VERSITA

TATRA MOUNTAINS
Mathematical Publications

DOI: 10.2478/v10127-012-0036-6
Tatra Mt. Math. Publ. **53** (2012), 1–20

A COMPARISON OF LOCAL REDUCTION AND SAT-SOLVER BASED ALGEBRAIC CRYPTANALYSIS OF JH AND KECCAK

Peter Adamček — Marek Loderer — Pavol Zajac

ABSTRACT. Local reduction methods can be used to assess the resistance of cryptosystems against algebraic attacks. The assessment is based on the separation of the attack into polynomial-time reduction algorithm, and exponential time guessing and backtracking. This approach is similar to that employed by the DPLL algorithm that is used as a core of various modern SAT-solvers. In the article we show the application of this method to evaluate the strength of (reduced versions of) two chosen SHA-3 candidates: JH, and Keccak, respectively. We compare the complexity estimates with the behavior of the full search algorithm. We also compare the results based on the local reduction with the attack based on the use of SAT-solvers PrecoSAT, and CryptoMiniSAT, respectively.

1. Introduction

Let $F : \mathbb{Z}_2^n \to \mathbb{Z}_2^m$ be a one-way function. Informally, it is easy to compute $F(x)$ for any x (in time polynomial in n), but it is difficult to find any x such that $F(x) = y$. If the function F behaves as a random function, we need on average 2^m randomly chosen x's to find a preimage of y (or we can check all 2^n input options, if $n \leq m$). One-way functions play a central role in cryptography, where the preimage is usually something we want to keep secret, and the output is a public information. Simply said, a basic cryptanalytic problem is to recover the secret information using the publicly available data.

A lot of cryptanalytic problems can be formulated as a question of inverting a one-way function:

- given a plaintext-ciphertext pair, find the key of the block cipher;

2010 Mathematics Subject Classification: 94A60, 68Q17.
Keywords: algebraic cryptanalysis, local reduction, JH, Keccak.
This research was supported by grants APVV-0513–10, and APVV-0586–11.

- given a keystream of a stream cipher, determine the internal state;
- given a hash, find a preimage of the hash.

Most of the cryptanalytic research focuses on attacks based on statistics that require a lot of data. However, we consider a different situation, in which the attacker can only access a small number of data (e.g., a single encryption, or a single hash).

A function that can be efficiently evaluated (in polynomial time), can be described by a system of non-linear Boolean equations (polynomial in the size of the input). The unknowns in the system represent the input, intermediate, and output bits of the computation. After obtaining the output of the function F, we can substitute the values of the output bits, and compute the input bits by solving the equation system. This transformation is the core of the algebraic cryptanalysis, although the scope of algebraic cryptanalysis is much broader nowadays, see, e.g., [6]. Moreover, algebraic attacks can be combined with side--channel analysis. The correct complexity evaluation is critical when selecting suitable cryptographic functions for constrained solutions found in telemedicine systems such as [21].

The problem of solving a system of non-linear Boolean equations is NP-hard. We expect that (in general) its complexity is exponential in the number of unknown variables. However, there exist some classes of problems that are easy to solve even in polynomial time [16]. We note that in the instances derived from a problem of inverting a one-way function, the complexity can be upper bound by 2^n (the number of possible inputs) regardless of the number of variables in the equation system: once we choose all input bits, we can compute all intermediate values (and verify the output bits).

In cryptographic practice, it is very important to know what is the expected complexity of the given cryptanalytic problem, and how does it scale with the parameters (e.g., the number of rounds of the iterated block cipher). The theoretical complexity bounds (e.g., [19]) are usually based on random equation system models, not on a concrete ciphers or families of ciphers. On the other hand, the experimental results (mostly based on SAT solvers) are restricted to smaller instances, and are often hard to scale to larger systems.

In [23], we present an evaluation methodology based on local reduction that can be used to estimate the complexity of algebraic attacks. It is a generalization of our results from [24], [27]. In this paper we extend these results further. We demonstrate the use of the methodology in comparing the security estimates for two selected SHA-3 candidate functions: JH [22], and Keccak [3], respectively. We compare the estimated complexity results with the complexity of the real attack using small versions of the problem, and also compare the estimates with the results obtained with SAT solvers.

The article is organized as follows. In Section 2 we present the basic preliminaries: local reduction, and SAT-solver based algebraic cryptanalysis, respectively. Sections 3 and 4 contain the methodology and experimental results of the analysis of Keccak, and JH, respectively. Final Section 5 is used to discuss the results, and possible open problems and limitations of our approach.

2. Preliminaries

Let

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_m) &= 0, \\ f_2(x_1, x_2, \ldots, x_m) &= 0, \\ &\vdots \\ f_n(x_1, x_2, \ldots, x_m) &= 0 \end{aligned}$$

be a system of Boolean equations, denoted by S. Let each f_i depend only on unknowns from the set X_i, i.e., if $x_j \notin X_i$, then

$$f_i(x_1, x_2, \ldots, x_j, \ldots, x_m) = f_i(x_1, x_2, \ldots, x_j + 1, \ldots, x_m)$$
$$\text{for each} \quad \overline{x} = (x_1, x_2, \ldots, x_j, \ldots, x_m) \in \mathbb{Z}_2^m.$$

Let $|X_i| = l_i$. We say that S is l-sparse, if $l_i \leq l$ for each $i = 1, 2, \ldots, n$. Vector $\overline{x} \in \mathbb{Z}_2^m$ is a solution of the system S if it is a solution of each equation $f_i(\overline{x}) = 0$ in the system.

We note that if the equation $f_i(\overline{x}) = 0$ does not depend on variable x_j, then the value of the jth coordinate is irrelevant in determining whether $\overline{x}$ is, or is not, a solution of the equation. Thus, if we want to check whether $\overline{x}$ is a solution of $f_i(\overline{x}) = 0$, we only need to know the values of the coordinates of $\overline{x}$ corresponding to unknowns from X_i.

Equation $f_i(\overline{x}) = 0$ has at most 2^{l_i} possible solutions in variables from X_i. Let V_i denote the set of all such solutions. The pair (X_i, V_i), which we will call a symbol[1] [13], uniquely represents the equation $f_i(\overline{x}) = 0$ (and vice-versa). The whole system of equations can be stored as a set of symbols $S = \big\{(X_i, V_i);\ i = 1, 2, \ldots, n\big\}$. An l-sparse system can be stored (and enumerated) in at most $O(n2^l)$ bits (operations). Thus, for a (small) fixed l, the time and memory complexity of the symbol representation is polynomial in the system size.

For each vector $v \in V_i$ we can find 2^{m-l_i} vectors in the full m-dimensional solution space, which have the required fixed values in the coordinates given by X_i. Let V_i^* denote the set of vectors in the solution space corresponding to any of the vectors in V_i. A vector is a solution of system S, if it is a solution of each equation

[1] A symbol is a compressed representation of the underlying algebraic variety $\{\overline{x}; f_i(\overline{x}) = 0\}$.

in the system. Thus all solutions of the system S are given by $\bigcap_{i=1}^{n} V_i^*$. An effective algorithm to compute this intersection is Gluing [17], [18]. Complexity estimates for an improved version of Gluing on random equation systems are presented in [19].

Alternative methods that can be used to solve systems in symbol representation are Agreeing [15], or the method of syllogisms [25] extended by using guessing and backtracking. These methods implement in a more efficient way an older idea of the local reduction [28] (see Section 2.1), with the addition of a slightly different representation, and many optimizations known from the development of the efficient SAT solvers.

2.1. Local reduction

Let $S = \{(X_i, V_i)\}$ be a system of equations in the symbol representation. Let us consider a system with a single solution s. Then each V_i contains exactly one vector (s_i) that is a projection of the solution s into coordinates given by X_i. We can remove any vector other than s_i from V_i, and the new system will have the same solution s. We will call the process of removing vectors from V_i's (without changing the solution of the system) a local reduction. We say, that $S' = \{(X_i, V_i')\}$ is a reduced version of the system S, if S and S' have the same set of solutions, and if $V_i' \subset V_i$ for each i. We will denote[2] this by $S' \leq S$.

In the following, we will only consider two types of systems, which are most common in the algebraic cryptanalysis: systems with a single solution, and systems with no solution, respectively. If S does not have any solution, the process of local reduction leads to a removal of each vector in some V_i. In this case, we say, that the reduction found a conflict in the system. If S has a single solution, then there exists a reduced version of the system with $|V_i| = 1$ for each i. We say that such a system is completely reduced. Reconstructing the full solution of a completely reduced system is a trivial task.

A main goal of the reduction based approach to solving equation systems is to remove individual solutions from the system, until we find a completely reduced system (or a conflict). We do this by a combination of (polynomial) local reduction algorithm, and (exponential) guessing and backtracking algorithm.

A polynomial local reduction algorithm $\mathcal{R}$ is any algorithm that fulfills the following conditions:

(1) Its input, and output, are systems of equations in symbol representation. We denote them by S, and $\mathcal{R}(S)$, respectively.

(2) S, and $\mathcal{R}(S)$ have the same set of solutions.

(3) $\mathcal{R}(S) \leq S$.

[2]It is easy to see, that the reduction property induces a partial order on a set of systems with the same solution and the same X_i's.

(4) $\mathcal{R}\big(\mathcal{R}(S)\big) = \mathcal{R}(S)$.
(5) The (time and memory) complexity of $\mathcal{R}$ is polynomial in the size of S (number of equations/variables).

A typical local reduction algorithm is the spreading of constants[3]: if all vectors in some V_i have the same value in a selected coordinate (e.g., $x_1 = 0$), then we can safely remove all vectors from other symbols that have a different value in this coordinate (i.e., $x_1 = 1$).

In our experiments, we use a local reduction algorithm based on the method of syllogisms [28], [26]. This method uses projections to pairs of variables (subsets of X_i). If some combination of values does not occur in V_i, it must not occur in any other V_j. Thus we can remove the offending vectors (if any). Moreover, each missing combination can be written as a pair of (global) implications about the values of variables in the solution. For example, if $x_1 = 0, x_2 = 0$ is not present in V_1, then we know that $(x_1 = 0) \Rightarrow (x_2 = 1)$, and $(x_2 = 0) \Rightarrow (x_1 = 1)$, respectively. Using the syllogisms rule (transitivity of implications), it is possible to find (in polynomial time) even more implications that are not directly induced by the equation system, possibly leading to further reduction of the system. The process is repeated, until we cannot find any more information about the system, and cannot reduce the system further.

Although we have restricted $\mathcal{R}$ to polynomial time algorithms, there are still some classes of systems, that can be reduced completely using just $\mathcal{R}$ [16]. However, under the exponential time hypothesis [10] we may assume that there does not exist polynomial time $\mathcal{R}$, that can completely reduce all possible systems. Condition $\mathcal{R}\big(\mathcal{R}(S)\big) = \mathcal{R}(S)$ means, that the system can only be reduced once (using the same algorithm). If $\mathcal{R}$ does not produce a solution, we can continue the process with the introduction of guesses (and a backtracking mechanism to compensate for incorrect guesses).

The basic alternative is to try to estimate the value of some variable (e.g., $x_1 = 0$). Then we can remove all vectors from V_i's that are not consistent with the guess. If the guess is correct, we get a reduced version of the system by a different algorithm than $\mathcal{R}$. Otherwise, we get an inconsistent system (assuming a single solution). In both cases, we can continue the process recursively until a solution/conflict, is found. In case of conflict, we must backtrack the algorithm and check the other possible values for the guessed variables. For each guess the number of options to verify is doubled, so the complexity of the process is 2^g, where g is the number of variables guessed until solution/conflict is found.

Another type of guessing and backtracking is based directly on the symbols in the system[4]. Without a loss of generality, let us focus on $V_1 = \{v_{1,1}, v_{1,2}, \ldots v_{1,N_1}\}$.

[3]Spreading of constants corresponds to Unit propagation in SAT solvers, see Section 2.3.
[4]This corresponds to Gluing2 algorithm from [14].

Under the assumption that there is at most one solution of the system, we can replace V_1 by $V_1' = \{v_{1,1}\}$ (a guess of the solution). If we guessed correctly (with probability N_1^{-1}, if we assume that each of the vectors is equally likely a projection of the global solution), we get a reduced system $S' \leq S$. In the other $N_1 - 1$ cases, we get a system that does not have a solution. We can again apply the reduction and guessing recursively to S', and if we get a conflict, we backtrack the algorithm, and try another guess(es). The complexity of guess and backtracking in this case is (approximately) $O(N_1 N_2 \cdots N_k)$, where k is the number of symbols involved in the guessing, and $N_1, N_2, \ldots, N_k$ are the numbers of solutions in each symbol we must verify. This corresponds to a bit complexity[5]

$$b = \sum_{i=1}^{k} \log_2 N_i.$$

Both the number of variables g, and value b strongly depend not only on the system, but also on the set of variables/symbols chosen for guessing [24]. We call the algorithm that chooses the sequence of variables/symbols a guessing strategy. There are several guessing strategies (see Section 2.2) that can be used, producing different results for different types of ciphers [27], [23].

Unfortunately, it is difficult to (theoretically) compute k, and N_i's for a given system, and reduction algorithm $\mathcal{R}$. In practice, if we just want to estimate the security of some cipher, we do not need to implement the full guess and backtracking algorithm (with the exponential complexity). Instead, we follow the algorithm with:

- random guesses, until we get a conflict, i.e., an empty system — an uninformed evaluation;
- always correct guesses, until we get a completely reduced system — an oracle-based evaluation.

The oracle-based evaluation is only possible if we know the solution of the system a-priori (we only want to estimate the complexity). A real-world attacker can generate random instances of the cryptanalytic problem with known solution to tune up the parameters of the full attack.

It is important to note that the complexity estimates using uninformed strategies can be much lower than the oracle based estimates [24]. We can imagine the guess and backtracking algorithm as a traversal of a tree of possible options. A lower estimate in uninformed case means that the tree is not "wide", and we can get conflicts sooner than we can find the solution of the system.

[5]Bit complexity b for cryptanalytic problems corresponds to complexity $O(2^b)$ of the total search, or to chance $c2^{-b}$ of the random guess to be a correct solution of the problem.

In practice, this also means that the real complexity might be lower than expected by the oracle based estimate. The oracle based estimate is good for the worst-case scenarios (this can also be seen in Section 4.3).

2.2. Guessing strategies

We use three main guessing strategies in our experiments:

Rand: a random selection of symbols. If the solver/evaluator needs a new guess, it chooses a symbol uniformly at random from all symbols that have at least 2 solutions. This strategy can be used to provide an estimate for the least sophisticated attack possible (worst-case average complexity).

Guess: selects randomly a symbol from the set of all symbols with maximal $2^{l_i}/|V_i|$, where l_i is the number of variables with unknown value in a given symbol. This strategy tries to maximize the information gained by each guess, as each guess fixes all l_i variables with probability of success $1/|V_i|$ [24].

Impact2: is based on guessing variable values directly. It guesses the value of a variable, which is active in the highest number of unreduced symbols (symbols with at least 2 solutions) in the system. We note, that the variables which have a fixed value in any symbol are removed from this symbol during the local reduction in each iteration of the algorithm. This strategy tries to maximize the impact of each guessed variable in the system, because we expect to remove half of the solution in each symbol where the guessed variable is active.

We note that it is possible to define many different guessing strategies. Our choice of strategies is for evaluation is limited to generic strategies (independent of the type of the problem). A real-world attacker can construct a strategy that copies some (presumed) flaws of the cipher, or use some other knowledge in the attack to make it more effective.

2.3. SAT solvers

A logic expression C is called a k-CNF, if $C = \bigwedge_{i=1}^{n} C_i$, and $C_i = \bigvee_{j=1}^{k} L_{i,j}$, where $L_{i,j} \in \{x_1, \ldots, x_m, \neg x_1, \ldots, \neg x_m\}$. C_i's are called clauses, and $L_{i,j}$'s literals. k-CNF-SAT problem is a decision problem asking whether there exists an assignment of truth values to x_i's such that C is satisfied (evaluates to true). We delegate the answer to this problem to a black-box tool, a SAT solver.

If the answer is positive (C can be satisfied), we also expect that the SAT--solver produces a proof, i.e., the actual assignment of the values such that C evaluates to TRUE. In general, the running time of a SAT-solver is exponential in the problem size (n, m). However, there are instances, where the SAT-solver can provide an answer very quickly [5], [7].

SAT solvers are important in algebraic cryptanalysis: we encode a cryptanalytic problem in k-CNF form, and ask for the proof of satisfiability (SAT). We run a SAT-solver to provide the proof. Alternatively, we try to guess some part of the solution (e.g., some of the key bits), and ask the solver to decide whether the problem is unsatisfiable (UNSAT, indicating an incorrect guess), or provide a SAT-proof (and thus the whole solution).

Most of the SAT-solvers used in algebraic cryptanalysis are based on the DPLL algorithm [8]. The main idea of DPLL is to use two operations Unit propagation, and Pure literal elimination, respectively. They are used to simplify the CNF formula (following certain rules that can be quickly evaluated). These operations play the same role as the local reduction algorithm $\mathcal{R}$ (they can be rewritten in symbol form, e.g., the Unit propagation corresponds to the spreading of constants). If it is not possible to continue with these two operations, the solver makes a decision: it assigns a truth value to some variable, and encodes it as a new clause added to the CNF formula. It then tries to prove a new formula with the added decision clause (recursively). If it fails, it tries the other possible assignment. Only if both options are exhausted, the SAT solver can say that the formula is unsatisfiable.

The total number of decisions during DPLL algorithm (D) is one of the values reported by the solver. This value is typically log-normal [2], so the average value of $(\log_2 D)$, can be used as the estimates of the bit complexity of the SAT-solver based algebraic attack.

2.4. Symbol representation and CNF-SAT

There is a correspondence between a symbol representation and k-CNF representation of the cryptanalytic problem. It is possible to convert any of these representations in polynomial time to the other one.

Let X_i denote the set of active variables in clause C_i (if $L_{i,j}$ is either x_1 or $\neg x_1$, then x_1 is active). There are exactly $2^k - 1$ possible assignments of truth values for x_j's that allows C_i to be satisfied. Let us encode truth values by 0, and 1, respectively. Let V_i be a set of vectors (indexed by variables in X_i) of satisfying assignments for C_i. Then (X_i, V_i) is a symbol representing a (k-sparse) Boolean equation. Solutions of this equation are in one-to-one correspondence with all proofs of C_i.

Let $S = \{(X_i, V_i)\}$ be a (k-sparse) system of equations, with individual symbols (X_i, V_i) induced by clauses C_i of $C = \bigwedge_{i=1}^{n} C_i$. It is easy to see that each solution of the system S is in one-to-one correspondence with positive assignments for C. Given a solver for S, it is possible to use it to decide k-CNF-SAT, and provide a proof if the answer is SAT.

Let $S = \{(X_i, V_i)\}$ be a k-sparse equation system. For the sake of simplicity, let $|X_i| = k$ for each i (it is easy to generalize to any k_i). Let $v \in Z_2^k \setminus V_i$.

There is a single clause that is not satisfied by v, which we denote by C_v. For example, if $X_i = \{x_1, x_3, x_5\}$, $v = 011$ (TRUE is encoded as 1), then v is not a solution of $x_1 \vee \neg x_3 \vee \neg x_5$. It is easy to see that V_i corresponds to all positive assignments of

$$C_{(X_i,V_i)} = \bigwedge_{v \in \mathbb{Z}_2^k \setminus V_i} C_v \,.$$

Vector w is a solution of the system S, if it is a solution of each equation in the system. This means w corresponds to a positive assignment of

$$C_S = \bigwedge_{i=1}^{n} C_{(X_i,V_i)}.$$

A SAT-solver thus can be used to decide, whether system S has a solution or not, or to find a single solution of the system (if the solver provides a proof for SAT instances). To find all solutions of S we need a solver that can enumerate all possible positive assignments of C_S. However, in algebraic cryptanalysis, we are usually content with a single solution, or with a decision version of the problem.

We note that the above transformation between the symbol representation, and the CNF representation, is quite ineffective. In one direction, it produces many large symbols, with pairs of symbols with many common variables. In practice, we can glue such symbols together to produce a more compact representation of the system without a significant impact on the system size. In the other direction, the conversion produces many clauses with common variables (different only in polarity of literals). These can be simplified in a similar way, e.g., by using the rule $(x_1 \vee C) \wedge (\neg x_1 \vee C) = C$.

3. Evaluation of the Keccak f-function

Our first aim was to compare the results obtained by the local reduction with the existing SAT-solver based attacks. As a reference, we have reproduced the attack of Morawiecki and Srebrny [12] on the SHA-3 candidate Keccak [3].

3.1. Keccak

Keccak is a family of sponge functions. Our experiments are limited to the hash function Keccak [24], [26] which is based on permutation Keccak-f [1] (simplified to f in the following). Our goal is to find the preimage of a hash, which was created using a single application of the function f. More precisely, we compute the inversion of the 24-bit value $h = \lfloor f(m||0) \rfloor_{24}$, where m is an unknown message (at most 24 unknown bits). In the experiments, we only use

messages with 16 unknown bits. The dependence on the size of the unknown message was also explored (see Figures 2, and 3, respectively).

Permutation f is defined as a sequence of 5 operations (one of them non-linear), repeated in n_r rounds. We do not go into details of these operations (they can be found in [3], and in various other sources), as they are not relevant from the evaluation point of view. The goal of the evaluation is just to estimate how the complexity of algebraic attacks is changed if we change n_r.

3.2. Overview of the experiments

The experiments were conducted as follows:

(1) CNF files that encode the problem were prepared according to [12] for $n_r = 1, 2, \ldots, 14$. Inputs and outputs of f were also encoded as unknowns.

(2) Each CNF was converted to a symbol representation using the method described in Section 2.4. Symbols corresponding to clauses with the same variables were glued together.

(3) For each n_r, and for each $b = 2, 4, \ldots, 16$: 30 messages were hashed, and the encoded outputs were added into CNF, and symbol-encoded instances of the problem. The message bits were stored to provide a guessing oracle.

(4) For each instance of the problem, the complexity was evaluated using the SAT solver PrecoSAT version 576 [4], and using the sylog software [26] with 3 guessing strategies: Rand, Guess, Impact2 (described in Section 2.2).

The parameters of the equation system/CNF are summarized in Table 1. Complete description of the experiments as well as additional results are provided in [1].

3.3. Experimental results

The experimental results of the evaluation are summarized in Figure 1. Messages with 16 unknown bits were used, so the total complexity is bounded by 16 bits. Number of rounds must be at least 4 (out of 14 rounds of Keccak-f [1]) before this bound is reached. After four rounds, evaluation strategy Guess, as well as the results from the SAT-solver follow this bound independently of the size of the system. Other two strategies are not as successful in identifying the solution.

A further comparison of the selected guessing strategies, and the SAT-solver is provided in Figures 2, and 3, respectively. We compare the estimated security as we change the number of unknown bits of the message. This number is also a complexity bound for the brute force attack.

Figure 2 shows the situation for $n_r = 2$. In this case, the reported/estimated complexity is lower than brute force attack, so the function is insecure. Local reduction with Impact2 strategy leads to the best attack (in average).

TABLE 1. Number of variables, symbols, and clauses in the instances encoding the inversion of KECCAK-$f[1]$ reduced to n_r rounds.

Rounds	Variables	Symbols	Clauses
1	200	199	730
2	472	661	2136
3	804	1250	3772
4	1007	1541	4747
5	1525	2437	7366
6	1754	2803	8346
7	2206	3566	10743
8	2469	3991	11949
9	2910	4787	14218
10	3128	5099	15184
11	3526	5776	17214
12	3834	6305	18575
13	4198	6885	20603
14	4461	7336	21829

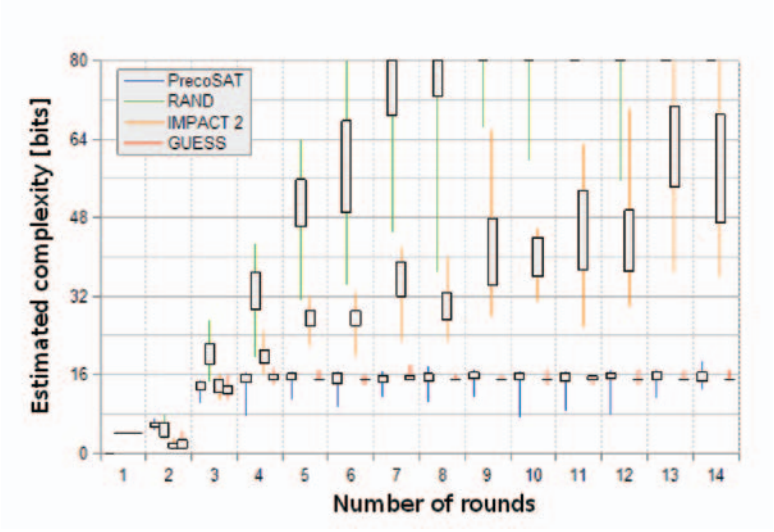

FIGURE 1. Comparison of selected guessing strategies for local reduction with the bit complexity of PrecoSAT solver based results on round reduced Keccak preimage problem.

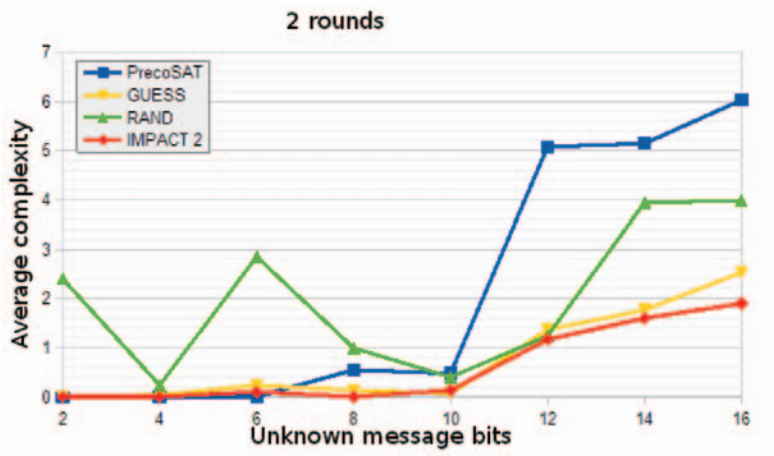

FIGURE 2. The growth of average complexity of local reduction strategies and PrecoSAT solver for 2-round Keccak when increasing the unknown message bits.

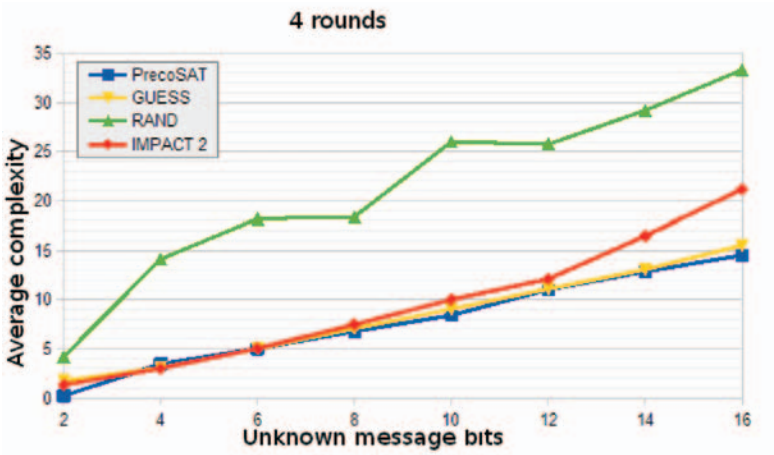

FIGURE 3. The growth of average complexity of local reduction strategies and PrecoSAT solver for 4-round Keccak when increasing the unknown message bits.

Figure 3 shows the situation for $n_r = 4$ that is also typical for a higher number of rounds. The average complexity reported by the SAT-solver as well as estimated by the Guess strategy copies the expected complexity based on the number of unknown message bits. Other strategies depend on the size of the whole system, and this leads to overrating the complexity of the attack.

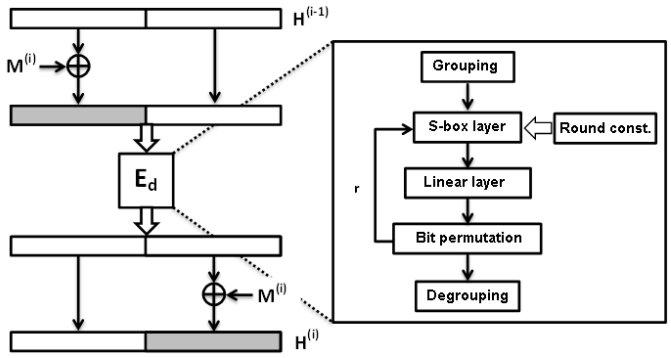

FIGURE 4. Compression function F_d of the JH hash function.

We find that this behavior of local reduction is typical, if the complexity of the attack is comparable with the brute force. The evaluation strategy can either identify the key bits (inputs), and solve the system with the expected complexity independent of the system size; or the estimate is higher, and depends on the system size (a typical example is the random guessing strategy Rand).

4. Evaluation of JH

In this section we provide an analysis of the SHA-3 candidate JH [22]. We start by using the local-reduction based evaluation. To simplify the interpretation of the results (in comparison to Section 3), we take only the value provided by the best guessing strategy. We also compare the estimate with the actual number of reductions required for the full attack (on a small version of JH). Finally, we compare the results obtained by the local reduction with the complexity reported by two SAT-solvers: PrecoSAT, and CryptoMiniSAT2, respectively.

4.1. JH

JH is an iterative hash function based on the generalized AES design methodology. Its compression function F_d uses as a core a bijective function E_d, which again is a composition of several round transformations (see Figure 4). Each round consists of an application of S-boxes, a linear transformation, and a permutation layer. Parameter d stands for the dimension of a block of bits. This parameter affects the size of the state, the number of message bits, and the (recommended) number of rounds of the function E_d. For a more detailed information we refer the reader to a specification of JH [22].

TABLE 2. Parameters used by the generator of equations.

Dimension d	State length	Message length	Number of rounds n_r	Number of variables
3	32	16	$1, \ldots, 12$	$80 + 32n_r$
4	64	32	$1, \ldots, 18$	$160 + 64n_r$
5	128	64	$1, \ldots, 24$	$320 + 128n_r$

In our experiments, we explore only small versions of JH function, where parameter d has values $d = 3$, $d = 4$ and $d = 5$ (the SHA-3 candidate JH has $d = 8$). We want to find a preimage of a hash that requires just one application of compression function F_d (a single application of bijective function E_d). JH's padding rule adds one more block, so the message after the padding passes through the compression function F_d at least twice. However, if we assume that the attacker knows the whole final state[6] of JH, and the length of secret message, he can invert the second F_d, and compute the state after the first application of F_d.

In this paper we only consider inverting a single application of function F_d, assuming the whole output is known, and only message bits are unknown (so we get a similar situation to Section 3). For the sake of simplicity, we simply say we want to "invert JH". The main goal of the experiments is to estimate how the complexity of algebraic attacks is changed, if we change n_r, and d. More details of the experiments with the local reduction, as well as experiments in different models are covered in details in [11].

4.2. Overview of the experiments

The experiments were conducted as follows:

(1) The problem of inverting JH was encoded as a local reduction problem using symbol representation. The intermediate, message, and output bits were encoded as unknowns. In practice, a software generator of equations prepared files with the equation system, given parameters n_r, and d, respectively (see Table 2).

(2) For each $d = 3, 4, 5$, and $n_r = 1, 2, \ldots, 6(d-1)$: 100 messages were hashed, and assignments of the known outputs of F_d were added to the stored equation systems. The original message bits were stored to provide a guessing oracle.

[6]In practice, the hash value is produced by truncating the final state.

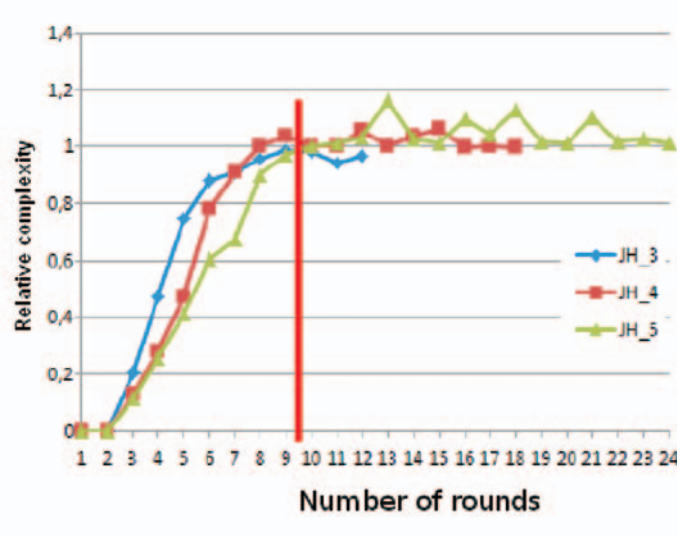

FIGURE 5. The average estimated complexity of the best guessing strategy for the local reduction attack for various versions of JH.

(3) For the latter experiments with SAT solvers, each file was also converted to CNF file (using the algorithm in Section 2.4)[7].

(4) For each instance of the problem, the complexity was evaluated using the sylog software [26] with guessing strategies: Rand, Guess, Impact2 (described in Section 2.2).

(5) Selected instances of the problem were solved by the full guess and backtrack algorithm, to compare the estimated and real complexity of the attack. A modified version of the sylog software was used. In this version, the algorithm is not stopped immediately after the solution is found, but instead it explores all branches of the search tree (branch either ends in a conflict, or in a solution, which is either the original message or a second preimage). In this case, we measure the bit complexity as $\log_2 R$, where R is the number of applications of the reduction algorithm $\mathcal{R}$.

(6) Finally, the CNF files were used as the input to SAT solvers PrecoSAT version 576 [4], and CryptoMiniSAT2 version 2.9.4 [20], respectively.

4.3. Experimental results

The experimental results of the complexity evaluation are summarized in Figure 5. Messages with 16, 32 and 64 unknown bits were used (depending

[7]Unlike in Section 3, we start from the symbol representation, as it is easier to construct from a given encryption scheme.

on the parameter d, $len = 2^{d+1}$). The values are normalized, so that value 1 (on y-axis) denotes the bit complexity of the brute force attack (len). We show only the best average complexity (this was achieved mostly by Impact2, but in some cases also by Guess). We can quickly compare various versions of the hash function, and estimate the overall trend. We can see that the average expected complexity of attack on JH with $d = 3$ is below the brute force bounds. According to the overall trend in respect with d, we can expect that full JH (with $d = 8$) also needs at least 10 rounds (out of 42 rounds) to reach the expected security against local reduction attacks.

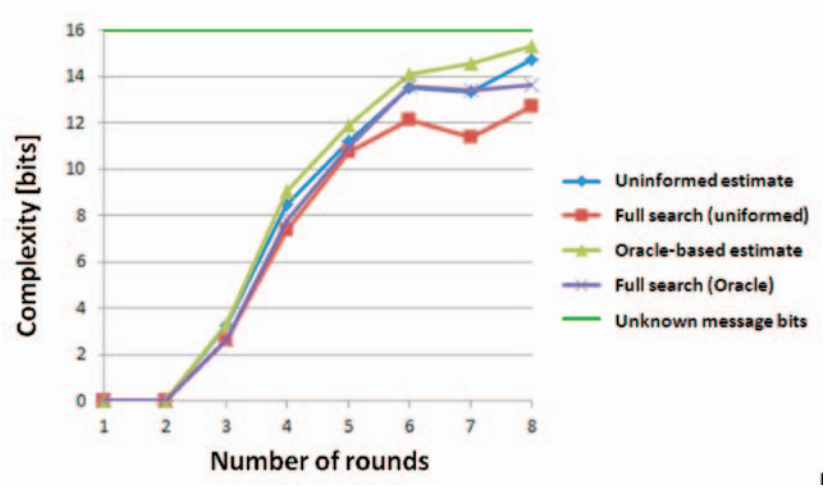

FIGURE 6. A comparison of the complexity estimate with the complexity of the full attack.

Figure 6 shows the comparison of the estimated complexity with the complexity of the full guess and backtrack local reduction attack. Due to the complexity of the full attack, we only use 16 bit message and JH in version with $d = 3$. Full attack used the search tree based on the order of variables given by the guessing strategy Impact2 (which is variable based, instead of symbol based strategies Rand, and Guess, respectively). One set of estimates was oracle-based (a correct guess was found during the evaluation), and the other one was uninformed (the complexity estimate was based on the first conflict). The measured bit complexity of the full attack is slightly lower than the estimated complexity, but it copies the general trend (in respect to n_r). We note, that the full search based on the uninformed strategy sometimes fails to find the original message (only 31 % chance to find the solution for $n_r = 8$). This is due to the fact, that we only use guessing for those variables, that were marked by the evaluation.

In practice, the attacker will add an additional variable(s) to guess, if the solution was not found by examining the symbols pointed by the evaluation. In this case, his results will be similar to the results provided by the oracle-based set of symbols/variables.

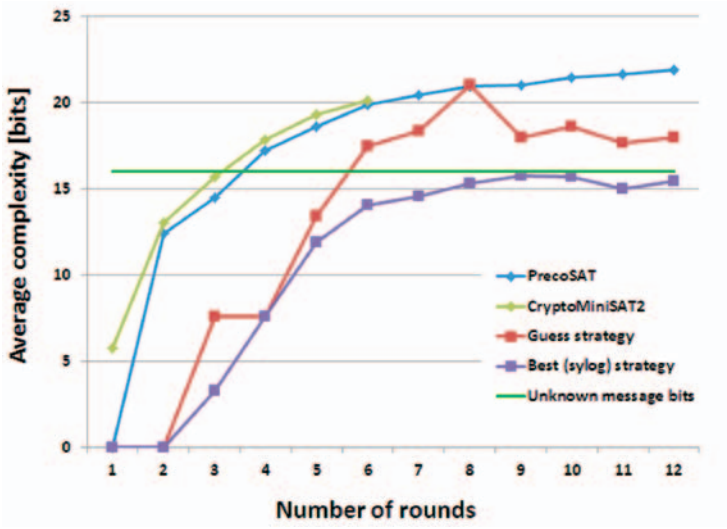

FIGURE 7. Comparison of the complexity estimate with the complexity of SAT-solvers.

Finally, we compare the results of the evaluation with the complexity of the SAT-solver based algebraic attack. Our observations are summarized in Figure 7. Again, messages with 16 bits were used ($d = 3$). The complexity reported by the SAT-solvers is higher than the estimates based on the local reduction, and after 4 rounds they are higher than the brute force attack. The trend provided by both SAT-solvers is the same[8]. On the other hand, a local reduction attack with the guessing strategy Guess needs six rounds to reach the complexity of the brute-force attack. The best average strategy (which is Impact2, if the result is lower than Guess) gives lower complexity estimate than SAT-solvers, but again, the trend with respect to n_r is similar.

[8]In practice, SAT solvers also spend a lot of time by reducing a large CNF, which is a result of the conversion from the symbol representation. Especially, as we do not encode XOR clauses for CryptoMiniSAT in a special way, the actual running times of CryptoMiniSAT including all precomputation were too large. Because the bit complexity was similar to that of the PrecoSAT, we decided not to run the experiments over 6 rounds also with CryptoMiniSAT.

5. Conclusions

The main goal of the paper was the use of the local reduction in the evaluation of the cipher designs. We can base the final comparison on the number of rounds that can be broken by a local reduction based attack. In case of KECCAK-f [1], we can break 3 rounds out of 14. The structure of KECCAK family supports the hypothesis that the results for the full SHA-3 candidate using KECCAK-f [6] will similarly be weak until it has at least 4 rounds (out of 24), giving security margin $24/4 = 6$. On the other hand, JH with $d = 3$ seems to be broken, and we expect that full JH with $d = 8$ needs at least 10 rounds (out of 42), giving security margin $42/10 = 4.2$. We should note that the attacks in practice can be better than expected, as the evaluation targets a case of generic algebraic attacks. If we can attack a version of the cipher with the higher number of rounds, it can be a potentially easier target for a more advanced algebraic attack based on the algebraic complexity reduction [6].

Our results indicate that the estimate of the complexity based on the oracle-based evaluation is similar to actual number of reduction required to explore the whole search space using the guess and backtrack algorithm. The best average expected complexity has also a similar trend with respect to the number of rounds as the number of decisions reported by the SAT-solver during the attack. Although the expected number of reductions is lower, SAT-solvers are faster in practice (due to the easier "reduction steps") than the solver implementing the method of syllogisms. This can however change after more efficient local reduction solvers are developed (or by the development of a special purpose hardware like [9]).

The experiments (see also [24], [27]) show that none of the presented guessing strategies can be considered optimal in the sense that it produces a lower bound for the complexity of the generic local reduction based algebraic attack (a natural "upper bound" is provided by Rand strategy). It is interesting to note that Guess strategy achieves better results with KECCAK, and Impact2 strategy with JH, respectively. All the strategies presented so far are relatively simple to speed up the computation. However, even a guessing strategy that is relatively complex can be used in an offline evaluation phase (which is fast), and the actual full-search attack can use the order of variables/symbols precomputed by the evaluation.

Acknowledgements. The authors are grateful to an anonymous referee for helpful comments.

REFERENCES

[1] ADAMČEK, P.: *Cryptanalysis of Keccak*. Master's Thesis. Slovak University of Technology in Bratislava, 2012. (In Slovak)

[2] BARD, G. V.: *Algebraic Cryptanalysis*, Springer, Dordrecht, 2009.

[3] BERTONI, G. — DAEMEN, J.— PEETERS, M.— VAN ASSCHE, G.: *The Keccak reference*, Tech. Report, 2011, http://keccak.noekeon.org/Keccak-reference-3.0.pdf.

[4] BIERE, A.: *Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT Race 2010*, Tech. Report, Institute for Formal Models and Verication, Johannes Kepler University, Linz, Austria, 2010.

[5] COURTOIS, N. T. — BARD, G. V.: *Algebraic cryptanalysis of the data encryption standard*, in: Cryptography and Coding (S. Galbraith, ed.), Lecture Notes in Comput. Sci. Vol. 4887, Springer Berlin, 2007, pp. 152–169, http://dx.doi.org/10.1007/978-3-540-77272-9_10.

[6] COURTOIS, N. T.: *Algebraic complexity reduction and cryptanalysis of GOST*, Cryptology ePrint Archive, Report 2011/626, http://eprint.iacr.org/.

[7] COURTOIS, N. T. — BARD, G. V. — WAGNER, D.: *Algebraic and slide attacks on KeeLoq*, Cryptology ePrint Archive, Report 2007/062, http://eprint.iacr.org/.

[8] DAVIS, M. — LOGEMANN, G. — LOVELAND, D.: *A machine program for theorem-proving*, Commun. ACM **5** (1962), 394–397, http://doi.acm.org/10.1145/368273.368557.

[9] GEISELMANN, W. — MATHEIS, K. — STEINWANDT, R.: *PET SNAKE: A special purpose architecture to implement an algebraic attack in hardware*, Cryptology ePrint Archive, Report 2009/222, http://eprint.iacr.org/.

[10] IMPAGLIAZZO, R. — PATURI, R.: *The Complexity of k-SAT*, in: Computational Complexity, Annual IEEE Conference 1999, IEEE Computer Society, Los Alamitos, CA, USA, pp. 237–240.

[11] LODERER, M.: *Cryptanalysis of JH function*. Master's Thesis, Slovak University of Technology in Bratislava, 2012. (In Slovak)

[12] MORAWIECKI, P.— SREBRNY, M.: *A SAT-based preimage analysis of reduced KECCAK hash functions*, Cryptology ePrint Archive, Report 2010/285, 2010, http://eprint.iacr.org/.

[13] RADDUM, H.: *Cryptanalytic results on TRIVIUM*. Tech. Report 2006/039, eSTREAM, ECRYPT Stream Cipher Project, 2006.

[14] RADDUM, H.— SEMAEV, I.: *New technique for solving sparse equation systems*, Cryptology ePrint Archive, Report 475/2006, http://eprint.iacr.org/2006/475.

[15] SCHILLING, T. — RADDUM, H.: *Solving equation systems by agreeing and learning*, in: Arithmetic of Finite Fields (M. Hasan, T. Helleseth, eds.), Lecture Notes in Comput. Sci. Vol. 6087, Springer, Berlin, 2010, pp. 151–165. http://dx.doi.org/10.1007/978-3-642-13797-6_11.

[16] SCHILLING, T. — ZAJAC, P.: *Phase transition in a system of random sparse Boolean equations*, Tatra Mt. Math. Publ. **45** (2010), 93–105.

[17] SEMAEV, I.: *On solving sparse algebraic equations over finite fields*, Des. Codes Cryptography **49** (2008), 47–60.

[18] SEMAEV, I.: *Sparse algebraic equations over finite fields*, SIAM J. Comput. **39** (2009), 388–409.

[19] SEMAEV, I.: *Improved agreeing-gluing algorithm*, Cryptology ePrint Archive, Report 140/2010, http://eprint.iacr.org/2010/140.

[20] SOOS, M.: *CryptoMiniSat2*, 2012, `http://www.msoos.org/cryptominisat2/`.

[21] VALKY, G.— LEHOCKI, F.: *Modern approach in multiple patients ECG monitoring*, in: BHI, IEEE, 2012, pp. 131–134.

[22] WU, H.: *The hash function JH*, Submission to NIST (round 3), (2011), `http://www3.ntu.edu.sg/home/wuhj/research/jh/jh_round3.pdf`.

[23] ZAJAC, P.: *Local reduction in evaluation of cipher security*, 2012 (preprint).

[24] ZAJAC, P.: *Solving trivium-based Boolean equations using the method of syllogisms*, Fund. Inform. **114** (2012), 359–373.

[25] ZAJAC, P.: *On the use of the method of syllogisms in algebraic cryptanalysis*, in: Proceedings of the 1st Plenary Conference of the NIL-I-004, University of Bergen, 2009, pp. 21–30.

[26] ZAJAC, P.: *Implementation of the method of syllogisms*, 2010 (preprint).

[27] ZAJAC, P. — ČAGALA, R.: *Local reduction and the algebraic cryptanalysis of the block cipher GOST*, Period. Math. Hungar. **65** (2012), 239–255.

[28] ZAKREVSKIJ, A. — VASILKOVA, I.: *Reducing large systems of Boolean equations*, in: 4th International Workshop on Boolean Problems, Freiberg University, 2000, pp. 21–22.

Received October 18, 2012

Institute of Computer Science and Mathematics
Faculty of Electrical Engineering and
Information Technology
Slovak University of Technology
Ilkovičova 3
SK–812-19 Bratislava
SLOVAKIA

E-mail: peter.adamcek@gmail.com
marek.loderer@stuba.sk
pavol.zajac@stuba.sk