






Measuring the Effectiveness of SAT-Based Guess-and-Determine Attacks in Algebraic Cryptanalysis

Andrey Gladush¹ , Irina Griбанова² , Viktor Kondratiev³ ,
Artem Pavlenko³ , and Alexander Semenov³  

¹ Special Technological Center, St. Petersburg, Russia

² Matrosov Institute for System Dynamics and Control Theory, Irkutsk, Russia

³ ITMO University, St. Petersburg, Russia

`alex.a.semenov@itmo.ru`

Abstract. This paper studies the problem of algebraic cryptanalysis where state-of-the-art SAT solvers are used to invert some cryptographic function. We define a new metric of the hardness of CNF formulas that encode the corresponding cryptanalysis problems. The introduced metric is similar to the well-known tree-like metrics used in the theory of propositional proofs. However, unlike the latter, the new metric can be effectively estimated in application to specific cryptographic functions. The corresponding approach combines the Monte Carlo method and metaheuristic black-box optimization algorithms. The proposed algorithms require a large amount of computational resources, and for their experimental evaluation we used a supercomputer. In the experiments, we applied the proposed metrics to construct estimations of guess-and-determine attacks on the compression function of the well-known MD4 cryptographic hash algorithm.

Keywords: Algebraic cryptanalysis · Boolean Satisfiability Problem (SAT) · SAT solvers · Guess-and-determine attacks · Inverse Backdoor Set (IBS)

1 Introduction

The present paper studies the application of parallel algorithms for solving the Boolean satisfiability problem (SAT) to the problems of algebraic cryptanalysis. In particular, we introduce new metrics that make it possible to estimate the complexity of SAT-based guess-and-determine cryptographic attacks [6]. The problem of constructing estimations of this kind is reduced to the optimization problem of the special fitness function, which is defined at the points of a Boolean hypercube. This fitness function is a black-box function whose values are calculated using the Monte Carlo method. To minimize this function, we use metaheuristic algorithms implemented in form of an MPI application. All computational experiments are carried out on a supercomputer.

The Boolean satisfiability problem (SAT) is a combinatorial problem with an extremely wide range of practical applications. Like many other NP-hard problems, SAT is not always difficult in practice, and for many of its particular cases can be solved quite effectively using various additional techniques and heuristics. As said above, we use state-of-the-art SAT solving algorithms in application to algebraic cryptanalysis. In more detail, we construct the so-called guess-and-determine attacks on some cryptographic functions using SAT solvers. In this context, we follow [6] and many other works in which SAT solvers are applied to algebraic equations describing the calculation process of the considered cipher.

Roughly speaking, the idea of a guess-and-determine attack consists in choosing (guessing) bits from a certain set, the substitution of which greatly simplifies the original cryptanalysis problem. Such a set is called a *guessed bits set*. Various methods can be used to solve problems weakened by such substitutions. For example, a substitution of bits from some guessed bits set to the Multivariate Quadratic (MQ) system [22] may turn it into a linear system. However, examples of successful attacks of this kind are very rare (see, e.g., [4]). The approach with a wider area of applications implies a reduction of the considered cryptographic problem to some combinatorial problem, the algorithmic base of which is well developed. As G. Bard notices in [6], SAT is a good example of such a problem. In application to the SAT encodings of cryptanalysis problems, guess-and-determine strategies can also be defined quite naturally.

The main problem that arises when using SAT in cryptanalysis is that the runtime of a SAT solver on a specific formula is hard to predict [17]. There are a number of serious studies of possible measures that can be used to evaluate the hardness of concrete formulas w.r.t. concrete algorithms for solving SAT. Below we rely on the results of [5], in which various approaches to measuring tree-like metrics of the hardness of Boolean formulas are studied. Specifically, in this paper, several different tree-like metrics used in propositional proof complexity are unified. The most important result of [5] for our case is the conclusion about the relationship between the tree-like metric and the notion of a Backdoor set (concretely, we mean the Strong Backdoor Set), presented in [37]. This notion in the sense of the idea is very close to that of a guessed bits set. The Strong Backdoor Set (SBS) is a set of variables such that the substitution of any values to the corresponding variables in the original formula results in a formula for which SAT is solved by some polynomial algorithm A . As we will see below, only extremely small SBSs can give a gain in complexity, but such situations are very rare in practice.

The approach in which we do not require the algorithm A to have polynomial complexity turns out to be more practical. In such a case, A can be an arbitrary complete algorithm for solving some NP-hard problem, for example, a SAT solver. This is exactly the approach used in a number of works on the decomposition of hard SAT instances, including applications in cryptanalysis [31–33, 38]. In fact, in these papers, it is shown that one can estimate the complexity of Boolean formulas using SBS generalizations for the case when A is a complete SAT solver. Specifically, in such cases, a special technique that

combines the stochastic Monte Carlo method and metaheuristic optimization is used. In [34] there is presented a class of SAT-based cryptographic attacks that uses the so-called Inverse Backdoor Sets (IBS). For such backdoors, the complexity estimations of the corresponding attacks have convincing guarantees of accuracy. This allows one to compare the effectiveness of attacks based on an IBS with the effectiveness of other attacks, and in a number of cases, IBS-based attacks have the best-known effectiveness.

In the present paper, we combine ideas from [5] with ideas from [34] and propose new tree-like metrics of the hardness of SAT instances that encode the inversion problems of cryptographic functions. For these metrics, it is possible to construct estimations using probabilistic algorithms similar to those used in [34]. We construct such estimations for the functions considered in [18–20]. For this purpose, we use a metaheuristic optimization of a special fitness function calculated on a computing cluster.

As a result, we construct SAT-based guessed-and-determine attacks on reduced-round versions of the compression function of the well-known MD4 hashing algorithm. Namely, we mean the functions of the kind $MD4-k$, where k is the number of steps of the base algorithm ($MD4-48$ corresponds to the complete-round version of the considered function). In the computational part of our work, we present non-trivial attacks on the functions $MD4-43$, $MD4-45$ and $MD4-47$. The estimations of the hardness of these attacks are significantly smaller than those of brute-force attacks for these functions.

2 Preliminaries

All variables considered below are Boolean variables, i.e., they take values from $\{0, 1\}$. Assume that $X = \{x_1, \dots, x_n\}$ is a set of Boolean variables. Then the Boolean formula F is an expression constructed with respect to special rules over the alphabet including X , brackets, and special symbols called logical connectives. The simplest Boolean formulas of the kind x or $\neg x$ are called *literals* (here \neg is the negation connective). The Boolean Satisfiability Problem (SAT) is a problem that requires one to determine for an arbitrary Boolean formula F whether there exists such an assignment α of variables from X , the substitution of which [9] to F results in 1 (true). If such an α exists, then it is called a *satisfying assignment*, and the formula F is called *satisfiable*. Otherwise, F is called *unsatisfiable*. SAT is NP-complete [11, 25] in the decision variant and NP-hard in its search variant (where if the considered formula is satisfiable, it is required to find at least one satisfying assignment). Thus, SAT is most likely to be unsolvable in polynomial time in the general case. On the other hand, combinatorial problems from a wide variety of subject areas can be effectively reduced to SAT [8].

It is well known that the Tseitin transformations [35] can be used to transition from SAT for an arbitrary Boolean formula to SAT for a formula in the Conjunctive Normal Form (CNF). With respect to this, hereinafter we consider SAT for arbitrary CNFs. The CNF is a conjunction of elementary Boolean constraints called *clauses*. Each clause is a disjunction of different literals among

which there are not complementary ones (remind that the literals x and $\neg x$ are called complementary). If x is an arbitrary Boolean variable, then the notation x^α , $\alpha \in \{0, 1\}$ means $\neg x$ if $\alpha = 0$ and x if $\alpha = 1$.

Let C be an arbitrary CNF over the set $X = \{0, 1\}^n$. Consider an arbitrary subset $B \subseteq X$. Denote all possible assignments of variables from B as $\{0, 1\}^{|B|}$. By $C[\beta/B]$ denote the CNF constructed from C by substituting an arbitrary assignment $\beta \in \{0, 1\}^{|B|}$ to C . Let A be an arbitrary polynomial algorithm (subsolver in terms of [37]).

Definition 1. ([37]). *For a CNF C over the set of Boolean variables X , the set $B \subseteq X$ is a Strong Backdoor Set w.r.t. the polynomial subsolver A if for each $\beta \in \{0, 1\}^{|B|}$, the algorithm A outputs the solution of SAT for $C[\beta/B]$.*

In [37] there is described an algorithm for solving SAT with complexity $O\left(p(|C|) \cdot \left(\frac{2^{|X|}}{\sqrt{|B|}}\right)^{|B|}\right)$ under the assumption that there exists an SBS B : $|B| \leq |X|/2$. However, this algorithm can be used in practice only if the considered formula has extremely small SBSs.

It is shown in [5] that the notion of SBS can be used to evaluate the hardness of Boolean formulas. In particular, the paper studies different approaches to estimating this hardness using tree-like metrics employed in propositional proof complexity. In addition, [5] demonstrates the relationship between these metrics and the so-called Backdoor hardness (the exact notion of Backdoor hardness is presented in [31]).

In [31], the following problem is considered: to construct such a set B , $B \subseteq X$ that the total time $\mu_{A,B}(C)$ of solving SAT for CNFs of the kind $C[\beta/B]$ over all possible $\beta \in \{0, 1\}^{|B|}$ by some complete SAT solving algorithm A (which is not necessarily polynomial) is less than the runtime of A on the original CNF C . Following the terminology from [34], let us refer to such a set B as to a Non-deterministic Oracle Backdoor Set (NOBS). In [31], it is shown that one can estimate the value of $\mu_{A,B}(C)$ using Monte Carlo sampling. The problem of finding the NOBS with the smallest value of $\mu_{A,B}(C)$ can thus be viewed as the problem of minimizing a special pseudo-Boolean black-box fitness function [31–33, 38].

Now consider the function

$$f : \{0, 1\}^n \rightarrow \{0, 1\}^m \quad (1)$$

defined by some cryptographic algorithm (cipher). This function is defined everywhere on $\{0, 1\}^n$ (in the case of a stream cipher, it corresponds to the set of all possible secret keys). Denote the set of possible images of f as $Range f$, $Range f \subseteq \{0, 1\}^m$. The main problem considered below is the problem of inversion (or of finding preimages) of the function f : given an arbitrary $\gamma \in Range f$ to find some $\alpha \in \{0, 1\}^n$ such that $f(\alpha) = \gamma$. It is a well-known fact that such problems can be effectively reduced to SAT [6].

To reduce the problem of the inversion of a specific function to SAT, one can use a number of software tools: for example, the well-known CBMC tool for verifying C programs [10] or the TRANSALG translator specialized for cryptanalysis instances [29]. The advantage of TRANSALG consists in that it constructs the so-called template CNF C_f for the function f [23]. Essentially, the template CNF is a symbolic representation of the algorithm A_f that specifies the function f . An important fact is that one can model how A_f works on an arbitrary input $\alpha \in \{0, 1\}^n$, $\alpha = (\alpha_1, \dots, \alpha_n)$ by the iterative application of the Unit Propagation rule [27] to the CNF

$$x_1^{\alpha_1} \wedge \dots \wedge x_n^{\alpha_n} \wedge C_f. \quad (2)$$

In (2), the variables x_1, \dots, x_n form a set, which we denote as X^{in} . These variables encode the input of the function f .

Our first goal is to use a SAT solver to construct an attack on the cryptographic function f : in particular, to learn how to effectively invert this function at least on some portion of its images. As mentioned above, we will construct SAT-based guess-and-determine attacks, i.e., search for such sets of guessed bits that make it possible to substantially simplify the solution of the inversion problem for f . It is quite clear that the notions of the guessed bits set and that of NOBS are very close in spirit. Unfortunately, as it follows from the results of [33, 34], NOBS do not suit well to constructing runtime estimations of guess-and-determine attacks due to the fact that in such estimations there is an unknown variance of some random variable.

To account for this, the notion of Inverse Backdoor Sets (IBS) is proposed in [34]. For IBS-based attacks, one can estimate their runtime with any predefined accuracy using the Monte Carlo scheme.

Briefly, the basic idea of IBS-based attacks proposed in [34] is as follows. Consider a template CNF C_f for the function f . Define a uniform distribution over $\{0, 1\}^n$ and construct a random sample of inputs $\alpha^1, \dots, \alpha^N$, $\alpha^j = (\alpha_1^j, \dots, \alpha_n^j)$, $j \in \{1, \dots, N\}$ for f . For each α^j , $j \in \{1, \dots, N\}$ consider CNF (2). It is well-known [7] that one can derive from (2) the values of all variables in the CNF C_f using only the Unit Propagation rule. We will say that such values are induced by the input α^j of the considered function. In particular, during this process the values $y_1 = \gamma_1^j, \dots, y_m = \gamma_m^j$, such that $f(\alpha^j) = \gamma^j$, $\gamma^j = (\gamma_1^j, \dots, \gamma_m^j)$ will be obtained.

Assume that A is a SAT solving algorithm, $B \subseteq X$ is an arbitrary set of variables in C_f , $\alpha \in \{0, 1\}^n$ is some input of f , and t is a positive constant that limits the runtime of A . Denote by $\beta(\alpha)$ the assignment of variables from B induced by the input α and by $\gamma(\alpha)$ the value of the function f on the input α . Associate with an arbitrary $\alpha \in \{0, 1\}^n$ the value of the function $\xi : \{0, 1\}^n \rightarrow \{0, 1\}$ defined as follows: $\xi(\alpha) = 1$ if A in time $\leq t$ finds a satisfying assignment of the CNF $C_f(\beta(\alpha), \gamma(\alpha))$, which results from substituting the assignments $\beta(\alpha)$ and $\gamma(\alpha)$ to the template CNF C_f ; otherwise $\xi(\alpha) = 0$.

Define a uniform distribution over $\{0, 1\}^n$. Then the portion of vectors from $\{0, 1\}^n$ on which the random variable ξ takes the value 1 is defined by the following probability:

$$\rho_{A,t}(B) = \frac{|\{\alpha \in \{0, 1\}^n \mid \xi(\alpha) = 1\}|}{2^n} \quad (3)$$

Since ξ is a Bernoulli random variable, then $\rho_{A,t}(B) = E[\xi]$, and $E[\xi]$ can be estimated via the Monte Carlo method [28]. In particular, one can use the sample mean of the values of ξ observed in N independent experiments in the role of the estimation of $E[\xi]$. It is important to note that this estimation does not depend on the characteristics of the algorithm A (as in the case of the estimations for the NOBS in [31]) and can be made arbitrarily precise by increasing the number of observations N . In more detail, let ξ^1, \dots, ξ^N be independent observations of the variable ξ . Applying the Chebyshev inequality [16] (w.r.t. that ξ is a Bernoulli variable) to ξ , we conclude that the following holds:

$$Pr \left\{ \left| \rho_{A,t}(B) - \frac{1}{N} \sum_{j=1}^N \xi^j \right| \leq \varepsilon \right\} \geq 1 - \frac{1}{4\varepsilon^2 N} \quad (4)$$

Definition 2 ([34]). *A non-empty set $B \subseteq X : |B| = s$ with the properties described above is called an Inverse Backdoor Set (IBS) with the parameters $(s, t, \rho_{A,t}(B))$ for C_f w.r.t. the algorithm A .*

In [34] there is described a general IBS-based guess-and-determine attack applicable to any function (1). This attack is applied to a set of outputs $\gamma^1, \dots, \gamma^M$ of function (1). If we consider the probability of inverting at least one $\gamma^k, k \in \{1, \dots, M\}$ to be $\geq 95\%$, then the runtime of this attack for some IBS B is:

$$Time_{A,t}(B) = 2^{|B|} \cdot t \cdot \frac{3}{\rho_{A,t}(B)} \quad (5)$$

As outlined in [34], it is possible to view the problem of constructing an effective guess-and-determine attack as a problem of finding a set B with the smallest value of (5). The latter problem can be considered as a problem of minimizing the following pseudo-Boolean fitness function:

$$\Phi_{A,t}(\theta_B) = 2^{|B|} \cdot t \cdot \frac{3N}{\sum_{j=1}^N \xi^j} \quad (6)$$

In (6), by θ_B we denote a Boolean vector of length $|X|$ where ones correspond to variables from X present in B , and zeroes correspond to variables from X absent from B . Probability (3) in (6) is replaced by its statistic estimation w.r.t. (4). To optimize functions (6), in [30, 33, 34] a computing cluster is used. In the role of optimization schemes the papers employ both local search algorithms and evolutionary algorithms.

3 Using Tree-Like Metrics to Estimate the Effectiveness of SAT-Based Guess-and-Determine Attacks

In this section, we introduce new metrics of effectiveness for IBS-based algebraic attacks. Our main motivation is inspired by the results of [5], where several different approaches to estimating the hardness of a Boolean formula via tree-like metrics are considered. For the measures studied in [5], it is possible to construct estimations of hardness for several infinite families of formulas (e.g., for pigeonhole principle formulas [12]). These estimations are constructed analytically, and it is completely unclear how one can obtain such estimations for arbitrary Boolean formulas. Below we introduce tree-like metrics of effectiveness for IBS-based guess-and-determine attacks. To solve SAT for CNFs of the kind $C_f(\beta, \gamma)$, we use modern complete SAT solvers based on the CDCL concept [27].

Let B , $|B| = s$ be an arbitrary IBS. The first observation consists in the fact that a Boolean hypercube $\{0, 1\}^s$ can be represented by a complete binary tree $T_s(B)$ of the depth s : the arbitrary assignment β of variables from B corresponds to the path $\pi(\beta)$ in $T_s(B)$, which goes from a root to a leaf. We can establish some order over $B = \{x_1^B, \dots, x_s^B\}$ so that all paths in $T_s(B)$ are traversed in this order, e.g., $x_1^B < \dots < x_s^B$ (we assume that the variable x_1^B is associated with a root of $T_s(B)$).

Let f be function (1), C_f be a template CNF for f over a set X of Boolean variables, and B , $B \subseteq X$ be an arbitrary IBS. Assume that A is a complete deterministic SAT solving algorithm that traverses some tree or forest in the process of its work. Note that both DPLL and CDCL are examples of such algorithms. In particular, DPLL traverses a binary tree in which all paths follow some common order. On the other hand, CDCL works with a forest formed by different binary trees, and each tree corresponds to a specific restart. In general, different paths in such a forest have different variable orderings. If C is a fixed CNF, then by $F_A(C)$ denote the tree (in the case of DPLL) or forest (in the case of CDCL) traversed by the considered algorithm A in order to construct an unsatisfiability proof for C or its satisfying assignment. By $F_A(C, t)$ denote the part of $F_A(C)$ that contains the first t paths traversed by A .

Consider an arbitrary $\gamma \in \text{Range } f$. Let $\pi(\beta)$ be an arbitrary path in $T_s(B)$, and $l(\beta)$ be a leaf of this path. Let us connect $l(\beta)$ with the root or with the set of roots $F_A(C_f(\beta, \gamma), t)$ by a new edge or several edges. We do this for each $\beta \in \{0, 1\}^{|B|}$ and for the corresponding path in $T_s(B)$. Denote the constructed tree as $T^*(C_f, A, B, \gamma, t)$. Let π^* be an arbitrary path in $T^*(C_f, A, B, \gamma, t)$. Taking into account the nature of the DPLL and CDCL algorithms, the path π^* corresponds to a sequence of decision levels [27] and literals derived via UP, which either ends in a conflict or in a derivation of an assignment that satisfies $C_f(\beta, \gamma)$.

Definition 3. *If the path π^* ends in a derivation of a satisfying assignment for $C_f(\beta, \gamma)$, then we refer to π^* as a positive path, otherwise, π^* is a negative path.*

Now let us establish the following fact.

Theorem 1. *Let α be an arbitrary input of f chosen from $\{0, 1\}^n$ in accordance with a uniform distribution, and $\gamma = f(\alpha)$. Assume that B is an IBS with the parameters $(s, t, \rho_{A,t}(B))$ for C_f w.r.t. a SAT solving algorithm A . Then the probability that the tree $T^*(C_f, A, B, \gamma, t)$ contains a positive path is $\rho_{A,t}(B)$.*

Sketch Proof. As follows from the definition of $\rho_{A,t}(B)$, this is the probability of an event that consists in finding by the algorithm A a satisfying assignment for $C_f(\beta(\alpha), \gamma(\alpha))$ in time $\leq t$. It is supposed that α is chosen from $\{0, 1\}^n$ w.r.t. the uniform distribution, and $\beta(\alpha)$, $\gamma(\alpha)$ are the assignments of variables from B and from the output of the function f , induced by α in the sense defined above. Denote the upper bound on the number of paths traversed by A in a tree or forest $F_A(C_f(\beta, \gamma))$ as t . Note that $T^*(C_f, A, B, \gamma, t)$ at the beginning is essentially a tree-like representation of the set $\{0, 1\}^{|B|}$. However, in this case, the tree contains a path that starts from $\beta(\alpha)$, where α is the preimage of γ w.r.t. f . Denote this path as $\tilde{\pi}$. From (3) it follows that $\rho_{A,t}(B)$ is the probability that $\tilde{\pi}$ is positive. Thus, the theorem is proved.

If we construct trees of the kind $T^*(C_f, A, B, \gamma^k, t)$ for $k \in \{1, \dots, M\}$, then from the proven theorem and the results of [34] it follows that for $M \geq 3/\rho_{A,t}(B)$, the probability that at least one of these trees has a positive path (which corresponds to the successful solution of the inversion problem for a specific γ^k) is at least 95%.

Definition 4. *We define the hardness of a guess-and-determine attack based on an IBS B in the context of the tree-like metric proposed above as the total number of paths in all trees $T^*(C_f, A, B, \gamma^k, t)$, $k \in \{1, \dots, M\}$.*

Note that the approach from [34] that uses the statistic estimation of the hardness of IBS-based guess-and-determine attacks can be naturally transferred to the introduced tree-like metric. Thus, we can use the computational scheme of the Monte Carlo method in combination with metaheuristic optimization for this purpose.

4 Class of Considered Functions

In this section, we describe the functions for the inversion of which we run computational experiments on a supercomputer. These are the variants of the compression function of the well-known cryptographic MD4 hash function. The MD4 function is vulnerable to the so-called collision attack [36]. However, for its inversion problem, no attacks with realistic runtime are known so far. The best preimage attack known to the authors is the one described in [24]. The attack proposed in the paper has a complexity of 2^{96} calls of the MD4 compression function. It is noteworthy that the computational results of [24] are quite hard to reproduce.

Our goal is to construct an estimate of the hardness of the inversion problem of the MD4 compression function w.r.t. the tree-like metrics proposed above and compare it with the results of [24]. We would like to highlight that it is not hard

to reproduce our results since for this purpose one can use publicly available software tools and a computer cluster.

In addition to the complete-round MD4 compression function, we will consider the inversion problems of the variants of this function, limited in the number of steps, for which we use the notation f_{MD4-k} , where k is the number of compression steps, $k \in \{1, \dots, 48\}$, and the case $k = 48$ corresponds to the complete-round compression function.

In application to functions of the kind f_{MD4-k} , we use special techniques for weakening their inversion problems. The basic idea of such techniques goes back to H. Dobbertin [14] and consists in fixing the values of some chaining variables to a constant. This step leads to the derivation of the values of some other variables. The SAT variant of Dobbertin's attack is described in [13], and this attack turns out to be substantially more effective compared to the original one. In [18], it is suggested to use the automatic search of relaxation constraints a la Dobbertin by applying black-box optimization algorithms. As a result, new relaxation constraints are found, they give an attack that is significantly more effective than the attack from [13]. In particular, the attack from [18] allows inverting the MD4-39 function in less than 1 minute on randomly selected vectors from $\{0, 1\}^{128}$ on a personal computer.

The key idea of the attacks from [18–20] is to reduce the inversion problem of functions of the kind

$$f_{MD4-k} : \{0, 1\}^{512} \rightarrow \{0, 1\}^{128}$$

to inverting functions of the kind

$$g_{MD4-k}^{\lambda_r} : \{0, 1\}^{p_r} \rightarrow \{0, 1\}^{128}, \quad (7)$$

where $p_r < 512$. Functions (7) are built using the vectors λ_r , which specify the sets of Dobbertin's relaxation constraints. Any λ_r is a Boolean vector of length 48: ones in it indicate at which step of the compression algorithm the corresponding chaining variable is replaced by the constant 0³². So, for example, the vector λ_1 used in [18] (in the paper it appears as ρ_1) defines the function

$$g_{MD4-k}^{\lambda_1} : \{0, 1\}^{128} \rightarrow \{0, 1\}^{128}. \quad (8)$$

As shown in [19], function (8) is defined almost everywhere on $\{0, 1\}^{128}$. At least half of the vectors from $\{0, 1\}^{128}$ have $g_{MD4-k}^{\lambda_1}$ -preimages (at least for $k \in \{39, \dots, 45\}$). If for $\chi \in \{0, 1\}^{128}$ there exists such $\alpha' \in \{0, 1\}^{128}$ that $g_{MD4-k}^{\lambda_1}(\alpha') = \chi$, then from such α' we can effectively recover $\alpha \in \{0, 1\}^{512}$ for which the following holds: $f_{MD4-k}(\alpha) = \chi$.

5 Computational Experiments

In this section, we provide a description of computational experiments in which SAT-based guess-and-determine attacks for functions of kind (8) are built.

In these experiments, at the stage of debugging and testing software applications, the “Akademik V.M. Matrosov” cluster of Irkutsk Supercomputer Center [2] is used. The main computational experiments are carried out on the cluster of Peter the Great St. Petersburg Polytechnic University [3].

5.1 General Scenario

Once again let us give a brief description of the computational problem, for the solution of which a supercomputer is used. We consider the inversion problems of a function of kind (8) for different k , $k \in \{40, \dots, 48\}$ (for $k < 40$, the corresponding problems can be solved on a PC). For such functions, we build IBS-based guess-and-determine attacks. In the role of a SAT solving algorithm A , we use several state-of-the-art CDCL-based SAT solvers: **MiniSat**, **Glucose**, **Cadical**, **MapleLCM**.

Let us construct a template CNF $C_{g_{MD4-k}^{\lambda_1}}$ using the TRANSALG tool [29]. For each considered function of kind (8) we generate a sample on N random inputs $\alpha^1, \dots, \alpha^N \in \{0, 1\}^{128}$, and for each $j \in \{1, \dots, N\}$ we generate the assignment of all variables in $C_{g_{MD4-k}^{\lambda_1}}$ by applying UP to CNFs of the kind $x_1^{\alpha_1^j} \wedge \dots \wedge x_n^{\alpha_n^j} \wedge C_{g_{MD4-k}^{\lambda_1}}$. An arbitrary set $B \subseteq X$ is represented by a Boolean vector θ_B of length $|X|$. Let us define at an arbitrary point $\theta_B \in \{0, 1\}^{|X|}$ the value of fitness function (6) as follows:

1. For a random sample $\alpha^1, \dots, \alpha^N$ generate outputs $\gamma^1, \dots, \gamma^N$ of the function $g_{MD4-k}^{\lambda_1}$ and assignments of variables from B : β^1, \dots, β^N ;
2. Consider CNF formulas $C_{g_{MD4-k}^{\lambda_1}}(\beta^j, \gamma^j)$, $j \in \{1, \dots, N\}$ and apply the SAT solver A to each such a formula with a constraint on the number of conflicts $\leq t$ (the constant t is chosen during the experiments). If A has made more than t conflicts, the corresponding computation is interrupted;
3. With each run of A on the CNF $C_{g_{MD4-k}^{\lambda_1}}(\beta^j, \gamma^j)$ we associate the observed value ξ^j of the random variable ξ : if A managed to find a satisfying assignment for $C_{g_{MD4-k}^{\lambda_1}}(\beta^j, \gamma^j)$ using $\leq t$ conflicts, then $\xi^j = 1$, otherwise $\xi^j = 0$;
4. Calculate the value of the fitness function at the point θ_B according to formula (6);
5. Go to a new point θ_B using some metaheuristic strategy [26].

Note that a similar approach is used in several prior papers, e.g., [30, 34]. However, in these papers, the runtime of the algorithm A is limited by a straightforward time limit in seconds, while in the present paper we impose a limit in the number of conflicts. In addition, the cited papers employ different algorithms to optimize the fitness function, in particular, [34] uses the tabu search algorithm, while [30] utilizes several variants of the (1+1) Evolutionary algorithm, as well as a special genetic algorithm (GA). It is the latter algorithm that we use in our experiments, however, for the purposes of the present paper it is reimplemented to fully work on clusters.

In this work, we use a version of the algorithm from [30] implemented as an MPI program, and thus one can run it on a computer cluster of any capacity. Specifically, the algorithm works with several points of the hypercube forming a population $\theta_{B_1}, \dots, \theta_{B_Q}$. Let Π be an arbitrary population of the size Q . We associate the distribution $D_\Pi = \{p_1, \dots, p_Q\}$ with it, defining the probabilities $p_l, l \in \{1, \dots, Q\}$ as follows:

$$p_l = \frac{1/\Phi(\theta_{B_l})}{\sum_{i=1}^Q (1/\Phi(\theta_{B_i}))} \quad (9)$$

The transition from the current population Π towards the new population Π' is performed as follows. We select individuals from Π randomly w.r.t. the distribution D_Π and to each pair of selected individuals we apply the standard two-point crossover operator [26] in a combination of the FGA-mutation operator proposed in [15]. In such a way we construct G individuals of a new population. We also add to the new population H individuals from Π that have the best values of the fitness function (this step corresponds to the so-called elitism concept [26]). As a result, we ensure that the following holds: $G + H = Q$. In all computational experiments we use the following values of these parameters: $Q = 10, G = 8, H = 2$.

From the definition of the fitness function, it directly follows that this function is quite costly in a computational sense. Even with the use of a powerful supercomputer, optimizing such a function over the hypercube $\{0, 1\}^{|X|}$ would require colossal computational resources. That is why in our experiments we solve the optimization problems of functions (6) on some special subsets of X . As noted in [34], we can take as a started point of our optimization process some Strong Unit Propagation Backdoor Set (SUPBS). If B is a SUPBS, then the calculation of the value $\Phi(\theta_B)$ takes $t = 0$ conflicts. In our cases, a trivial SUPBS is formed by the input variables of the considered function, and we denote this set by X^{in} . For any function considered in this paper this set consists of 128 variables. In our experiments we optimize the functions of the kind $\Phi(\theta_B)$, namely, on $2^{X^{in}}$.

5.2 Implementation and Results

As said above, we conduct computational experiments on two supercomputers, the ‘‘Akademik V.M. Matrosov’’ cluster of Irkutsk Supercomputer center [2] and the computing cluster of St. Petersburg Polytechnic University (SPPU) [3]. Each compute node of the former is equipped with two 18-core processors Intel Xeon E5 2695 v4 and 128 GB DDR4-2400 RAM. In our experiments, we employ up to 10 nodes (180 cores) for up to one day. The nodes of the latter cluster are equipped with four 14-core processors Intel Xeon E5 2695 v3 and 64 GB DDR4-2400 RAM. In each experiment, we harness 25 nodes (1400 cores) and use the same duration (one day).

The ‘‘Akademik V.M. Matrosov’’ cluster is mainly used to debug and test the developed MPI program and the employed SAT solvers (see details further). The major part of the computational experiments is run on the SPPU cluster.

In our experiments, we consider the problems of inverting the functions $g_{MD4-k}^{\lambda_1}$, $k \in \{43, 45, 47, 48\}$ described above. To minimize the functions $\Phi(\theta_B)$, we employ the EvoGuess framework [1], specially developed for solving pseudo-Boolean optimization problems associated with SAT. This framework implements several evolutionary algorithms, however, we mainly use the implementation of the Genetic Algorithm described above. EvoGuess can be used as an MPI application and, thus, can harness any number of available cores. Actually, EvoGuess can be considered as an optimization wrap-around for the PySAT tool [21], which, in turn, is the environment for invoking SAT and MaxSAT solvers incrementally from Python. PySAT supports a number of modern SAT solvers (MiniSat, Glucose, Cadical, MapleLCMDist, etc.). On the testing stage, we check all of them, but for our class of instances the best results are obtained using MiniSat 2.2, thus in all experiments we use this solver.

The results of the computational experiments are presented in the following table.

Table 1. Results of the experiments. In each experiment, 1400 cores of the SPPU cluster are used (25 nodes) for 1 day (24 h). The SAT solver MiniSat 2.2 (embedded in the PySAT tool) is applied.

	$MD4-43$, (inverting $g_{MD4-43}^{\lambda_1}$)	$MD4-45$, (inverting $g_{MD4-45}^{\lambda_1}$)	$MD4-47$, (inverting $g_{MD4-47}^{\lambda_1}$)	$MD4-48$, (inverting $g_{MD4-48}^{\lambda_1}$)
Best value of $\Phi(\theta_B)$ in the proposed tree-like metric (estimation of guess-and-determine attack hardness)	2.489460e+13	1.770887e+21	2.415492e+28	8.222028e+36
Size of the best backdoor (w.r.t. the best value of $\Phi(\theta_B)$)	19	46	70	99
Number of visited points in the hypercube	54312	47840	42456	41984

Let us briefly discuss the obtained results. Recall that with respect to what is said above, we can consider one conflict as one elementary call of a special function (actually, a SAT solver), which is used to solve the corresponding cryptanalysis problem. On the other hand, in brute-force attacks on each of the considered functions we make $2^{128} \approx 3,4e+38$ calls of the corresponding function in the worst case scenario and 2^{127} calls on average. Thus, we can conclude that the MD4 compression function has non-trivial SAT based guess-and-determine attacks for $k = 43, 45, 47$. For a complete-round version of this function, i.e., for MD4-48,

the obtained attack is comparable to a brute-force attack, and thus the use of SAT solvers does not lead to any advantage in this case.

6 Conclusion

In this paper, we present new measures to estimate the hardness of algebraic attacks on cryptographic functions that use state-of-the-art SAT solvers. The proposed measures are tree-like. In fact, they are statistic estimations of the number of paths in some tree, which corresponds to the process of enumerating all possible assignments of variables in some guessed bits set. By estimating this number, we construct the estimation of the corresponding attack. The problem of constructing the best attack of this kind is viewed as a problem of minimizing some fitness function, the values of which are calculated probabilistically using the Monte Carlo method. To optimize such a function, we use a specially developed framework for solving black-box optimization problems associated with SAT, which is an MPI program that can work on a supercomputer. In our computational experiments, two clusters [2,3] are harnessed. As a result, we construct a non-trivial SAT based guess-and-determine attack on reduced-round versions of the compression function of the well-known MD4 hashing algorithm, namely, for MD4-43, MD4-45, MD4-47. Using the proposed tree-like metrics, it is shown that the constructed attacks are significantly more effective than brute-force attacks.

Acknowledgments. The research was supported by the Russian Science Foundation, project No. 22-21-00583.

References

1. Evoguess: Framework for hardness estimating of SAT instances by decomposition set searching. <https://github.com/ctlab/evoguess>. Accessed 11 Mar 2022
2. Irkutsk Supercomputer Center of the SB RAS. <https://hpc.icc.ru/>. Accessed 11 Mar 2022
3. Supercomputer center “Polytechnic”. <https://research.spbstu.ru/skc/>. Accessed 11 Mar 2022
4. Anderson, R.: A5 (was: Hacking digital phones). Newsgroup Communication (1994)
5. Ansótegui, C., Bonet, M.L., Levy, J., Manyà, F.: Measuring the hardness of SAT instances. In: Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 1, AAAI2008, pp. 222–228. AAAI Press (2008)
6. Bard, G.: Algebraic Cryptanalysis. Springer New York (2009). <https://doi.org/10.1007/978-0-387-88757-9>
7. Bessiere, C., Katsirelos, G., Narodytska, N., Walsh, T.: Circuit complexity and decompositions of global constraints. In: IJCAI, pp. 412–418. Morgan Kaufmann Publishers Inc., San Francisco (2009)
8. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press, Amsterdam (2009)

9. Chin-Liang, C., Chang, C., Zhang, J., Lee, R., Coaut, C.: Symbolic logic and mechanical theorem proving. In: *Computer Science and Applied Mathematics : A Series of Monographs and Textbooks*. Elsevier Science (1973)
10. Clarke, E., Kroening, D., Lerda, F.: A tool for checking ANSI-C programs. In: Jensen, K., Podelski, A. (eds.) *TACAS 2004*. LNCS, vol. 2988, pp. 168–176. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24730-2_15
11. Cook, S.A.: The complexity of theorem-proving procedures. In: *Proceedings of the Third Annual ACM Symposium on Theory of Computing, STOC 1971*, pp. 151–158. Association for Computing Machinery, New York, NY, USA (1971)
12. Cook, S.A., Reckhow, R.A.: The relative efficiency of propositional proof systems. *J. Symb. Logic* **44**(1), 36–50 (1979)
13. De, D., Kumarasubramanian, A., Venkatesan, R.: Inversion attacks on secure hash functions using SAT solvers. In: Marques-Silva, J., Sakallah, K.A. (eds.) *SAT 2007*. LNCS, vol. 4501, pp. 377–382. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72788-0_36
14. Dobbertin, H.: The first two rounds of MD4 are not one-way. In: Vaudenay, S. (ed.) *FSE 1998*. LNCS, vol. 1372, pp. 284–292. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-69710-1_19
15. Doerr, B., Le, H.P., Makhmara, R., Nguyen, T.D.: Fast genetic algorithms. In: *Proceedings of GECCO 2017*, pp. 777–784 (2017)
16. Feller, W.: *An Introduction to Probability Theory and Its Applications*, vol. 1, 3rd edn. Wiley, New York (1968)
17. Gomes, C.P., Sabharwal, A.: Exploiting runtime variation in complete solvers. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) *Handbook of Satisfiability*, Amsterdam (2009)
18. Gribanova, I., Semenov, A.: Using automatic generation of relaxation constraints to improve the preimage attack on 39-step MD4. In: *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pp. 1174–1179 (2018)
19. Gribanova, I., Semenov, A.: Parallel guess-and-determine preimage attack with realistic complexity estimation for MD4-40 cryptographic hash function. In: *Proceedings of XIII Conference Parallel Computational Technologies (PaCT)*, pp. 8–18 (2019)
20. Gribanova, I., Semenov, A.: Constructing a set of weak values for full-round MD4 hash function. In: *2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO)*, pp. 1212–1217 (2020)
21. Ignatiev, A., Morgado, A., Marques-Silva, J.: PySAT: A Python toolkit for prototyping with SAT oracles. In: *SAT*, pp. 428–437 (2018)
22. Kipnis, A., Shamir, A.: Cryptanalysis of the HFE Public Key Cryptosystem by Relinearization. In: Wiener, M. (ed.) *CRYPTO 1999*. LNCS, vol. 1666, pp. 19–30. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_2
23. Kochemazov, S., Zaikin, O., Gribanova, I., Otpuschennikov, I., Semenov, A.: Translation of algorithmic descriptions of discrete functions to SAT with applications to cryptanalysis problems. *Log. Methods Comput. Sci.* **16**, 1–42 (2020)
24. Leurent, G.: MD4 is not one-way. In: Nyberg, K. (ed.) *FSE 2008*. LNCS, vol. 5086, pp. 412–428. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-71039-4_26
25. Levin, L.A.: Universal Sequential Search Problems. *Probl. Inf. Transm.* **9**(3) (1973)
26. Luke, S.: *Essentials of Metaheuristics*, 2nd edn. Lulu, Raleigh (2013)

27. Marques-Silva, J., Lynce, I., Malik, S.: Conflict-driven clause learning SAT solvers. In: Biere, A., Heule, M., Van Maaren, H., Walsh, T. (eds.) *Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications*, pp. 133–182. IOS Press BV, Amsterdam (2009)
28. Metropolis, N., Ulam, S.: The Monte Carlo method. *J. Am. Stat. Assoc.* **44**(247), 335–341 (1949)
29. Otpuschennikov, I., Semenov, A., Gribanova, I., Zaikin, O., Kochemazov, S.: Encoding cryptographic functions to SAT using TRANSALG system. In: *Proceedings of the Twenty-Second European Conference on Artificial Intelligence, ECAI 2016*, pp. 1594–1595. IOS Press, NLD, Amsterdam (2016)
30. Pavlenko, A., Semenov, A., Ulyantsev, V.: Evolutionary computation techniques for constructing SAT-based attacks in algebraic cryptanalysis. In: Kaufmann, P., Castillo, P.A. (eds.) *EvoApplications 2019. LNCS*, vol. 11454, pp. 237–253. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-16692-2_16
31. Semenov, A., Chivilikhin, D., Pavlenko, A., Otpuschennikov, I., Ulyantsev, V., Ignatiev, A.: Evaluating the hardness of SAT instances using evolutionary optimization algorithms. In: Michel, L.D. (ed.) *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*, *Leibniz International Proceedings in Informatics (LIPIcs)*, vol. 210, pp. 47:1–47:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2021)
32. Semenov, A., Zaikin, O.: Algorithm for finding partitionings of hard variants of boolean satisfiability problem with application to inversion of some cryptographic functions. *SpringerPlus* **5**(1), 1–16 (2016). <https://doi.org/10.1186/s40064-016-2187-4>
33. Semenov, A., Zaikin, O., Kochemazov, S.: Finding effective SAT Partitionings Via black-box optimization. In: Pardalos, P.M., Rasskazova, V., Vrahatis, M.N. (eds.) *Black Box Optimization, Machine Learning, and No-Free Lunch Theorems. SOIA*, vol. 170, pp. 319–355. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-66515-9_11
34. Semenov, A.A., Zaikin, O., Otpuschennikov, I.V., Kochemazov, S., Ignatiev, A.: On cryptographic attacks using backdoors for SAT. In: *Thirty-Second AAAI Conference on Artificial Intelligence*, pp. 6641–6648 (2018)
35. Tseitin, G.S.: On the complexity of derivation in propositional calculus. In: *Studies in Constructive Mathematics and Mathematical Logic, Part II, Seminars in Mathematics*, pp. 115–125 (1970)
36. Wang, X., Lai, X., Feng, D., Chen, H., Yu, X.: Cryptanalysis of the hash functions MD4 and RIPEMD. In: Cramer, R. (ed.) *EUROCRYPT 2005. LNCS*, vol. 3494, pp. 1–18. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_1
37. Williams, R., Gomes, C., Selman, B.: Backdoors to typical case complexity. In: *International Joint Conference on Artificial Intelligence*, pp. 1173–1178 (2003)
38. Zaikin, O.S., Kochemazov, S.E.: On black-box optimization in divide-and-conquer SAT solving. *Optim. Methods Softw.* **36**, 1–25 (2019)