in one-to-one correspondence with solutions $a \in V_i$ to equations E_i in (1) taken separately. So

1. each $a \in V_i$ defines the horizontal circuit marked a as in Fig. 2. Type 1 switches on the horizontal circuit a are connected in series. Type 2 switches are connected in parallel.
2. each tuple $\{a_1, \ldots, a_r; b_1, \ldots, b_s\}$ defines two vertical circuits, see Fig. 3. Both of them cross horizontal circuits marked $a_1, \ldots, a_r, b_1, \ldots, b_s$. One crosses horizontal circuits $a_1, \ldots, a_r$ at switches of type 1 and $b_1, \ldots, b_s$ at switches of type 2. Another vertical circuit crosses $a_1, \ldots, a_r$ at switches of type 2 and $b_1, \ldots, b_s$ at switches of type 1. Also see Fig. 4, which represents circuit lattice for equations (4).

The number of type 1 switches equals the number of type 2 switches on each horizontal circuit. This is the number of tuples (3), where a occurs. As the horizontal circuits are marked by vectors $a \in V_i$, there are $\sum_i |V_i|$ horizontal circuits. Switches of type 1 control vertical circuits by horizontal circuits and switches of
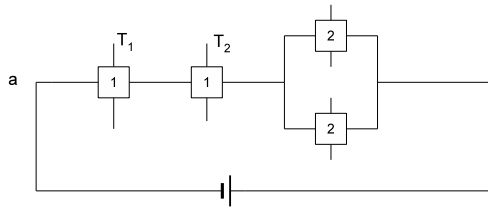
Fig. 2. Horizontal circuit for a particular solution a.

type 2 control horizontal circuits by the vertical. Assume voltage(potential) is detected in a horizontal circuit. That is due to one of type 2 switches on that circuit was closed. Then all type 1 switches on this circuit get closed too. This may imply voltage in vertical circuits, e.g. in circuits T_1 and T_2 in Fig. 2. That happens if all other type 1 switches on these vertical circuits(e.g. on T_1 and T_2) are closed. Then their type 2 switches get closed. That affects new horizontal circuits and voltage expands so on. We remark that all horizontal circuits consume power from the same battery. All vertical circuits may be powered from another battery.
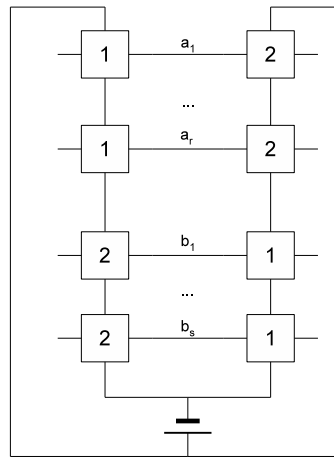
Fig. 3. Two vertical circuits defined by $\{a_1, \ldots, a_r; b_1, \ldots, b_s\}$.

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. Potential is detected in the horizontal circuit a if and only if a is marked by Agreeing2 algorithm. That is a can't be a part of any common solution to equations (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 of $x \in X_i$ 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 type 2 switch connected in parallel with other type 2 switches. Two vertical circuits are constructed: S_1 and S_2 by connecting new type 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 type 1 switches here as they won't play any role. Guessing $x = 0$ is accomplished by switching on the vertical circuit S_1, while S_2 is off, and guessing $x = 1$ is done by switching on another vertical circuit S_2 with S_1 is off. See Figure 4 for an example. Remark that S_1 and S_2 are there implementations for guessing the value of x_4 in E_2.

Example. Circuit lattice in case of (4) is represented in Fig. 4. Two vertical circuits related to tuples T_i are denoted T_i' and T_i''. 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 type 2 switch. So the device composes of 34 switches on the whole. In order to check $x_4 = 0$, one makes the circuit S_1 switch on, while S_2 is off. This results in type 2 switch on the circuit S_1 get close and voltage appears in the horizontal circuit b_1. Two type 1 switches on b_1 get closed and therefore voltage appears in two vertical circuits T_3'' and T_1''. All type 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

$$2 \sum_A \sum_b (|V_{i,j}(b)| + |V_{j,i}(b)|) = 2 \sum_A (|V_i| + |V_j|). \tag{5}$$
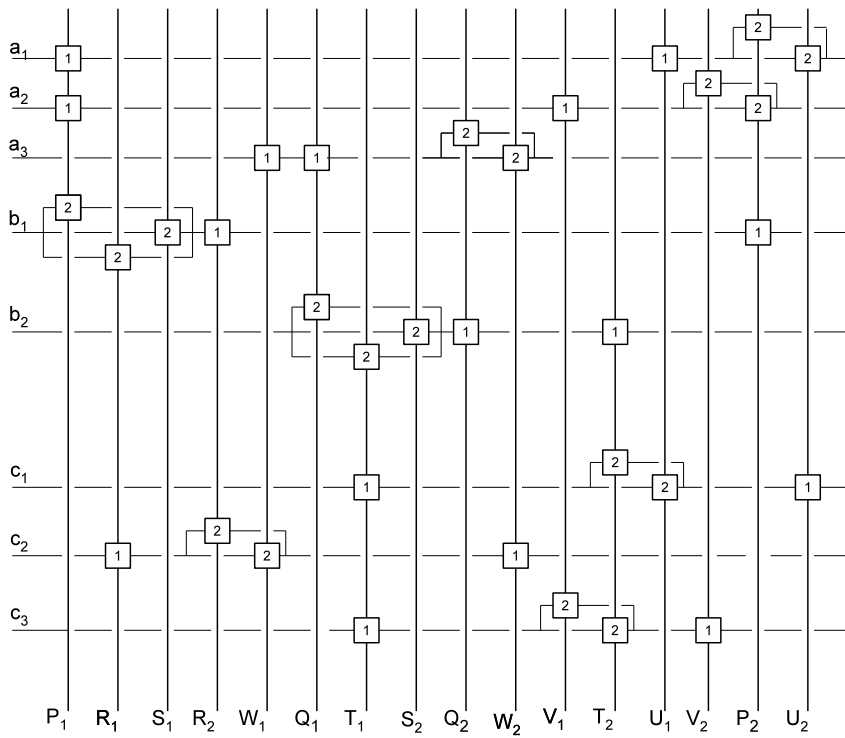
Fig. 4. Circuit lattice for equations (4).

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 $|V_{i_1}| + \ldots + |V_{i_s}|$ 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

$$5 \sum_A (|V_i| + |V_j|) + 2 \sum_i |V_i| + 2 \sum_{T_i} 1. \tag{6}$$

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

4 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 collapses by any of the Agreeing Algorithms. This is a variant of the brute force attack. If the Agreeing works faster than this cipher encryption, then an advantage over common brute force attack is observed. It might be well that a proper subset of key variables should be guessed before the system is solved with Agreeing, see this paper Introduction, where the issue was 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 [13]. Assume that an s-bit guess is enough for solving (1) with this method. Look at the gluing of some t equations:

$$(X(t), U_t) = (X_{i_1}, V_1) \circ (X_{i_2}, V_2) \circ \ldots \circ (X_{i_t}, V_t),$$

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 [16]. The vectors U_t are produced one after one as in [16]. The cost per vector is proportional to t. This is true for t smaller than some critical value $\frac{\alpha_0 n}{l}$, where $\alpha_0 = 2^{1/l} \ln(\frac{1-1/2}{1-(1/2)^{1/l}})$, see [16]. So the total complexity of solving is roughly proportional to 2^{s-t} of Agreeing runs.

Table 2. DES and Triple DES equations implementation.

Nmbr of	eqns	vrbls	edges	mx.edges	tuples	switches	wires
DES	128	632	3545	1409	16636	3.9×10^8	9.5×10^8
TDES	384	1712	16831	3929	71320	1.1×10^9	2.7×10^9

.

5 DES and Triple DES equations

The DES and the Triple DES equation systems are constructed in Appendix B. Assume that the input/output 64-bit blocks are variables. Then each equation comprises 20 variables and admits 2^{16} solutions. The device may be used to compute the key for any given plain-texts and related cipher-texts. These are introduced to the solver similarly to the guessed bits. However plain-text, cipher-text bits are not changing during the computation.

For both DES and Triple DES Table 2 represents data on the equation systems describing the ciphers: 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). Then the number of necessary switches and wires in the related circuit lattice is computed by formulas (5) and (6). 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 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.

6 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 2 data shows that the equation systems representing 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 one guesses on fixed variables is to be devised. Its speed should be comparable with that of the solver. The device is also constructed in wires and switches and controlled by the output signal from the solver. It is easy to understand, it 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 [21] for the next decade. Then a new speed record for the world fastest transistor which is more than 1THz(1000GHz), see [10], was reported. However, to be on the safe side we assume available transistors with speed about 100GHz. Assume it is feasible to integrate one billion or so such transistors on one semiconductor chip as a Triple DES circuit lattice. 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 1GHz in this case. It is compared favorably with what is currently achieved, about 0.034GHz.

References

1. M. Bardet, J.-C.Faugére, and B. Salvy, *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. R. Clayon and M.Bond, *Experience using a low-cost FPGA design to crack DES keys*, in CHES 2002, LNCS 2523, pp. 579–592, Springer-Verlag, 2002.
3. N. Courtois, A. Klimov, J. Patarin, and A. Shamir, *Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations,* in Eurocrypt 2000, LNCS 1807, pp. 392–407, Springer-Verlag, 2000.
4. N. T. Courtois and G. V. Bard, *Algebraic Cryptanalysis of the Data Encryption Standard*, Cryptology ePrint Archive: Report 2006/402.
5. N. T. Courtois, G. V. Bard, and D. Wagner, *Algebraic and Slide Attacks on KeeLoq*, Cryptology ePrint Archive: Report 2007/062.
6. W. Diffie and M. Hellman, *Exhaustive cryptanalysis of the NBS Data Encryption Standard,* Computer, 10(6), 1977, pp.74–84.
7. Electronic Frontier Foundation, *Cracking DES: Secrets of Encryption Research, Wiretap Politics and Chip Design*, O'Reilly and Assotiates Inc., 1998.
8. J.-C. Faugère, *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/itanium
10. http://www.semiconductor.net/article/CA6514491.html
11. K. Iwama, *Worst-Case Upper Bounds for kSAT,* The Bulletin of the EATCS, vol. 82(2004), pp. 61–71.

12. S. Kumar, C. Paar, J. Pelzl, G. Pfeiffer, and M. Schimmler, *Breaking ciphers with Copacobana-a cost-optimized parallel code breaker*, in CHES2006, LNCS 4249, pp. 101-118, 2006.
13. H. Raddum and I. Semaev, *New technique for solving sparse equation systems*, Cryptology ePrint Archive, 2006/475.
14. H. Raddum and I. Semaev, *Solving Multiple Right Hand Sides linear equations*, Designs, Codes and Cryptography, vol. 49(2008), pp. 147–160, extended abstract in Proceedings of WCC'07, 16-20 April 2007, Versailles, France, INRIA, pp. 323–332.
15. G. Rouvroy, F.-X. Standaert, J.-J. Quisquater, and J.-D. Legat, *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, 2003.
16. I. Semaev, *On solving sparse algebraic equations over finite fields*, Designs, Codes and Cryptography, vol. 49(2008), pp. 47–60, extended abstract in Proceedings of WCC'07, 16-20 April 2007, Versailles, France, INRIA, pp. 361–370.
17. I. Semaev, *Sparse algebraic equations over finite fields*, to appear in SIAM Journal on Computing, 2009, see also in Cryptology ePrint Archive, 2007/280.
18. B.-Y. Yang, J-M. Chen, and N.Courtois, *On asymptotic security estimates in XL and Gröbner bases-related algebraic cryptanalysis,* in ICICS 2004, LNCS 3269, pp. 401–413, Springer-Verlag, 2004.
19. M.J. Wiener, *Efficient DES key search*, In Willam R.Stalling, editor, Practical Cryptography for Data Interworks, pp.31–79, IEEE Computer Society Press,1996.
20. A. Zakrevskij, I. Vasilkova, *Reducing large systems of Boolean equations,* 4th Int.Workshop on Boolean Problems, Freiberg University, September, 21–22, 2000.
21. P. Zeitzoff, *2007 International Technology Roadmap: MOSFET scaling challenges,* Solid State Technology Magazine, February 2008.

7 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 $R_{-1}, R_0, R_1, \ldots, R_{15}, R_{16}$. They are numbered $-63, -62, \ldots, 512$. 56 key variables are numbered by $512 + j$, where $1 \le j \le 64$ and $j \neq 8, 16, \ldots, 64$.

At every round $i = 1, 2, \ldots, 16$, 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 equation (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 equation (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, $t = 1, 2, \ldots, 128$. One equation incorporates 20 variables. For instance, $E_{8,4} = E_{60}$ depends on 20 variables:

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

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.