From a Logical Approach to Internal States of Hash Functions
How SAT *Problem Can Help to Understand* SHA-⋆ *and* MD⋆

Florian Legendre[1], Gilles Dequen[2] and Michaël Krajecki[1]
[1]*UFR Sciences, University of Reims Champagne-Ardennes, Moulin de la Housse, Reims, France*
[2]*MIS, University of Picardie Jules Verne, Amiens, France*

Keywords: Hash Functions, Logical Cryptanalysis, MD5, SHA-1, Satisfiability.

Abstract: This paper deals with logical cryptanalysis of hash functions. They are commonly used to check data integrity and to authenticate protocols. These functions compute, from an any-length message, a fixed-length bit string, usually named *digest*. This work defines an experimental framework, that allows, thanks to the propositional formalism, to study cryptosystems at the bit level through corresponding instances of the SAT problem. Thus, we show that some internal words of popular hashing functions MD⋆ and SHA-⋆ are not as random as expected and provide some convincing elements to explain this phenomenon by the use of round constants. Because this presents several weaknesses, we show how to detect and exploit these ones through an application based on logical cryptanalysis. As a result we show equivalences, and quasi-equivalences between digits and explain how we inverse reduced-step versions of MD5 and SHA-1.

1 INTRODUCTION

In the last years, the proliferation of digital systems has placed cryptology at the heart of our communications. Within this context, studies about cryptographic functions are a keystone to preserve the sustainability of our systems. More specifically, the field of Cryptanalysis consists in finding weaknesses that will facilitate the retrieval of any secret information. Several general cryptanalysis approaches have been proposed over the years such as differential (Biham and Shamir, 1990) or linear (Matsui and Yamagishi, 1992) ones. This paper deals with cryptographic hash functions that are central elements of modern cryptography. A hash function can be defined as a deterministic process that generates a fixed-length bit string, usually named *digest*, from any-length bit string also named the *message*. It is commonly used to check integrity of files or communications. Moreover, it usually participate to authentication protocols. One of their main characteristic is to diffuse and confuse an input message in a very fast way. Within this framework, the internal 32-bit words of a hashing process need to look as random as possible. So that making this possible, the compression function often use round constants, derived from physical constants (Knuth, 1997). In (Legendre et al., 2012), we encoded the MD5 hash function in a DIMACS format in order to tackle the preimage thanks to SAT solving. In this paper we focus on our SAT modeling in order to examine the structural behavior of the internal words of the process by two different ways. The first one is via an automatic logical reasoning that allow to deduce equivalencies and some special relations between variables. The second one is by using the formula to generate statistical informations so that estimating a generic behavior of the process. From this, we then deduce classical and conditional probabilities that shed a light on unexpected structural informations. Indeed we show that just using round constants leads to belie the idea of a total randomness of the hashing process and can give some information that could help an attacker. From this, we present equivalences, quasi-equivalences and quasi-implications that could be used in other cryptanalytic approach. This paper is organized as follows: In section 2, we give an overview of hash functions more focused on those that use the *Merkle-Damgård* construction. We also recall some notations and objects related to the SAT problem and its solving rules. The section 3 deals with our probabilistic approach and give details about specific cases where probabilities aren't uniforms. In section 4, we show how to use these special weaknesses in a practical framework and particularly by using SAT solvers and logical reasoning. Finally we conclude and open perspectives.

Legendre F., Dequen G. and Krajecki M..
From a Logical Approach to Internal States of Hash Functions - How SAT Problem Can Help to Understand SHA-* and MD*.
DOI: 10.5220/0004534104350443
In *Proceedings of the 10th International Conference on Security and Cryptography* (SECRYPT-2013), pages 435-443
ISBN: 978-989-8565-73-0
Copyright © 2013 SCITEPRESS (Science and Technology Publications, Lda.)

2 BACKGROUND AND PRELIMINARIES

2.1 About Cryptographic Hash Functions

A cryptographic hash function can be defined as a deterministic algorithm that maps an any-length bit string (also named the *message*) to a fixed-length bit string, usually named *digest* or *hash*. Among the uses of such a function we can notice for instance the integrity check of files or communications or digital signature. It can also contribute to ensure authentication protocols with Message Authentication Codes (MACs) which are a mean that two users with a shared secret key can authenticate between each other. To make these functions secure, they required to be theoretically or computationally collision and (second) preimage resistant.

2.2 Notations

In this paper, we mainly focus on the popular MD$\star$ and SHA-$\star$ hash functions that are built following the *Merkle-Damgård* construction (Merkle, 1989; Damgård, 1989). Each of these functions uses internal 32-bit words that are described with the following notations. Let be the process at step i. We denote each word as:

- Q_i is the internal state obtained at the end of a step.
- f_i is a non-linear function. It can be named {F, G, H, I}, depending on the step considered.
- S_i is a sum resulting of a 4 operands addition. This represents the main operation of a round.
- Within the propositional context, an addition of 4 operands could generate 2 levels of carry. fC_i is the first level. sC_i is the second level of carry.
- tC_i is the first (and unique) level carry resulting of a 2 operands addition
- Cst_i is a round constant

Note also:

- $W[j]$ is the jth bit of a 32-bit word, $j \in \{0, \dots 31\}$
- M_k is the kth 32-bit word from the input message, $k \in \{0, \dots 15\}$

2.3 About MD5

MD5 was designed in 1991 by Ron Rivest as an evolution of MD4, strengthening its security by adding some improvements. The operating principle of this function consists in the repetition of 64 steps, defined with three sub-steps as follows:

a) $Q_i \leftarrow Q_{i-4} + f(Q_{i-1}, Q_{i-2}, Q_{i-3}) + M_k + Cst_i$
b) $Q_i \leftarrow Q_i \lll s_n$
c) $Q_i \leftarrow Q_i + Q_{i-1}$

where :

- i is the current step, $\in \{1, \dots, 64\}$
- $Q_{-3}, Q_{-2}, Q_{-1}, Q_0$ are the Initial Values (I.V.).
- $\lll s_n$ the circular shifting to the left(rotating) by n bits position, depends on i.
- $f_i \in$ { F, G, H, I }, where:
  F(X,Y,Z) = $(X \wedge Y) \vee (\overline{X} \wedge Z)$
  G(X,Y,Z) = F(Z,X,Y)
  H(X,Y,Z) = $X \oplus Y \oplus Z$
  I(X,Y,Z) = $Y \oplus (X \vee \overline{Z})$

2.4 About SHA-1

SHA-1 was designed in 1995 by the NSA as an improved version of SHA-0 in order to prevent some weaknesses. The operating principle is the same as the MD$\star$ family and consists in a hashing process where five states of 32-bit words are initialized and then modified at each of the 80 steps. A step can be defined with the following sub-steps:

a) $Q_i \leftarrow (Q_{i-1} \lll 5)$
b) $Q_i \leftarrow Q_i + f(Q_{i-2}, (Q_{i-3} \lll 30), Q_{i-4})$
c) $Q_i \leftarrow Q_{i-5} + W[i] + Cst_i$

where :

- i is the current step, $\in \{0, 1, \dots, 79\}$.
- $Q_{i-1}, Q_{i-2}, Q_{i-3}, Q_{i-4}, Q_{i-5}$ are the I.V
- Cst_i is defined among four predefined constants.
- $\lll r$, the circular shifting to the left(rotating) by r bits position.
- $f_i \in$ { F, G, H, I }, where:
  F(X,Y,Z) = $(X \wedge Y) \vee (\overline{X} \wedge Z)$
  G(X,Y,Z) = $X \oplus Y \oplus Z$
  H(X,Y,Z) = $(X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z)$
  I(X,Y,Z) = G(X,Y,Z)
- $W[i]$ is the i^{th} word of 32 bits. These words are built from the input message, as follows:
  - if $i < 16$
    $W[i]$ is the i^{th} 32-bit word from the message
  - if $16 \leq i \leq 79$
    $W[i] \leftarrow (W[i-3] \oplus W[i-8] \oplus W[i-14] \oplus W[i-16]) \lll 1$

Note SHA-1 differs from SHA-0 by the shifting of $W[i]$. Finally, note in the following of this paper, 1 round for a MD$\star$ (resp. SHA-$\star$) function corresponds to 16 (resp. 20) steps.

2.5 Notation about SAT Solving

Since our approach is based on the logical cryptanalysis principles, this work is closely related to SAT solving techniques. The *boolean* SAT*isfiablilty problem* (short for SAT) is a well-known NP-Complete problem (Biere et al., 2009; Cook, 1971). Its interest has grown significantly these past few years because of its simplicity and its ability to express a wide set of various problems. Moreover, the last progresses about solving techniques have led SAT to be a great and competitive approach to tackle a wide range of industrial and academic problems. Among them, logical cryptanalysis is a very recent use of SAT formalism and already have produced some new results.SAT is of determining if a boolean expression $\mathcal{F}$ has at least one assignment of truth value (also named an *interpretation*) $\{$TRUE, FALSE$\}$ to its variable so that it is TRUE. In this paper, $\mathcal{F}$ is considered as a CNF-formula (Conjunctive Normal Form) which can be defined as a set of clauses (interpreted as a conjunction) where a clause is a set (interpreted as a disjunction) of literals.

More precisely, let $\mathcal{V} = \{ v_1, \ldots, v_n \}$ be a set of n boolean variables. A signed boolean variable is named a *literal*. We denote, v_i and $\overline{v_i}$ the positive and negative literals referring to the variable v_i respectively. The literal v_i (resp. $\overline{v_i}$) is TRUE (also said *satisfied*) if the corresponding variable v_i is assigned to TRUE (resp. FALSE). Literals are commonly associated with logical AND and OR operators respectively denoted $\wedge$ and $\vee$. As mentioned above, a *clause* is a disjunction of literals, that is for instance $v_1 \vee \overline{v_2} \vee v_3 \vee v_4$. Hence, a clause is satisfied if at least one of its literals is satisfied. As a SAT formula $\mathcal{F}$ is considered under CNF, it is satisfied if all its clauses are satisfied. Finally, if its exists an assignment of $\mathcal{V}$ on $\{$TRUE, FALSE$\}$ such as to make the formula $\mathcal{F}$ TRUE, $\mathcal{F}$ is said SAT and UNSAT otherwise.

2.6 Our Approach

The cryptanalysis of a hash function can be done thanks to an algebraic approach. Generally, two ways are helpful to improve efficiently the solving method: Gröbner basis (Faugère and Joux, 2003; Bettale et al., 2012) and SAT solvers (Bard et al., 2007; Mironov and Zhang, 2006). Among these, the best way to study the function with a bitwise reasoning, *i.e.*in $\mathbb{F}_2$, seems to be the one using a SAT formalism because SAT tools propose an easy way to examine the problem modeled in its finest granularity. In our knowledge, no work exists about how analyzing and measuring the security of hash function thanks to this approach. In (De et al., 2007) and then in our previous work (Legendre et al., 2012), the method used consists in encode the MD5 hash function under its corresponding CNF expression so that practically tackling the (second) preimage. In these works, some hash functions are represented with a bitwise reasoning only by using boolean equations. Thus, the whole process is described with the tiniest modelling, then simplified thanks to logical simplifications and finally inverted reduced-step versions thanks to a generic SAT solver. In this paper, we choose to reuse our SAT representation of MD$\star$ and SHA-$\star$ with aim to identify some practical weaknesses that will allow future works on collision and (second) preimage inversion. In practice, we consider the CNF as a tool to examine the structural behavior of the internal words of the process by two different ways. First, we searched to outline the fact there are some special relations between variables as implications or equivalences. This can be done thanks to an automatic logical reasoning. Second, we use the CNF to estimate, thanks to statistics and within a generic framework, how behave each variable (correlated to each digit of the Hashing Process) in relation to all others. From this, we deduced classical and conditional probabilities. As a result, we presented equivalences, quasi-equivalences and quasi-implications that could be used in any cryptanalytic approach. In this sense, we finally talked about logical cryptanalysis and presented a practical inversion of a 23 steps SHA-1's process.

3 LOGICAL REASONING

In (Legendre et al., 2012) we showed that using logical simplifications applied on a SAT formula describing a process of hashing helps to tackle the second preimage of MD5 up to 28 steps which is still, in our knowledge, the best practical inversion. Since this appears to be a promising way to break more steps, we enriched the original simplification process. In the following, we describe logical simplifications we process to learn information that could be helpful within a reduced-step inversion of MD5 or SHA-1.

3.1 Detection of Equivalences

The existence of a logical equivalency, from a point of view of a valid process of hashing, means that at least two digits (it could be more) are linked by their respective value in every model. Practically and informally, this can be seen as a digit that has always the same (or opposite) value as another one. If such a case occurs, both digits represent the same information and only one of them should be considered into the process. Such a relation is denoted with the operator : $\Leftrightarrow$. As an example, consider the CNF formula $\mathcal{F}$ having the following clauses, the *detection of equivalencies* can be computed as :

$$\begin{array}{ll} c_1 = (a \vee \overline{b}) & c_2 = (\overline{a} \vee \overline{f}) \\ c_3 = (b \vee \overline{c}) & c_4 = (c \vee \overline{d}) \\ c_5 = (f \vee \overline{g}) & c_6 = (g \vee \overline{h}) \\ c_7 = (a \vee d \vee \overline{e}) & c_8 = (\overline{a} \vee h \vee e) \end{array}$$

- If a is set to FALSE then you can directly deduce that b, c, d and e must be set to FALSE, unless to falsify $\mathcal{F}$, thanks to the clauses c_1, c_3, c_4 and c_7 respectively. Hence, we notice that a equals to FALSE implies e equals to FALSE. We denote this implication $\overline{a} \Rightarrow \overline{e}$. The corresponding CNF expression is $\overline{e} \vee a$. As a remark, this clause also represents the implication $e \Rightarrow a$.
- In the same way, if a is set to TRUE then you can imply that f, g, h are set to FALSE and e to TRUE respectively.
- Consequently, since $e \Rightarrow a$ and $a \Rightarrow e$, this means that whatever could be the solution, a and e have the same value. This is denoted $a \Leftrightarrow e$. Therefore, you can substitute every e in the formula by a (and vice versa). From this, may result a cascade of new simplifications. For instance, proceeding that substitution in $\mathcal{F}$ leads c_7 and c_8 to become obsolete, $a \vee \overline{a}$ being tautological and so useless.

From applying this type of treatments on our SAT formulas results several equivalences. Some of them are trivial. In spite of this, others are not so. In the following, we mention some examples.

- *A Trivial Case:*
  $F_1[29] \Leftrightarrow Q_1[29]$. This equivalence is quite easy to detect because $F_1 = (Q_1 \wedge Q_0) \vee (\overline{Q_1} \wedge Q_{-1})$ and Q_0 and Q_{-1} are I.V. and hence are constant. This means that if $Q_0[i]$ differs from $Q_{-1}[i]$ $(i \in \{0 \ldots 31\})$, then F_1 depends exclusively on Q_1. There is a relation of equivalence which appears between F_1 and Q_1 which appears once on four on average.
- *Non Trivial Case:*
  This is the most interesting case. It seldom occurs within a general framework, but it gives a pertinent information to cryptanalysts. We name *special case* a non-trivial case which occurs in a specific formula. For instance, if we apply our treatment on a CNF describing a preimage attack of MD5, we exhibit equivalences that are not related to the entire MD5 process but to the specific instance. Thanks to that, if we consider a preimage attack on the 29 first steps of MD5 where the reduced-step digest is set to 0, we then deduce $M_8[2] \Leftrightarrow \overline{Q_{24}[2]}$, where M_8 is the 9th bloc of the input message M.
- *Direct Implication:*
  If two implications of the form $a \Rightarrow b$ and $\overline{a} \Rightarrow b$ occur then for all value of a, b equals to TRUE. Consequently, b must not be set to FALSE unless to falsify $\mathcal{F}$. As an illustration, we deduce that $sC_{39}[0]$ must be set to FALSE on our reduced-step preimage attack of MD5.

3.2 Static Look-ahead and More

We apply a classic local treatment in SAT solving named *Look-Ahead* (Li and Anbulagan, 1997) which consists in foreseeing the effects of choosing a branching variable to evaluate one of its values. From this evaluation, it can infer an assignment or some informations among equivalences (see section 3), fixed literals and new binary clauses. Hereafter, some details.

- *Fixed literals*:

  i) if $a \Rightarrow false$ then $\overline{a}$ must be set to TRUE
- *New binary clauses:*

  ii) if $a \Rightarrow b$ then the clause $(\overline{a} \vee b)$ can be added to $\mathcal{F}$. This will be same if $(a \wedge b)) \Rightarrow false$ occurs.

  iii) if $a \Rightarrow (x_1 \wedge x_2 \wedge \ldots \wedge x_n)$ and $\overline{a} \Rightarrow (y_1 \wedge y_2 \wedge \ldots \wedge y_m)$ then clauses $(x_i \vee y_j), \forall (1 \leq i \leq n)$ *and* $(1 \leq j \leq m)$ can be added to $\mathcal{F}$.
- *Subsuming Look-Ahead*

  By enhancing the principle of look-ahead, you check multiple implications in order to produce subsuming clause. Let be C a clause of the form: $x_1 \vee x_2 \vee \ldots \vee x_i \vee \ldots \vee x_k$.

  iv) if $\overline{x_1} \wedge \overline{x_2} \wedge \ldots \wedge \overline{x_i} \Rightarrow false$ then C should be replace by the clause (*is subsumed by*) $x_1 \vee x_2 \vee \ldots \vee x_i$ in $\mathcal{F}$.

  v) if $(x_1 \wedge \overline{x_2} \wedge \ldots \wedge \overline{x_i}) \Rightarrow false$ then C is subsumed by $x_2 \vee \ldots \vee x_i$

4 SPECIFIC PROBABILITIES

In this section, we define an experimental framework that allows to study at the bit-level the behavior of a hashing process. This framework uses a SAT formula modeling a cryptosystem that has been previously defined in (Legendre et al., 2012).

4.1 From a SAT Formula to get Statistics

The SAT formula $\mathcal{F}$ can be used as a tool. The assignment of variables corresponding to the input message leads the SAT engine to fix all the unassigned variables of $\mathcal{F}$ thanks to a linear and deterministic process named *unit propagation*. This corresponds to the hashing process and the complete assignment $\mathcal{A}$ of variables of $\mathcal{F}$ is a solution. Actually, $\mathcal{A}$ gives useful information on how a variable is set but also how two variables are set in pairs (and more). This last remark is interesting. Thus, memorizing each pair of variables allows us to appreciate the behavior of a variable with respect to another. For instance, let v and w be two boolean variables. From $\mathcal{A}$, looking at v and w at the same time lets to know which of the following subsets of $\mathcal{S}$ appears:

$$\{v = false \land w = false\} \ ; \{v = false \land w = true\}$$
$$\{v = true \land w = false\} \ ; \{v = true \land w = true\}$$

respectively denoted ($\overline{v} \land \overline{w}$), ($\overline{v} \land w$), ($v \land \overline{w}$), ($v \land w$). We establish a protocol to compute statistics from a SAT formula $\mathcal{F}$:

i) Create a random input message

ii) Assign this message and infers from $\mathcal{F}$. It generates $\mathcal{A}$.

iii) From $\mathcal{A}$, for each pairs of variables, memorize the subset which appears in $\mathcal{S}$

iv) goto i) (This loop should be iterated n times)

v) Group and overlay all the subsets and divide by n.

From this, we obtain the probability to be 1 for each couple of variables (v,w), denoted p($v \land w$).

4.2 Preliminary Remarks

The probability of a variable v to be 1 in a general framework is determined by p(v) = p($v \land v$). Moreover, the conditional probability of a variable v given w is determined by:

$$p(v|w) = \frac{p(v \land w)}{p(w)}$$

4.3 General Behavior of MD5

In one hand, we computed for each variable the theoretical probabilities we should have in a general process. In an other hand, we use our SAT formula representing the MD5 hash function in order to compute classical and conditional probabilities. Then, we compared these two type of probabilities. Within this way, we drew a PGM[1] image representing this comparison, vertically sort by step and horizontally sort by type of words (see fig 1). The used notation is the one defined in 2.2. If the pixel is black, the probability derived from the SAT formula differs by the theoretical probability by tending to 0, and if the pixel is white, the probability differs by tending to 1. Consequently, the more the pixel is white or black, the more the practical probability is far from the theory.

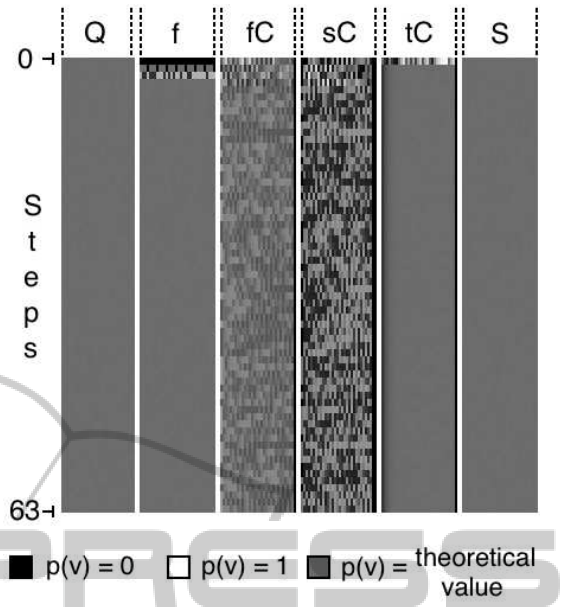

Figure 1: Probability of a variable from the MD5 process, to be 1, from step 0 to 63, sorted by type of 32-bit word on big endian.

In the columns representing the internal states (Q), the non-linear functions (f), the carries of the two operands addition (tC) and the four operands sum (S), the gray is uniform and corresponds to the theoretical probability which is $\simeq \frac{1}{2}$. This means, the words are totally random for each bit, as expected. However, this is not the case for the columns representing the carries of the four operands addition (fC and sC). Hereafter, some details and explanations.

4.3.1 About Carries

The theoretical probabilities to be 1 for the first carry[2] turns around 0.58. Focus on step 17, we get for in-

[1]Portable GrayMap file format

[2]Except the five first least significant bit where the probabilities are slightly higher.

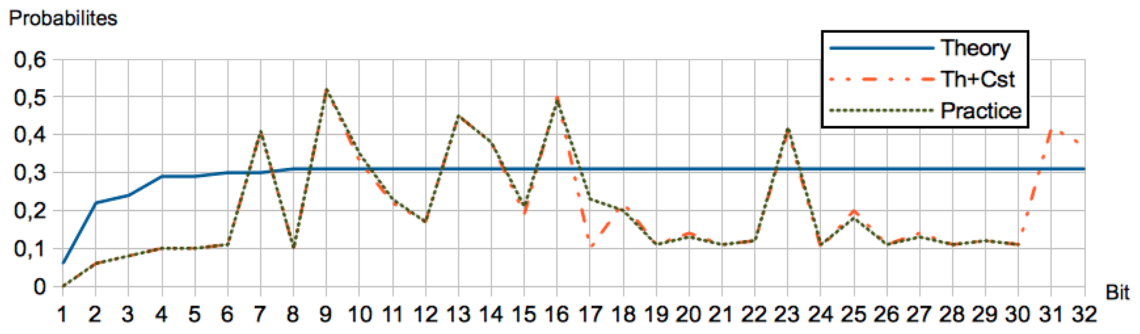

Figure 2: Comparison between theoretical probabilities with (*Th+Cst*) and without (*Theory*) a fixed round constant, and computed probabilities in our benchmark (*Practice*). Here we can observe that the curve representing the theory by considering a constant and the curve extracted from our statistical database are grouped. This means, they have the same behavior, which differs from the theory.

stance p($fC_{17}[5]$) = 0.67 and p($fC_{17}[23]$) = 0.49. The gap with the theory is approximately 15%. However, the gaps are widening even more so if we look at the second carries.
The figure 3 is a zoom in the second carries, step 17. We can observe two things: probabilities are not uniforms (because there are several gray tones) and some go far away of the theoretical probability by tending to 0, for instance the bit 7, and others to 1, for instance the bit 22.

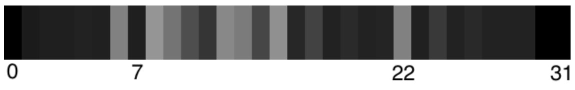

Figure 3: Second carries step 17, on big endian.

Our explanation is this phenomena is due to round constants. Our idea is the fixing of a round constant consists in assigning an operand in the general structure of the addition and has for consequence to create a new structure totally defined by the round constant. To confirm this point, we computed theoretical probabilities of the variables which are involved in this addition by considering one operand fixed. Continuing with the step 17, we concretely fix an operand to the value `0xc040b340` and observe how the structure behaves. The result is unequivocal: the experience confirms the practice because the observed probabilities in practice are very closed to the ones observed when we fixed a round constant (see fig 2 to compare probabilities). This means, in our point of view, round constants weaken the hashing process because several probabilities become very far from their theoretical values. This implies that the MD5 process is not entirely random. As an illustration, the bit 7, step 17 has a probability of 0.10 instead of 0.31 in theory and the bit 22 has a probability of 0.41.

4.4 Gather Probabilistic and Judicious Information from MD5

Once we built this statistical database, we can apply some specific treatments to detect factual relations as for instance fixed variables and equivalences but also probabilistic relations as quasi-fixed variables, quasi-implications or quasi-equivalences.

4.4.1 Factual Relations

We can identify statistics which represent a fixed variable just by extracting probabilities such as p(v) = 0 or p(v) = 1. For instance, in the MD5 process, the variable corresponding to $sC_1[26]$ is always assigned to 0. It's also possible to detect equivalences. Let be v and w two boolean variables. We have:

1) $if\ p(v) = p(v \wedge w),\ then\ v \Rightarrow w$
   In others words, $if\ p(w|v) = 1, then\ v \Rightarrow w$
2) $if\ p(v|w) = p(w|v) = 1,\ then\ v \Leftrightarrow w$

4.4.2 Probabilistic Relations

Let be t a threshold such as $t \leq p(v) < 1$. We call *quasi-fixed variable* a variable such as whatever the instance, the variable has a probability p(v) to be 1 higher than t. This means, the variable v (resp $\overline{v}$) is set to 1 (resp 0) in (p(v)*100) % of cases. We call *quasi-implication* a relation between two variables such as:

$$p(w|v) \geq t \text{ and note this relation } v \not\Rightarrow w$$

Finally, we call *quasi-equivalence* a relation between two variables such as:

$$p(w|v) \geq t,\ p(v|w) \geq t \text{ and note this relation } v \not\Leftrightarrow w$$

In this paper, we also talked about *quasi-relation* to mentioned a probabilistic relation.

Table 1: Some of Equivalences, Quasi-Equivalences and Quasi-implications detected from the MD5 and the SHA-1 processes with a threshold $t = 0.99$. The used notation is the one presented in 2.2. Moreover Hx and cHx are variables about the hash.

Relation	From MD5
Equivalences	$\overline{tC_0[0]} \Leftrightarrow Q_1[0]$, $\overline{S_0[25]} \Leftrightarrow Q_1[0]$, $\overline{S_0[0]} \Leftrightarrow fC_0[0]$, $M_0[0] \Leftrightarrow fC_0[0]$ $\overline{Hb[0]} \Leftrightarrow cHb[0]$, $\overline{Ha[0]} \Leftrightarrow cHa[0]$, $\overline{cHd[1]} \Leftrightarrow Hd[1]$ $\overline{Q_{62}[1]} \Leftrightarrow Hd[1]$, $\overline{cHc[1]} \Leftrightarrow Hc[1]$, $\overline{Q_{63}[1]} \Leftrightarrow Hc[1]$ $\overline{Q_{64}[0]} \Leftrightarrow Hb[0]$, $\overline{Q_{61}[0]} \Leftrightarrow Ha[0]$, $Hd[0] \Leftrightarrow Q_{62}[0]$, $Hc[0] \Leftrightarrow Q_{63}[0]$ $f_1[j] \Leftrightarrow Q_1[j]\ for\ j \in \{0,8,9,13,18,22,24,25,26,29,30\}$ $\overline{f_1[j]} \Leftrightarrow Q_1[j]\ for\ j \in \{1,2,4,5,6,10,12,14,17,20,21,28\}$
Quasi-Equivalences	$\overline{Q_1[21]} \nLeftrightarrow M_0[14]$, $Q_{61}[8] \nLeftrightarrow Ha[8]$, $Q_{61}[8] \nLeftrightarrow cHa[8]$ $Ha[8] \nLeftrightarrow Q_{61}[8]$, $Ha[8] \nLeftrightarrow cHa[8]$, $cHa[8] \nLeftrightarrow Q_{61}[8]$, $M_0[14] \nLeftrightarrow f_1[21]$ $cHa[8] \nLeftrightarrow Ha[8]$, $f_1[21] \nLeftrightarrow M_0[14]$, $M_0[14] \nLeftrightarrow Q_1[21]$
Quasi-Implications	$M_0[14] \nRightarrow Q_1[21]$, $Q_1[21] \nRightarrow M_0[14]$, $Ha[8] \nRightarrow Q_{16}[8]$, $Hb[28] \nRightarrow cHb[30]$ $Q_{61}[8] \nRightarrow Ha[8]$, $cHa[8] \nRightarrow Q_{61}[8]$, $Q_{64}[28] \nRightarrow cHb[30]$, $cHa[8] \nRightarrow Q_{61}[8]$ $Q_{61}[8] \nRightarrow Ha[8]$, $Ha[8] \nRightarrow Q_{61}[8]$, $cHa[8] \nRightarrow Ha[8]$, $Ha[8] \nRightarrow cHa[8]$
Relation	From SHA-1
Equivalences	$\overline{sC_0[28]} \Leftrightarrow fC_0[28]$, $\overline{sC_0[27]} \Leftrightarrow fC_0[27]$, $\overline{sC_0[25]} \Leftrightarrow fC_0[25]$, $\overline{sC_0[24]} \Leftrightarrow sC_0[17]$ $\overline{sC_0[21]} \Leftrightarrow fC_0[21]$, $\overline{sC_0[20]} \Leftrightarrow fC_0[20]$, $\overline{sC_0[19]} \Leftrightarrow fC_0[19]$ $\overline{sC_0[15]} \Leftrightarrow fC_0[15]$, $\overline{sC_0[12]} \Leftrightarrow fC_0[12]$
Quasi-Equivalences	$sC_0[28] \nLeftrightarrow sC_0[17]$, $sC_0[27] \nLeftrightarrow fC_0[28]$, $sC_0[24] \nLeftrightarrow cHc[7]$, $sC_0[24] \nLeftrightarrow fC_0[28]$ $sC_0[17] \nLeftrightarrow cHc[7]$, $sC_0[17] \nLeftrightarrow fC_0[28]$, $fC_1[16] \nLeftrightarrow sC_1[16]$, $sC_1[16] \nLeftrightarrow fC_1[16]$
Quasi-Implications	$\overline{M_0[8]} \nRightarrow sC_1[14]$, $M_1[13] \nRightarrow sC_1[14]$, $M_1[10] \nRightarrow sC_1[14]$ $M_2[25] \nRightarrow sC_2[27]$, $M_3[27] \nRightarrow sC_3[27]$

4.4.3 Application to MD5 and SHA-1

In this part, we just focus on the detection of probabilistic relations, because detecting factual relations is already done by the logical preprocessing (section 3). In practice, we applied a specific treatment which extract all the specific probabilities that correspond to logical simplifications. For instance, by defining $t = 0.99$, we found p($\overline{M_0[14]}$ | $f_2[21]$) = 0.9969 from the MD5 process. This means we have $f_2[21] \nRightarrow \overline{M_0[14]}$ and this can be added to the formula by the new clause ($\overline{f_2[21]} \vee \overline{M_0[14]}$).

We concretely illustrate our experimentations about MD5 and SHA-1 by giving a portion of different relations we detected in the table 1. We can observe for instance a lot of relations in the first steps facilitating the prediction of the behavior of the variables. Furthermore, we have relations between carries but also others involving states, the input message or non-linear functions. In our knowledge, this is the only approach where computed probabilities are concretely put together in order to emerge new relations that could be exploit in practice. To put the stress on the interest of our experimentations, we referenced in the table 2 the number of quasi-fixed variables and quasi-equivalences found from the MD5 and SHA-1

Table 2: Detection of quasi-fixed variables (q-Fixed) and quasi-equivalences (q-Equi) in the MD5 and SHA-1 processes according to a threshold.

	MD5		SHA-1	
Treshold	q-Fixed	q-Equi	q-Fixed	q-Equi
0.995	2	10	1	8
0.99	5	29	5	44
0.985	9	79	8	88
0.98	12	105	13	169

hashing processes, according to a threshold t. We can see that more the threshold is low, more we get information about special relations between variables.

5 EXPORT WEAKNESSES IN A PRACTICAL FRAMEWORK

Logical cryptanalysis (Massacci and Marraro, 2000; Mironov and Zhang, 2006) offers a perfect framework to exploit concrete weaknesses. Generally, it consists in tackle a crypto-system thanks to a two phases process where in a first part the problem is defined under a SAT formalism and in a second part it is solved thanks to the use of SAT solvers. In this manner, weaknesses can be used to help to reduce the practical complexity of a preimage problem during the boolean encoding or directly during the solving phase.

5.1 Reduction of the Practical Complexity

As we find fixed and quasi-fixed variables, we can reduce the practical complexity of a problem by assigning these variables. Indeed, for instance if a variable v is fixed to 1 then all the clauses which contains the corresponding positive literal v are SAT and all the clauses which contains the corresponding negative literal $\bar{v}$ are reduced. Moreover, as we find equivalences and quasi-equivalences, we can replace a variable by an other. This can provoke many simplifications as for instance the ones presented in this paper in 3.
Concretely, we alternatively preprocessed our SAT formulas and injected information uprooted from our detection method. In this way, we reduced the SHA-1 and MD5 formulas according to the statistics in table 3. The formulas's sizes are directly affected by the number of variables (Nb Var), the number of literals (Nb Lit) and the number of clauses (Nb Cl) used to represent the problem. In addition, we also counted the number of binary clauses (Bin cl) in the formulas since these clauses are very interesting to increase the efficiency of SAT solving[3]. In this manner, we can observe that applying our treatments (pp) allows to decrease the sizes of our SAT formulas and enrich the practical complexity of their SAT solving thanks to an importing of binary clauses for both SHA-1 and MD5.

5.2 Improve Heuristic in SAT Solvers

A good way to solve algebraic systems, especially boolean systems is the use of *complete* SAT solvers. These ones are mainly based on the DPLL (Davis et al., 1962) or the CDCL (Zhang et al., 2001) algorithms which consist in a systematic enumeration of truth assignments thanks to a binary search-tree[4] At each node, their is a policy choice made to decide the next variable to be assign.

Our probabilities can be a good mean to improve the splitting choice policy of a dedicated SAT solver. In practice, the heuristic branching compute with a very precise evaluation the new node of the binary tree, but this choice is often difficult. In these cases, probabilities can help to enrich the evaluation, in particular when a variable has a very high probability to be 0 or 1.

[3]Solving a SAT problem is polynomial if it is composed only of binary clauses.

Table 3: Reduction of the SAT formulas sizes for MD5 and SHA-1 with an enrichment of information.

MD5				
	Nb Var	Nb Lit	Nb Cl	Bin cl
Before pp	12749	≈1.23M	224653	381
After pp	12677	≈ 0.98M	206807	2252
SHA-1				
	Nb Var	Nb Lit	Nb Cl	Bin cl
Before pp	12771	≈ 2.21M	375195	908
After pp	12732	≈ 2.18M	374541	1454

5.3 Tackle the Preimage of Hash Functions

In order to tackle the preimage of a hash function, we can instantiate the variables corresponding to a chosen hash in a CNF representing a reduced-step process of the function. Then, thanks to the logical simplifications presented in this paper, both factual and probabilistic (section 3), the formula is then preprocessed to decrease its practical complexity. Finally we apply a SAT engine to search for a solution. If the formula is solved, necessarily all the variables are assigned, even those from the input message.
In practice, our best result in tackling the preimage of MD5 is about 1 round and 12 steps. About SHA-0 and SHA-1 we inverted 1 round 3 steps. The formula for SHA-1 is composed of 75,974 clauses and 3,321 variables. In our knowledge, this is not the best practical result about the inversion of SHA-$\star$ (Cannière and Rechberger, 2008)(Christian, 2010). Nevertheless, our work has an original approach and is hopeful to conclude in better results. Hereafter, an instance of a SHA-1 hash and its corresponding input, obtained at the end of the hash function and not by the compression function.

[4]The reader should refer to (Biere et al., 2009)

1 round 3 steps on SHA-1 *(23 steps)*
Fixed Hash:
```
0x00000000 0x00000000 0x00000000 0x00000000
```
Input found:
```
0x35691c1a 0xead7eb26 0xcac76b0e 0x00000000
0x51e43c45 0xaa8bc12a 0xdb8fa47c 0x00000000
0x637c1517 0x80abea2e 0x9339f44e 0x00000000
0x6367caee 0xbc8920ec 0x1084c8d7 0x45075a9e
```

6 CONCLUSIONS

In this paper [5], we propose a based logical approach to bring out some cryptographic weaknesses in hash functions. Indeed, we noticed that model the function in a binary field ($\mathbb{F}_2$), allows to point out several variables which have not a random behavior, as expected in a hashing process. In this context, certain internal words, especially carries, are weak and the function may be tackled by these partially open doors to get new information.
We confirmed this point through an experimentation where we found equivalences, quasi-equivalences and quasi-implication by two different ways: an automatic logical reasoning and a probabilistic approach. Thanks to the first technique, we show factual relations that could be used in a general case. Moreover, the probabilistic method allow to outline an overview of quasi-relation. This attests that the variables are strongly correlated and their relations can be exploited to gather new information. As a result, we presented a set of equivalences and quasi-equivalences and explain why they exist through an observation of the influence of round constants. Finally, we talked about logical cryptanalysis by importing these weaknesses in SAT formulas. In this sense, we show how to improve an heuristic in SAT solvers and show practical preimage attacks against SHA-1.
In our knowledge, this is the only approach where logical and probabilistic deductions highlights weaknesses in hash functions. Moreover, our method is generic and so we can also export our method on others cryptographic schemes and underline bitwise weaknesses that could be exploited. Interestingly, improving heuristic in SAT solvers seems to be a very hopeful way to improve practical preimage attacks as, nowadays, it does not exist any dedicated solver to logical cryptanalysis.

[5]This work is supported by the Direction Générale de l’Armement : http://www.defense.gouv.fr/dga

REFERENCES

Bard, G. V., Courtois, N. T., and Jefferson., C. (2007). Efficient methods for conversion and solution of sparse systems of low-degree multivariate polynomials over gf(2) via sat-solvers. Cryptology ePrint Archive, Report 2007/024.

Bettale, L., Faugère, J.-C., and Perret, L. (2012). Solving polynomial systems over finite fields: improved analysis of the hybrid approach. In *ISSAC*, pages 67–74.

Biere, A., Heule, M. J. H., Maaren, H. V., and Walsh, T., editors (2009). *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press.

Biham, E. and Shamir, A. (1990). Differential cryptanalysis of des-like cryptosystems. In *CRYPTO*, pages 2–21.

Cannière, C. D. and Rechberger, C. (2008). Preimages for reduced sha-0 and sha-1. In *CRYPTO*, pages 179–202.

Christian, R. (2010). Second-preimage analysis of reduced sha-1. In *Proceedings of the Australasian conference on Information security and privacy*, pages 104–116.

Cook, S. A. (1971). The Complexity of Theorem Proving Procedures. *In* 3rd *ACM Symp. on Theory of Computing, Ohio*, pages 151–158.

Damgård, I. (1989). A design principle for hash functions. In *CRYPTO*, pages 416–427.

Davis, M., Logemann, G., and Loveland, D. (1962). A Machine Program for Theorem-Proving. *Journal Association for Computing Machine*, (5):394–397.

De, D., Kumarasubramanian, A., and Venkatesan, R. (2007). Inversion attacks on secure hash functions using satsolvers. In *SAT*, pages 377–382.

Faugère, J.-C. and Joux, A. (2003). Algebraic Cryptanalysis of Hidden Field Equation (HFE) Cryptosystems Using Gröbner Bases. In *Advances in Cryptology - CRYPTO 2003*, volume 2729, pages 44–60.

Knuth, D. E. (1997). *The art of computer programming, volume 2 (3rd ed.): seminumerical algorithms*. Addison-Wesley Longman Publishing Co., Inc.

Legendre, F., Dequen, G., and Krajecki, M. (2012). Inverting thanks to sat solving - an application on reduced-step md*. In *SECRYPT*, pages 339–344.

Li, C.-M. and Anbulagan (1997). Heuristics based on unit propagation for satisfiability problems. In *the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI97), Nagoya (JAPAN)*, page 366371.

Massacci, F. and Marraro, L. (2000). Logical cryptanalysis as a sat problem. *J.Autom.Reasoning*, pages 165–203.

Matsui, M. and Yamagishi, A. (1992). A new method for known plaintext attack of feal cipher. In *EUROCRYPT*, pages 81–91.

Merkle, R. (1989). One way hash functions and des. In *CRYPTO*, pages 428–446.

Mironov, I. and Zhang, L. (2006). Applications of sat solvers to cryptanalysis of hash functions. In *SAT*, pages 102–115.

Zhang, L., Madigan, C., Moskewicz, M., and Malik, S. (2001). Efficient conflict driven learning in a boolean satisfiability solver. In *ICCAD*.