algorithms

MDPI

Article

Converting of Boolean Expression to Linear Equations, Inequalities and QUBO penalties for Cryptanalysis

Aleksey I. Pakhomchik, Vladimir V. Voloshinov, Valerii M. Vinokur * and Gordey B. Lesovik

Terra Quantum AG, 9400 Rorschach, Switzerland; apa@terraquantum.swiss (A.I.P.); vvo@terraquantum.swiss (V.V.V.); gl@terraquantum.swiss (G.B.L.)
* Correspondence: vv@terraquantum.swiss

Abstract: There exists a wide range of constraint programming (CP) problems defined on Boolean functions depending on binary variables. One of the approaches to solving CP problems is using specific appropriate solvers, e.g., SAT solvers. An alternative is using the generic solvers for mixed-integer linear programming problems (MILP), but they require transforming expressions with Boolean functions to linear equations or inequalities. Here, we present two methods of such a transformation which applies to any Boolean function defined by explicit rules giving values of the Boolean function for all combinations of its Boolean variables. The first method represents the Boolean function as a linear equation in the original binary variables and, possibly, binary ancillaries, which become additional variables of the MILP problem being composed. The second method represents the Boolean function as a set of linear inequalities in the original binary variables and one additional continuous variable (representing the value of the function). The choice between the first or second method is a trade-off between the number of binary variables and number of linear constraints in the emerging MP problem. The advantage of the proposed approach is that both methods reduce important cryptanalysis problems, such as the preimaging of hash functions or breaking symmetric ciphers as the MILP problems, which are solved by the generic MILP solvers. Furthermore, the first method enables to reduce the binary linear equations to quadratic unconstrained binary optimization (QUBO), by the quantum annealer, e.g., D-Wave.

Keywords: constrained programming; quadratic unconstrained binary optimization; hash function; cryptanalysis

Citation: Pakhomchik, A.I.; Voloshinov, V.V.; Vinokur, V.M.; Lesovik, G.B. Converting of Boolean Expression to Linear Equations, Inequalities and QUBO penalties for Cryptanalysis. *Algorithms* **2022**, *15*, 33. https://doi.org/10.3390/a15020033

Academic Editor: Binlin Zhang

Received: 21 December 2021
Accepted: 19 January 2022
Published: 21 January 2022

1. Introduction

There exists a wide range of constraint programming problems defined on Boolean functions of binary variables. One of the common approaches to solving CP problems is using specific appropriate solvers, e.g., SAT solvers. Note that regardless of the SAT-to-MILP relationships, the transformation of the Boolean expression to linear equations or inequalities is a challenge in computing science [1–3]. A rich lore of publications presents the variety of the systems of linear inequalities and linear equations equivalent to basic logical bi-variant functions; see Table 1.

Returning to the SAT problems, one recalls that it targets checking the satisfiability of "a long" Boolean expression, presented in one of two special forms: either the conjunctive normal form (CNF) or the disjunctive normal form (DNF). If the Boolean function has the DNF or CNF representation, then the generation of the corresponding linear inequalities/equations can be done directly on the base of Table 1 because the DNF and CNF are compositions of the elementary bi-variant Boolean operations. This, however, requires a lot of auxiliary variables to be used for interim values of $\wedge$ and $\vee$ operands. This approach applies to all Boolean functions, because any of them can be converted into DNF or CNF. Yet, the main disadvantageous of the "SAT-based" approach to getting the MILP representation is that it is based on the elementary DNF/CNF operands corresponding to a given

Boolean function. The number of these operands can be much more than the number of the function's variables. Thus, the number of the auxiliary variables denoted by z in Table 1 (fortunately, all these z-s may be assumed continuous; they will take 0–1-values "automatically") and inequalities/equations may turn out to be very high, even for a function with a moderate number of variables. Moreover, the resulting system will have a lot of redundant secondary inequalities/equations following from the original ones.

Table 1. Systems of linear inequalities and linear equations for basic Boolean functions.

	Formula	Systems of Linear Inequalities	Linear Equation
AND	$x \wedge y = z$	$\{z \leqslant x,\ z \leqslant y,\ z \geqslant x+y-1,\ z \geqslant 0\}$	$x+y-2z-a=0$
OR	$x \vee y = z$	$\{x \leqslant z,\ y \leqslant z,\ z \leqslant x+y,\ z \leqslant 1\}$	$x+y-2z+a=0$
XOR	$x \oplus y = z$	$\{z \leqslant x+y,\ z \geqslant x-y,\ z \geqslant y-x,\ z \leqslant 2-x-y\}$	$x+y-z-2a=0$
NOR	$\overline{x \vee y} = z$	$\{x \leqslant 1-z,\ y \leqslant 1-z,\ 1-z \leqslant x+y,\ -z \leqslant 0\}$	$x+y+2z-a=1$
NAND	$\overline{x \wedge y} = z$	$\{1-z \leqslant x,\ 1-z \leqslant y,\ -z \geqslant x+y-2,\ 1-z \geqslant 0\}$	$x+y+2z-a=2$

2. Methods

Here, we describe two methods of transforming the cryptographic problem into the MILP or QUBO. We start our presentation with the approach introduced in Section 2.1. This approach is a novel method representing the Boolean function as one linear equation in terms of the original binary variables and, possibly, ancillary binary variables that become additional variables of the obtained MILP problem. The innovative feature of the approach is its increase in the number of binary variables of the ILP problems. Ordinarily, it is believed to even rise the computational complexity of the ILP problem, at least for the generic ILP solvers. However, the emerging quantum annealers (QA) have inspired an upsurge in the interest for constraints in the "equation form" because they may be directly converted to summands of the objective function of the QUBO problems, which suit the QA [4,5]. For a given Boolean function, this method is based on solving the special CP programming problems with the quadratic objective function. On the contrary, the second method formulated in Section 2.2 provides a uniform approach to generating an irreducible system of linear inequalities for any Boolean function.

In contrast to the past study [6], based on the Grover algorithm [7] allowing for quadratic improvement, where a pre-image attack on hash functions using gate-based approach on universal quantum computers was touched upon, we propose to solve the same problem using quantum annealing devices, which offers an advantage of the possibility of utilizing more qubits.

For a Boolean function depending on the big number of binary variables, both methods may become rather time consuming. The approach enabling the speeding up both methods in the case where a given Boolean function is a composite of the set of Boolean functions defined on a smaller number of variables is described in Section 2.4. Computing the complexity of building the system of linear equations and inequalities for Boolean functions is not a crucial obstacle for the approach. The point is that every cryptography algorithm is based on a fixed set of Boolean functions and building the linear inequalities for them should be done only once. In other words, for creating the system of equations or inequalities, we use numerical approaches for every considered algorithm.

Both proposed methods may be used for representing important cryptanalysis problems, such as the pre-imaging of hash functions or breaking symmetric ciphers, as the MILP problems, which are solved by the generic MILP solvers. For the case of the first method using emerging quantum solvers, more details will be given in Section 2.5. In Section 3, the first, "equation and binary ancillas", method is demonstrated for the MD5 and SHA-265 hash functions. Moreover, we consider the symmetric block cipher AES in Section 3.4.

2.1. Transformation to One Equation with Binary Ancillaries

Let us consider Boolean function $f(\vec{x}) = y$, where $\vec{x} \in \{0,1\}^{N_x}$ and denote $\vec{z} = (\vec{x}, y) \in \{0,1\}^{N_x+1}$. Our goal is to find the linear equation with binary ancilla (let N_a be the number of ancillas) and continuous coefficients:

$$\vec{c_z}^T \vec{z} + \vec{c_a}^T \vec{a} = b, \tag{1}$$

such that it satisfies two conditions:

$$\begin{cases} \forall \vec{z} \in \mathrm{F}\ \exists \vec{a} \in \{0,1\}^{N_a} : \vec{c_z}^T \vec{z} + \vec{c_a}^T \vec{a} = b \\ \forall \vec{z} \notin F\ \forall \vec{a} \in \{0,1\}^{N_a} : \vec{c_z}^T \vec{z} + \vec{c_a}^T \vec{a} \neq b \end{cases}, \tag{2}$$

where F is the feasible region ($f(x) = y$). The converting is committed using a constraint programming (CP) solver (e.g., CPLEX). The aim of the CP solver is to find $\vec{c_z}, \vec{c_a}, b$ coefficients; after obtaining them, we can easily build Equation (1). For the running solver, we have to reduce the problem from Equation (2) into a CP problem. For this purpose, we consider each constraint separately. In the beginning, imagine the simplest CP problem, which has no objective function or constraints. In the first property, each feasible pair $(\vec{x}, y)$ adds the new constraint with a new ancilla:

$$\vec{c_z}^T \vec{z} + \vec{c_a}^T \vec{a_z} = b\,. \tag{3}$$

In the second property, for each pair $(\vec{x}, y)$ from unfeasible configurations, we add new 2^{N_a} constraints:

$$\forall \vec{a_v} \in \{0,1\}^{N_a} : \vec{c_z}^T \vec{z} + \vec{c_a}^T \vec{a_v} \neq b\,. \tag{4}$$

Eventually, we have $|F| + 2^{N_a}|F^*|$ constraints ($|F|, |F^*|$—number of feasible and unfeasible configurations) and $N_x + N_a + 1 + |F| \cdot N_a$ variables. This "homogeneous" CP has many solutions (multiplying all the coefficients by the same number also gives a solution). We prefer integer coefficients with a small absolute value; therefore, let $\vec{c_z}, \vec{c_a}, b$ be integers with a range of -100–100 (as we will see later, it is enough for finding coefficients). For obtaining the small absolute value from the CP solver, we add into the objective function in the minimization problem the following term:

$$g \mathrel{+}= \vec{c_z}^2\,. \tag{5}$$

In addition, we would like obtain a positive value for the first element in $\vec{c_z}[0]$, so we add a new term as follows:

$$g \mathrel{-}= \vec{c_z}[0]\,. \tag{6}$$

Moreover, we expand the cost function with a large fine ($\lambda \gg 1$) for ancilla coefficients:

$$g \mathrel{+}= \lambda \cdot \vec{c_a}^2\,. \tag{7}$$

This term is added because linear Equation (1) with the minimum number of ancillaries is much more preferable. Finally, we can write the full CP problem (with bi-linear equation constraints in $\vec{c_a}, \vec{a_z}$) for finding the linear equation:

$$\begin{aligned} &\vec{c_z}^2 - \vec{c_z}[0] + \lambda \cdot \vec{c_a}^2 \to \min_{\vec{c_z}, \vec{a_z}} \\ &\text{subject to} \\ &\vec{c_z}^T \vec{z} + \vec{c_a}^T \vec{a_z} = b \quad (|F| \text{ constraints})\,, \\ &\vec{c_z}^T \vec{z} + \vec{c_a}^T \vec{a_v} \neq b \quad (2^{N_a}|F^*| \text{ constraints})\,, \end{aligned} \tag{8}$$

where $\vec{a_v}$ are the fixed values of variables $\vec{a_z}$. Problem (8) is the mixed-integer CP and CPLEX CP optimizer can handle this problem. However, the problem with the chosen

number of ancillary variables may be infeasible. In this case, we add one more ancilla and try to solve the updated problem again. If the new problem is also infeasible, we add one more ancilla and so on. We continue to increase the number of ancillary variables until a feasible solution is found.

Consider, for example, AND gate ($y = x_1$ AND x_2) for which we will find the corresponding linear equation. It has the following truth table:

x_1	x_2	y	
0	0	0	feasible
0	1	0	feasible
1	0	0	feasible
1	1	1	feasible
0	0	1	infeasible
0	1	1	infeasible
1	0	1	infeasible
1	1	0	infeasible

(9)

All feasible combinations generate four constraints, whereas infeasible configurations add eight inequalities. In total, the CP problem is as follows:

$$\begin{aligned}
&\min\left({c_{x_1}}^2 + {c_{x_2}}^2 + {c_y}^2 - c_{x_1} + \lambda \cdot {c_a}^2\right)\\
&\text{such that}\\
&\left.\begin{aligned}
&c_{x_1}\cdot 0 + c_{x_2}\cdot 0 + c_y\cdot 0 + c_a a_0 = b\\
&c_{x_1}\cdot 0 + c_{x_2}\cdot 1 + c_y\cdot 0 + c_a a_1 = b\\
&c_{x_1}\cdot 1 + c_{x_2}\cdot 0 + c_y\cdot 0 + c_a a_2 = b\\
&c_{x_1}\cdot 1 + c_{x_2}\cdot 1 + c_y\cdot 1 + c_a a_3 = b
\end{aligned}\right\}\text{Feasible part}\\
&\left.\begin{aligned}
&c_{x_1}\cdot 0 + c_{x_2}\cdot 0 + c_y\cdot 1 + c_a\cdot 0 \neq b\\
&c_{x_1}\cdot 0 + c_{x_2}\cdot 0 + c_y\cdot 1 + c_a\cdot 1 \neq b\\
&c_{x_1}\cdot 0 + c_{x_2}\cdot 1 + c_y\cdot 1 + c_a\cdot 0 \neq b\\
&c_{x_1}\cdot 0 + c_{x_2}\cdot 1 + c_y\cdot 1 + c_a\cdot 1 \neq b\\
&c_{x_1}\cdot 1 + c_{x_2}\cdot 0 + c_y\cdot 1 + c_a\cdot 0 \neq b\\
&c_{x_1}\cdot 1 + c_{x_2}\cdot 0 + c_y\cdot 1 + c_a\cdot 1 \neq b\\
&c_{x_1}\cdot 1 + c_{x_2}\cdot 1 + c_y\cdot 0 + c_a\cdot 0 \neq b\\
&c_{x_1}\cdot 1 + c_{x_2}\cdot 1 + c_y\cdot 0 + c_a\cdot 1 \neq b
\end{aligned}\right\}\text{Infeasible part}
\end{aligned} \tag{10}$$

Each of the new variables, a_0, a_1, a_2, a_3, ensures that there exists at least one value of ancilla which satisfies the linear equation. After solving this CP problem, we obtain the following linear equation:

$$x_1 + x_2 - 2y - a = 0. \tag{11}$$

By this approach, we can get linear equations for basic bi-variant Boolean functions:

	Formula	Linear equation
AND	$x \wedge y = z$	$x + y - 2z - a = 0$
OR	$x \vee y = z$	$x + y - 2z + a = 0$
XOR	$x \oplus y = z$	$x + y - z - 2a = 0$
NOR	$\overline{x \vee y} = z$	$x + y + 2z - a = 1$
NAND	$\overline{x \wedge y} = z$	$x + y + 2z - a = 2$

(12)

2.2. Transformation to a System of Linear Inequalities

Let us take the Boolean function $f\colon \mathbb{B}^n \to \{0,1\}$ of n binary variables $\vec{x} = (x_1, x_2, \ldots, x_n)$. Here $\mathbb{B}^n \doteq \{\vec{x} : x_i \in \{0,1\}\}$, $\mathbb{B}^n \subset \mathbb{R}^n$ is a set of all 2^n vertices of an n-dimensional unit cube $U^n \doteq \{\vec{x} : 0 \leqslant x_i \leqslant 1\}$, i.e., the set of all 0–1 arguments of n-variant Boolean function.

For a given Boolean function f, we consider (as in the previous section) the set F of 2^n vectors in $\mathbb{R}^{n+1}$ (a kind of graph of function f) and their convex hull:

$$\begin{array}{rcl} F & \doteq & \{(\vec{x}, f(\vec{x})) : \vec{x} \in \mathbb{B}^n\}, \\ \mathsf{U}[F] & \doteq & \mathsf{conv}(F). \end{array} \tag{13}$$

From the definitions, we have $F \subset \mathbb{B}^{n+1}$ and $\mathsf{U}[F] \subset U^{n+1}$.

Let us establish important properties of vectors from F and polyhedron $\mathsf{U}[F]$.

Theorem 1. *Let F and $\mathsf{U}[F]$ be defined by (13). Then, we have the following:*

(*a*) *Every vector $(\vec{x}, f(\vec{x})) \in F$ is an extreme point of convex hull $\mathsf{U}[F]$ (i.e., $(\vec{x}, f(\vec{x}))$ is a vertex of polyhedron $\mathsf{U}[F]$);*

(*b*) *If $(\vec{x}, y) \in \mathsf{U}[F]$ and $\vec{x} \in \mathbb{B}^n$ then $y = f(\vec{x})$.*

Proof. Let us take any $(\vec{x}_k, f(\vec{x}_k)) \in F$. If it is not an extreme point of $\mathsf{U}[F]$, then there exists a nontrivial convex combination (hereinafter $\vec{x}_q \in B^n (q = 1{:}2^n)$)

$$(\vec{x}_k, f(\vec{x}_k)) = \sum_{q=1:2^n} \lambda_q (\vec{x}_q, f(\vec{x}_q)),\ \lambda_q \geqslant 0, \lambda_k = 0,\ \sum_{q=1:2^n} \lambda_q = 1. \tag{14}$$

However, (14) implies impossible equation

$$\vec{x}_k = \sum_{q=1:2^n} \lambda_q \vec{x}_q,\ \lambda_q \geqslant 0, \lambda_k = 0,\ \sum_{q=1:2^n} \lambda_q = 1,$$

because $\vec{x}_k \in B^n$ is a vertex of unit cube U^n and cannot be represented as a convex combination of other vertices of $\mathsf{U}[F]$. So, item (a) is proved.

The proof of item (b) is quite similar. Let $\vec{x}_k \in \mathbb{B}^n$. If $(\vec{x}_k, y) \in \mathsf{U}[F]$, then

$$(\vec{x}_k, y)) = \sum_{q=1:2^n} \lambda_q (\vec{x}_q, f(\vec{x}_q)),\ \lambda_q \geqslant 0,\ \sum_{q=1:2^n} \lambda_i = 1. \tag{15}$$

So, we have

$$\vec{x}_k = \sum_{q=1:2^n} \lambda_q \vec{x}_q,\ \lambda_q \geqslant 0,\ \sum_{q=1:2^n} \lambda_q = 1,$$

which implies $\lambda_k = 1$ and $y = f(\vec{x}_k)$ (see (15)). Theorem 1 is proved. □

Thus, the function f may be uniquely defined by corresponding convex polyhedron $\mathsf{U}[F]$. For example, see in Figure 1 examples of polyhedrons of Boolean functions on two variables.

As it is well known in the convex analysis (e.g., see the classical book [8], Section 19) any polyhedron may be uniquely represented either by the set of its vertices, the so-called V-representation, or by the system of linear inequalities, the so-called H-representation. The sought system of the linear inequalities is the H-representation of the polyhedron $\mathsf{U}[F]$. Therefore, there exists a system of linear inequalities of $n+1$ variables $(\vec{x}, x_{n+1})$:

$$\mathsf{S}(\vec{x}, x_{n+1}) = \{a_i + l_i(\vec{x}) + b_i x_{n+1} \geqslant 0 : i = 1, \ldots, m\} \tag{16}$$

(where $a_i, b_i, b_i \neq 0$ are scalars, and $l_i(\vec{x})$ are linear functions of n variables) such that

$$\mathsf{U}[F] = \{(\vec{x}, x_{n+1}) : a_i + l_i(\vec{x}) + b_i x_{n+1} \geqslant 0 : i = 1, \ldots, m\}.$$

Finally, from Theorem 1 we have

$$\forall \vec{x} \in \mathbb{B}^n : \{f(\vec{x}) = x_{n+1}\} \Leftrightarrow \{a_i + l_i(\vec{x}) + b_i x_{n+1} \geqslant 0 : i = 1, \ldots, m\}, \tag{17}$$

i.e., for any $\vec{x} \in \mathbb{B}^n$, $f(\vec{x}) = x_{n+1}$ if and only if $\mathsf{S}(\vec{x}, x_{n+1})$ holds. It is important that due to item (b) of Theorem 1, we do not need to assume that x_{n+1} is a discrete, 0–1, variable. For any 0–1 vector $\vec{x}$, the system $\mathsf{S}(\vec{x}, x_{n+1})$ has the only solution $(\vec{x},\ x_{n+1})$, where $x_{n+1}{=}f(\vec{x})$.

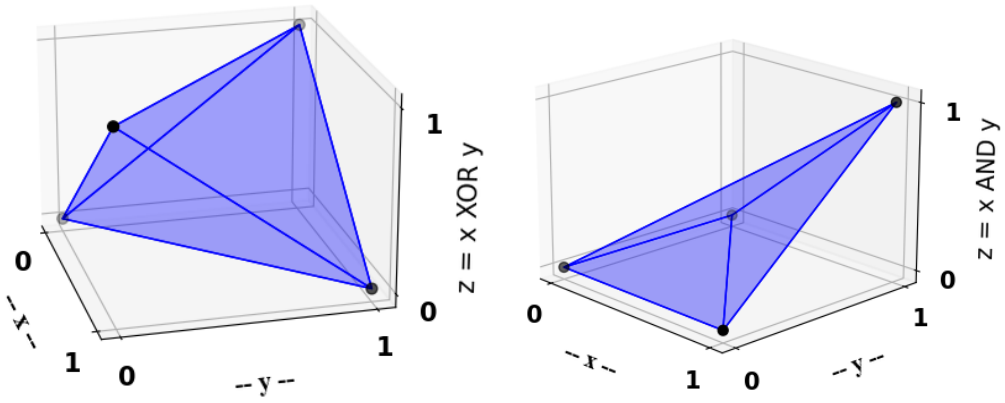

Figure 1. Examples of polyhedrons for bi-variant Boolean functions. (**a**) XOR function. (**b**) AND function.

In Refs. [9–11], one can find a description of one of the known algorithms to obtain the V-representation from an H-representation and vice versa. It is based on the pivoting method (similar to that used in the simplex method of linear programming) of polyhedron vertices/facets enumeration.

There are a number of licensed and free software to handle polyhedra. The following two (free and open source) are rather popular: the CDD [12] and the LRS [13]. Both applications have similar functionality and are compatible in data formats (input and output ones). In particular, they can calculate V- and H-representations of convex hulls of a finite set of points in Euclidean spaces. It is significant that the irreducible H-representation is produced, i.e., all the redundant linear inequalities are eliminated. It decreases the complexity of the resulting MILP representation of discrete optimization problems with Boolean functions.

The available experience of using the CDD and LRS suggests to choose the LRS application for practical usage of the approach. It is slightly newer and has parallel implementation of the reverse search algorithm (on which both above applications are based). Moreover, in the LRS, all the computations may be done without any loss of accuracy in the case of polyhedra with integral-valued vertices (as it is with $\mathsf{U}[F]$). All linear systems for basic Boolean bi-variant functions presented in Table 1 may be obtained by the approach outlined above. We take as an example the following Boolean function that is used as a round function in cryptographic algorithms

$$J(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_2 \wedge x_3) = x_4 \tag{18}$$

to compare "handmade" and "automatically generated" systems of inequalities.

As it was mentioned in the Introduction, any Boolean function given explicitly in the DNF or CNF form may be presented as a system of linear inequalities on the base of such representations for elementary bi-variant Boolean functions (see Table 1). Here, we need four ancillary continuous variables p_j, q_j, u_j, v_j. Let

$$p_j = (x_1 \wedge x_2),\ q_j = (x_1 \wedge x_3),\ u_j = (p_j \vee q_j),\ v_j = (x_2 \wedge x_3).$$

So, we have $x_4 = u_j \vee v_j$. The systems for AND and OR functions from Table 1 let us obtain the following system equivalent to (18)

$$\begin{aligned}
&p_j \leqslant x_1,\ p_j \leqslant x_2,\ p_j \geqslant x_1 + x_2 - 1,\ p_j \geqslant 0,\\
&q_j \leqslant x_1,\ q_j \leqslant x_3,\ q_j \geqslant x_1 + x_3 - 1,\ q_j \geqslant 0,\\
&v_j \leqslant x_2,\ v_j \leqslant x_3,\ v_j \geqslant x_2 + x_3 - 1,\ v_j \geqslant 0,\\
&p_j \leqslant u_j,\ q_j \leqslant u_j,\ u_j \leqslant p_j + q_j,\ u_j \leqslant 1,\\
&u_j \leqslant x_4,\ v_j \leqslant x_4,\ x_4 \leqslant u_j + v_j,\ x_4 \leqslant 1,
\end{aligned} \tag{19}$$

and we do not need to require that x_4, p_j, q_j, u_j, v_j are binaries. Thus, the expression (18) with a 3-variant Boolean function was represented as linear system with 5 continuous ancillas and 20 inequalities.

The LRS gives a more compact system with eight inequalities and only one continuous ancilla, x_4. The reason is that instead of the successive construction of the final system from elementary bi-variant Boolean functions and intermediate (continuous) ancillary variables, the LRS application treats Boolean function as a whole, i.e., as the corresponding polyhedron $\mathsf{U}[J]$ in $\mathbb{R}^4$.

$$\begin{bmatrix} 1\\1\\1\\0\\1\\0\\0\\0 \end{bmatrix} + \begin{bmatrix} -1 & -1 & 0 & 1\\ -1 & -1 & -1 & 2\\ 0 & -1 & -1 & 1\\ 1 & 0 & 1 & -1\\ -1 & 0 & -1 & 1\\ 1 & 1 & 0 & -1\\ 0 & 1 & 1 & -1\\ 1 & 1 & 1 & -2 \end{bmatrix} \begin{bmatrix} x_1\\x_2\\x_3\\x_4 \end{bmatrix} \geqslant \overrightarrow{[\mathbf{0}]} \tag{20}$$

Let us take, for example, a more complex expression with five binary variables $x_1, x_2, x_3, x_4, x_5, x_6$:

$$x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5 = x_6 \tag{21}$$

is equivalent to the system (22) of linear inequalities (in vector-matrix form). Because x_1, x_2, x_3, x_4, x_5 are all binaries, we exclude redundant trivial inequalities ($0 \leqslant x_1 \leqslant 1$) and those with zero coefficients for the x_6 variable. As a result, we have the following 34 linear inequalities, and an output value x_6 will be binary "automatically":

$$\begin{bmatrix} 1 \\ 0 \\ 4 \\ 4 \\ 4 \\ 4 \\ 4 \\ 2 \\ 2 \\ 2 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 4 \\ 2 \\ 2 \\ 2 \\ 2 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ -1 & -1 & -1 & -1 & -1 & 1 \\ -1 & -1 & -1 & -1 & 1 & -1 \\ -1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & 1 & -1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 \\ -1 & 1 & -1 & 1 & 1 & -1 \\ 1 & 1 & -1 & -1 & -1 & 1 \\ 1 & -1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 & 1 & 1 \\ -1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & 1 & -1 \\ 1 & 1 & -1 & 1 & -1 & -1 \\ 1 & -1 & 1 & 1 & -1 & -1 \\ -1 & 1 & 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 & 1 & -1 \\ 1 & -1 & 1 & -1 & 1 & -1 \\ -1 & 1 & 1 & -1 & 1 & -1 \\ -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 & -1 & 1 \\ -1 & 1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & -1 & 1 & 1 \\ -1 & -1 & -1 & 1 & -1 & -1 \\ -1 & 1 & -1 & 1 & -1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 1 & -1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} \geqslant \overrightarrow{[\mathbf{0}]} \tag{22}$$

There are two main disadvantageous of the demonstrated approach:

(1) To obtain polyhedron $\mathsf{U}[F]$ for n-variant function $f : \mathbb{B}^n \to \{0.1\}$, we need to calculate all 2^n values $f(\vec{x})$ for all $\vec{x} \in \mathbb{B}^n$;

(2) The complexity of the facet enumeration algorithm grows dramatically as the value of n increases.

For example, let us take the following function of 10 variables

$$f(x_1, x_2, \ldots, x_{10}) = \begin{cases} 1, \text{ if } \sum\limits_{i=1:10} x_i \ = \ 0 \ (\text{mod } 3); \\ 0, \text{ otherwise} \end{cases}$$

The LRS application gives a desired system with 2691 inequalities (with 10 binary and one continuous variables) in 55 min of running on one half of the computing power of Intel i7 @ 3.4 GHz (4 CPU core of 8 was used).

At this point, we cannot offer analytical evaluations neither for the upper nor for the lower bounds of complexity of the reverse search algorithm implemented by the LRS application, nor for the number of inequalities (i.e., number of facets of polytope corresponding to 0–1 function of binary variables). However, numerical experiments with the number of 10 variant functions (on the Intel i7 computer mentioned above) give a wide range of run times (from an hour to 10 min) and various numbers of inequalities (from more

than 2500 to 12). So, the complexity and the number of inequalities depend substantially on the specific considered Boolean function.

2.3. Comparison of Two Ways of Reducing Boolean Expressions to MILP Problem

So, we have two approaches of reducing the cryptography problem to the MILP problem:

1. By equations with binary ancillaries, see Section 2.1—Boolean expressions of cryptography algorithms are reduced to equations with binary ancillaries; CPLEX-solver is used to obtain coefficients of the desired equation for a given Boolean expression;
2. By inequalities without binary ancillaries, see Section 2.2—Boolean expressions are replaced with systems of linear inequalities without any extra binary variables; polyhedra-handling application LRS [13] is used to generate these inequalities in explicit form.

The first approach is suited mainly to quantum annealers (such as D-Wave) because all equations may be directly converted to summands of the QUBO criterion (just by squaring). At the same time, both MILP and QUBO problems may be (theoretically) solved by classical, generic solvers (CPLEX, Gurobi, SCIP, CBC, etc.). However, extra binary ancillaries may increase the computing complexity of the problem for classical solvers.

It is important to mention that in the context of the transformation of the equation with a Boolean function to a summand of QUBO objective function (as a penalty for violating that equation), another approach exists. For example, it is known [14] that equation $x_1 \wedge x_2 = y$ is equivalent to bi-linear penalty $P(x_1, x_2, y) = x_1x_2 - 2x_1y - 2x_2y + 3y$ (without any binary ancilla, which appears in the equation for XOR in (12)) in the following sense:

$$\begin{array}{l}(x_1 \wedge x_2 = y) \Leftrightarrow (P(x_1, x_2, y) = 0),\\ (x_1 \wedge x_2 \neq y) \Leftrightarrow (P(x_1, x_2, y) \geqslant 1).\end{array} \tag{23}$$

The second approach is suited to classical MILP solvers only. Theoretically, the absence of binary ancillaries may give an easier MILP problem than that obtained by the first approach. The number of binary variables of the resulting MILP problem will be equal to the number of unknown bits in the hash pre-image problems. However, it is still impossible to evaluate the number of linear inequalities. Comparison of the first and second approaches requires intensive computing experiments. For some Boolean function, there is another, hybrid, method based on the equation with binary ancillas and on a couple of inequalities. This method enables replacing one binary variable with a continuous one.

Assume that we have representation of the Boolean function $f\colon \mathbb{B}^n \to \{0, 1\}$ as a linear equation defined in (1) and (2). Let us rewrite linear Equation (1) to pick out the summand with the y variable, keeping the value of the function

$$c_y y + \vec{c_x}^T \vec{x} + \vec{c_a}^T \vec{a} = b \tag{24}$$

The method is based on the following simple statement.

Theorem 2. *Let, in (24), $|c_y| = 1$; all components of vectors $\vec{c_x}$, $\vec{c_a}$ and scalar b are all integers. Let the following inequalities hold also*

$$0 \leqslant y \leqslant 1. \tag{25}$$

Then Equation (24) and inequalities (25) imply that y is binary "automatically". In other words, in (2), we can drop the constraint that y must be binary variable.

Proof. The proof is very simple. The assumption that $\vec{c_x}$, $\vec{c_a}$ and b are all integers means that if (24) holds, then y is an integer. From inequality (25), we obtain that y can be either 0 or 1. □

Theorem 2 enables us to reduce the complexity of the corresponding MILP by replacing one binary variable with a continuous one at the extent of two additional inequalities.

Unfortunately, of the elementary Boolean function, only the $\oplus$ satisfies this statement. Another example is the round function H(B, C, D), presented further in Table 2.

Table 2. Nonlinear functions in MD5 hash function.

	Boolean Formula	**Linear Equation**
F(B, C, D)	$(B \wedge C) \vee (\neg B \wedge D)$	$B + 3C + 2D - 6F + 2a_0 - 3a_1 + 2a_2 = 0$
G(B, C, D)	$(B \wedge D) \vee (C \wedge \neg D)$	$3B + 2C + D - 6G - 3a_0 + 2a_1 + 2a_2 = 0$
H(B, C, D)	$B \oplus C \oplus D$	$B + C + D - H - 2a = 0$
I(B, C, D)	$C \oplus (B \vee \neg D)$	$B + 2C - D - 2I + a_0 - 4a_1 = 0$

2.4. The Case of Boolean Functions with Many Binary Arguments

Both methods of representation of the Boolean function, either as one linear equation with original variables and binary ancillaries, or as a system of linear inequalities without binary ancillaries, may require a long computing time. If so, the following workaround may be used, based on the representation of Boolean function as a composition of Boolean functions with fewer arguments.

Take the Boolean function $f(\vec{x}) : B^n \to \{0,1\}$, which may be represented as composition of K Boolean functions with fewer arguments:

$$\begin{gathered} f(\vec{x}) = f(x_1, x_2, \ldots, x_n) = \Phi(\vec{Y}(\vec{x})) = \Phi\Big(Y_1(\vec{\xi}_1(\vec{x})), Y_2(\vec{\xi}_2(\vec{x})), \ldots, Y_K(\vec{\xi}_K(\vec{x}))\Big), \text{ where} \\ \forall k \in \{1 \ldots K\} : \Big(Y_k(\vec{\xi}_k) : B^{n_k} \to \{0,1\}, \vec{\xi}_k(\vec{x}) = \Big\{x_{\nu(k,j)} : j \in \{1 \ldots n_k\}\Big\}, n_k \ll n\Big). \end{gathered} \tag{26}$$

2.4.1. Linear Equations with Binary Ancillaries for Composition of Boolean Functions

Let the function $\Phi(\vec{Y})$ have the following representation as a linear equation with binary ancillaries:

$$\Big(\vec{Y} \in B^K, \Phi(\vec{Y}) = z\Big) \Leftrightarrow \Big(\exists \vec{\alpha} \in B^M : E(\vec{Y}, \vec{\alpha}, z) = 0\Big), \tag{27}$$

where $E(\vec{Y}, \vec{a}, z)$ is an affine function (there may be some nonzero constant) on original binary variables $\vec{Y}$; a set of M binary ancillaries $\vec{a}$; and binary scalar z is a value of $\Phi(\vec{Y})$.

Then, let us assume that we know the similar representation for all "sub-functions" Y_k $(k = 1{:}K)$:

$$\Big(\vec{\xi}_k \in B^{n_k}, Y_k(\vec{\xi}_k) = y_k\Big) \Leftrightarrow \Big(\exists \vec{a}_k \in B^{m_k} : e_k(\vec{\xi}_k, \vec{a}_k, y_k) = 0\Big), \tag{28}$$

where $e_k(\vec{\xi}_k, \vec{a}_k, y_k)$ is an affine function on a subset of original binary variables $\vec{\xi}_k(\vec{x})$; a set of m_k binary ancillaries $\vec{a}_k$; and binary scalar y_k is a value of $Y_k(\vec{\xi}_k)$.

Now we can represent original function $f(\cdot)$ (26) as a system of $K{+}1$ linear equations and $K{+}M{+}\sum_{k=1}^{K} m_k$ binary ancillaries (components of vectors $\vec{Y} \in B^K$, $\vec{\alpha} \in B^M$ and $\vec{a}_k \in B^{m_k}, k = 1 : K$):

$$\Big(\vec{x} \in B^n, f(\vec{x}) = \Phi(\vec{Y}(\vec{x})) = z\Big) \Leftrightarrow \begin{cases} \exists \vec{\alpha} \in B^M : E((y_1, y_2, \ldots, y_K), \vec{\alpha}, z) = 0; \\ \exists \vec{a}_1 \in B^{m_1} : e_1(\vec{\xi}_1(\vec{x}), \vec{a}_1, y_1) = 0; \\ \exists \vec{a}_2 \in B^{m_2} : e_2(\vec{\xi}_2(\vec{x}), \vec{a}_2, y_2) = 0; \\ \ldots \\ \exists \vec{a}_K \in B^{m_K} : e_k(\vec{\xi}_K(\vec{x}), \vec{a}_K, y_K) = 0. \end{cases} \tag{29}$$

2.4.2. Linear Inequalities without Binary Ancillaries for Composition of Boolean Functions

Let function $\Phi(\vec{Y})$ have the following representation as a system of linear inequalities without auxiliary binary variables (see (16) and (17)):

$$\left(\vec{Y}\in B^K, \Phi(\vec{Y}) = z\right) \Leftrightarrow \left(\left\{\alpha_i + L_i(\vec{Y}) + \beta_i z \geqslant 0 : i = 1, \ldots, M\right\}\right), \tag{30}$$

where $L_i(\cdot)$ are linear functions. Let the similar representation hold for all functions Y_k (k=1 ... K):

$$\left(\vec{\xi}_k\in B^{n_k}, Y_k(\vec{\xi}_k) = y_k\right) \Leftrightarrow \left(\left\{a_{k,i} + l_{k,i}(\vec{\xi}_k) + b_{k,i}\, y_k \geqslant 0 : i = 1, \ldots, m_k\right\}\right), \tag{31}$$

where $l_{k,i}(\vec{\xi}_k)$, $i = 1 \ldots m_k, k = 1 \ldots K$, are all linear functions.

Now we have the representation of the original function as a system of $M+\sum\limits_{k=1}^{K} m_k$ linear inequalities and original n-vector of binary variables $\vec{x}$, and K+1 continues ancillaries y_k ($k = 1 \ldots K$) and z:

$$\left(\vec{x}\in B^n, f(\vec{x})=\Phi(\vec{Y}(\vec{x}))=z\right) \Leftrightarrow \begin{cases} \left\{\alpha_i + L_i(\vec{Y}) + \beta_i z \geqslant 0 : i = 1, \ldots, M\right\}; \\ \left\{a_{1,i} + l_{1,i}(\vec{\xi}_1(\vec{x})) + b_{1,i}\, y_1 \geqslant 0 : i = 1, \ldots, m_1\right\}; \\ \left\{a_{2,i} + l_{2,i}(\vec{\xi}_2(\vec{x})) + b_{2,i}\, y_2 \geqslant 0 : i = 1, \ldots, m_2\right\}; \\ \ldots \\ \left\{a_{K,i} + l_{K,i}(\vec{\xi}_K(\vec{x})) + b_{K,i}\, y_K \geqslant 0 : i = 1, \ldots, m_K\right\}. \end{cases} \tag{32}$$

2.5. Transform Subproblems from the MILP to QUBO

In this section, we consider converting our MILP problem into the QUBO. The QUBO representation provides an opportunity to run the problem at quantum annealers (e.g., at the D-Wave computer) or at other machines capable of finding the ground state of the Ising model. Let us take the following linear system:

$$A\vec{x} = \vec{b} \Leftrightarrow \sum_j a_{ij}x_j = b_i \quad (\forall i \in \{0 \ldots M-1\}), \tag{33}$$

where $A \in \mathbb{R}^{M\times N}, \vec{b} \in \mathbb{R}^M, \vec{x} \in \{0,1\}^N$. Move all terms from the right-hand side to the left:

$$\sum_j a_{ij}x_j - b_i = 0 \tag{34}$$

Square all terms in the lhs and sum up all of them:

$$\left(\sum_j a_{0j}x_j - b_0\right)^2 + \left(\sum_j a_{1j}x_j - b_1\right)^2 + \cdots + \left(\sum_j a_{(M-1)j}x_j - b_{M-1}\right)^2 \tag{35}$$

One can see that minimizing the expression (35) is equivalent to finding the solution of Equation (33). Summing up, we write the equivalent form for MILP and QUBO as follows:

$$A\vec{x} = \vec{b} \Leftrightarrow \min_{\vec{x}}\left(\left(\sum_j a_{0j}x_j - b_0\right)^2 + \left(\sum_j a_{1j}x_j - b_1\right)^2 + \cdots + \left(\sum_j a_{(M-1)j}x_j - b_{M-1}\right)^2\right) \tag{36}$$

3. Results

3.1. Transform Addition Modulo into the MILP

The addition modulo can be represented in the MILP form as well. Let us take the following addition modulo with the multiply input terms:

$$\sum_{i=0}^{K} a_i = b \Big(\mod 2^N \Big) \tag{37}$$

where $a_i, b \in \mathbb{Z}$. Consider each bit in the equation. Let us take $a_i = \sum_{j=0}^{K} a_{ij} 2^j, b = \sum_{j=0}^{K} b_j 2^j$ (a_{ij} or b_j are j-th bit of a_i or b number respectively). Then, we can write the system of equations that is equivalent to the previous Equation (37):

$$\begin{aligned}
\sum_{i=0}^{N} a_{i0} &= b_0 + \sum_{i=0}^{C_0} \xi_{i,0} 2^i \\
\sum_{i=0}^{1-1} \xi_{C_i - 1, i} + \sum_{i=0}^{N} a_{i1} &= b_1 + \sum_{i=0}^{C_1} \xi_{i,1} 2^i \\
\sum_{i=0}^{2-1} \xi_{C_i - 1, i} + \sum_{i=0}^{N} a_{i2} &= b_2 + \sum_{i=0}^{C_2} \xi_{i,2} 2^i \\
&\vdots \\
\sum_{i=0}^{K-1} \xi_{C_i - 1, i} + \sum_{i=0}^{N} a_{iK} &= b_K + \sum_{i=0}^{C_K} \xi_{i,K} 2^i
\end{aligned} \tag{38}$$

where C_i is a number of carried bits for the i-th bit of a or b. As one can see, Equation (38) is the MILP problem.

3.2. The MD5

In this subsection, we consider converting a pre-image attack on hash functions into an optimization problem. The hash functions, $H(x)$, possesses two features:

1. For any x number, there is easy to compute $H(x)$.
2. For any y number, there is hard to find x such that $H(x) = y$.

The pre-image attack tries to break the second feature. There is a paper in which researchers tried to commit this attack [15]. They used local collision, partial fixing and other classical cryptography techniques. The final complexity in the mentioned paper is $2^{123.4}$. Unfortunately, this is not practical at all.

Another approach based on the SAT for the MD-family hash functions was considered in Refs. [16,17]. In these papers, they used an idea similar to what we use in this work, but they converted the pre-image attack to the SAT problem as we convert it into the MILP problems. In Ref. [16], they successfully committed an attack on the MD4 with 39 rounds under Dobbertin's conditions [18].

Here, we develop an efficient technique to obtain the solution. Namely, we divide the MD5 hash function into simple operations, which we can easily convert to the MILP or QUBO (Figure 2). Each circle depicts an operation; the white rectangle is a 32 bit number. At a first step, we note that all round functions make the same operations, except a nonlinear function. Therefore, the MILP or QUBO will appear in the similar way. Taking a closer look at the round function, we notice that it consists of the following types of operations: the nonlinear function, addition modulo and shift. Nonlinear functions are used in the MD5 to act on 32 bit numbers separately, and thus, we can find the MILP/QUBO representation for each bit separately. Table 2 gives all nonlinear functions in the MD5 and the respective linear equations for them.

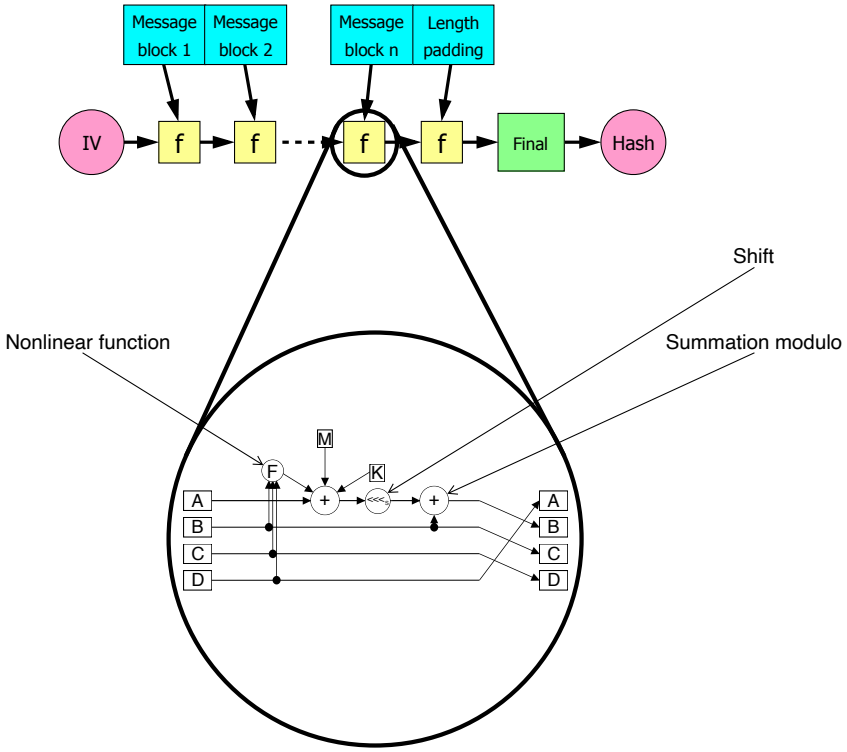

Figure 2. Dividing MD5 hashing to "simple" problems. The circles denote operations and rectangles indicate integer numbers.

The addition modulo is considered in Section 3.1. For implementing a shift operation (left or right bit rotation), one should just rename the corresponding variables. Let us consider the linear system $\vec{\Phi}(\vec{x},\vec{y}) = \vec{b}$, where $\vec{x}$ and $\vec{y}$ are vectors of variables, and $\vec{b}$ is a fixed vector of parameters. We need to add a shift operation to $\vec{x}$ and to leave $\vec{y}$ unchanged. We denote by $\vec{x}\,'$ the "shifted" $\vec{x}$. By definition, $\vec{x} = (x_1, x_2, \ldots, x_N)$, then $\vec{x}\,' = (x_{s+1}, x_{s+2}, \ldots, x_N, x_1, x_2, \ldots, x_s)$ in the case of right rotation $(x' = x >>> s)$ and $\vec{x}\,' = (x_{N-s}, x_{N-s+1}, \ldots, x_N, x_1, x_2, \ldots, x_{N-s-1})$ in the case of left rotation $(x' = x <<< s)$. So, the new linear system (after shift) is just the system of simple equations ("reindexing") $\vec{\Phi}'(\vec{x},\vec{y}) = \vec{b} \Leftrightarrow \vec{\Phi}(\vec{x}\,',\vec{y}) = \vec{b}$.

After converting all types of operations in the MILP, we are ready to build it for the whole MD5. Before doing that though, let us shortly revise the definition of MD5. The hash function transforms the 512 bit input message into the hash value. The given message divides into 16 pieces of 32 bit numbers, $M_i (i \in 0 \ldots 15)$. In addition, the MD5 has four 32-bit constants, which we define as A_0, B_0, C_0, D_0, respectively. The calculating of the hash value consists of rounds. The MD5 has 64 rounds. In each round, we calculate some intermediate hash value as follows:

$$\begin{aligned} &A_k = D_{k-1}, B_k = \Big(F_k(B_{k-1}, C_{k-1}, D_{k-1}) + A_{k-1} + M_{\mu(k)} + K_k\Big) <<< s_k + B_{k-1}, \\ &C_k = B_{k-1}, D_k = C_{k-1} \end{aligned} \tag{39}$$

where $k \in 1 \ldots 64$ is the number of rounds, K_k and s_k are given constant numbers, and $\mu(k)$ are as follows:

1. $\mu(k) = k - 1 \quad (k \in 1 \ldots 16)$;
2. $\mu(k) = 5k - 4 (\mathrm{mod}\ 16) \quad (k \in 17 \ldots 32)$;
3. $\mu(k) = 3k + 2 (\mathrm{mod}\ 16) \quad (k \in 33 \ldots 48)$;
4. $\mu(k) = 7k - 7 (\mathrm{mod}\ 16) \quad (k \in 49 \ldots 64)$.

Here F_k is one of the nonlinear round functions. Depending on the round number, we have the following:

1. $F_k(B,C,D) = F(B,C,D) = (B \wedge C) \vee (!B \wedge D) \quad (k \in 1 \dots 16)$;
2. $F_k(B,C,D) = G(B,C,D) = (B \wedge D) \vee (C \wedge !D) \quad (k \in 17 \dots 32)$;
3. $F_k(B,C,D) = H(B,C,D) = B \oplus C \oplus D \quad (k \in 33 \dots 48)$;
4. $F_k(B,C,D) = I(B,C,D) = C \oplus (B \vee !D) \quad (k \in 49 \dots 64)$.

The final hash value of the MD5 is compound of $A_{64}, B_{64}, C_{64}, D_{64}$ blocks. Comparing with the notation from the beginning $(H(x) = y), x$ is divided into M_k blocks, y is the compound of $A_{64}, B_{64}, C_{64}, D_{64}$ blocks. More details can be found in [19].

In the Table 2, we specified all nonlinear functions used in MD5. Denote by $L_k(B_{k-1}, C_{k-1}, D_{k-1}, FF_k, \alpha_k) = 0$ the linear system of equations for nonlinear operations in the k-th round. This system consists of the linear equation for each bit of $B_{k-1}, C_{k-1}, D_{k-1}$ and is chosen from Table 2 in accordance with the hash function round. Here, FF_k is the output number of $F_k(B_{k-1}, C_{k-1}, D_{k-1})$, and α_k is a vector variable which denotes all ancilla bits. For example, let us consider the 33-rd round and let all the numbers in the blocks be 2 bit for simplicity's sake. In the 33-rd round, $H(B,C,D)$ is used from Table 2. Then, the linear system $L_{33}(B_{32}, C_{32}, D_{32}, FF_{33}, \alpha_{33}) = 0$ becomes

$$L_{33}(B_{32}, C_{32}, D_{32}, FF_{33}, \alpha_{33}) = 0 := \begin{cases} B_{32,0} + C_{32,0} + D_{32,0} - FF_{33,0} - 2\alpha_{33,0} = 0 \\ B_{32,1} + C_{32,1} + D_{32,1} - FF_{33,1} - 2\alpha_{33,1} = 0 \end{cases} \tag{40}$$

For the addition modulo, we use another notation, $\Sigma(A, B, \dots, D, R, \xi)$, where $A, B, \dots, D$ are input numbers, R is the output one and ξ is a vector variable of all ancilla bits as in the nonlinear function case. Let us write an explicit formula for Σ in the simple case. For example, consider two 2-bit summands, then Σ is as follows:

$$\Sigma(A, B, R, \xi) = 0 := \begin{cases} A_0 + B_0 = R_0 + 2\xi_0 \\ \xi_0 + A_1 + B_1 = R_1 + 2\xi_1 \end{cases} \tag{41}$$

Using the MD5 round definition and Figure 2, we can write the linear system for one round. Consider the k-th round:

$$\begin{cases} L_k(B_{k-1}, C_{k-1}, D_{k-1}, FF_k, \alpha_k) = 0 \\ \Sigma\left(FF_k, A_{k-1}, M_{\mu(k)}, K_k, FM_k, \xi_{k,1}\right) = 0 \\ \Sigma(FM_k <<< s_k, B_{k-1}, B_k, \xi_{k,2}) = 0 \\ A_k = D_{k-1}, C_k = B_{k-1}, D_k = C_{k-1} \end{cases} \tag{42}$$

Combining all of Equation (42), we obtain the MILP for the whole MD5:

$$\begin{cases} L_1(B_0, C_0, D_0, FF_1, \alpha_1) = 0 \\ \Sigma\Big(FF_1, A_0, M_{\mu(1)}, K_1, FM_1, \xi_{1,1}\Big) = 0 \\ \Sigma(FM_1 <<< s_1, B_0, B_1, \xi_{1,2}) = 0 \\ A_1 = D_0, C_1 = B_0, D_1 = C_0 \\ \dots \\ L_k(B_{k-1}, C_{k-1}, D_{k-1}, FF_k, \alpha_k) = 0 \\ \Sigma\Big(FF_k, A_{k-1}, M_{\mu(k)}, K_k, FM_k, \xi_{k,1}\Big) = 0 \\ \Sigma(FM_k <<< s_k, B_{k-1}, B_k, \xi_{k,2}) = 0 \\ A_k = D_{k-1}, C_k = B_{k-1}, D_k = C_{k-1} \\ \dots \\ L_{64}(B_{63}, C_{63}, D_{63}, FF_{64}, \alpha_{64}) = 0 \\ \Sigma\Big(FF_{64}, A_0, M_{\mu(64)}, K_{64}, FM_{64}, \xi_{64,1}\Big) = 0 \\ \Sigma(FM_{64} <<< s_{64}, B_{63}, B_{64}, \xi_{64,2}) = 0 \\ A_{64} = D_{63}, C_{64} = B_{63}, D_{64} = C_{63} \end{cases} \tag{43}$$

with the final hash value $A_{64}, B_{64}, C_{64}, D_{64}$.

Now, we calculate the number of binary variables in the MILP of the MD5 hash function. In the zeroth round, we have A_0, B_0, C_0, D_0 and all $M_i (i \in 0 \dots 16)$. They are $(4 + 16) \cdot 32 = 640$ bits. Each L_k adds F_k and α_k variables, and F_k in each round consists of 32 bits, whereas the number of bits in α_k depends on a round. From 1 to 32 rounds, α_k has $3 \cdot 32 = 96$ bits; from 33 to 48, α_k has 32 bits; and from 49 to 64, α_k has 64 bits. Finally, all L_k have $32 \cdot 64 + 96 \cdot 32 + 32 \cdot 16 + 64 \cdot 16 = 6656$ new bits.

Each $\xi_{k,1}$ has only 2 bits. One can easily check that because four terms generate 2 carry bits and in the next addition, six terms (4 + 2) also produce 2 carry bits. Taking into account a new bit for FM_k and 32 bits, we conclude that Σ adds $3 \cdot 32 = 96$ bits in the case of $\Sigma\Big(FF_k, A_{k-1}, M_{\mu(k)}, K_k, FM_k, \xi_{k,1}\Big) = 0$. Even easier is with $\xi_{k,2}$, which always has only 1 bit. Therefore, $\Sigma(FM_k <<< s_k, B_{k-1}, B_k, \xi_{k,2}) = 0$ adds 64 bits. In total, all Σ in all rounds generate 10,240 new bits. Taking into consideration A_0, B_0, C_0, D_0 and all M_k and excluding $A_{64}, B_{64}, C_{64}, D_{64}$, we get 17,408 bits in total.

3.3. The SHA-256

The SHA-256 hash function has a similar structure as well as the MD5 (see Section 3.2) and also has 64 rounds of the similar transformation; see Figure 3. However, there are some differences (for more details, see [20]), namely the following:

1. Each intermediate hash value has 256 bits, which is divided into 8 blocks with 32 bits.
2. A round has four nonlinear functions (Ch, Ma, $\Sigma 1, \Sigma 2$).
3. The input message has 512 bits (16 blocks with 32 bits), and it is extended to 64 blocks using right circular shift and XOR.

Nevertheless, we can convert the SHA256 into MILP (QUBO) as well. In Table 3, we specified all nonlinear functions used in SHA256 and the relevant linear equation. All the MILPs were built by the method from Section 2.1.

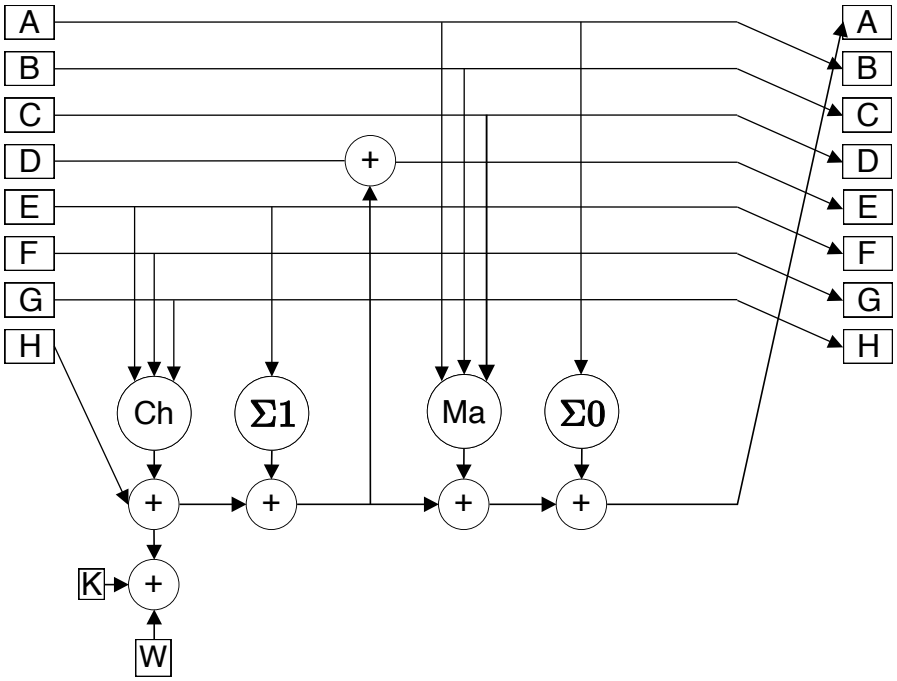

Figure 3. Schematic round of SHA-256 hash function.

Table 3. Non-linear functions in SHA-256 and relevant linear equations.

Boolean Formula	Linear Equation
$\mathrm{Ch(E,F,G)} = (\mathrm{E} \wedge \mathrm{F}) \oplus (\neg E \wedge \mathrm{G})$	$\mathrm{E} + 3\,\mathrm{F} + 2\mathrm{G} - 6\mathrm{Ch} + 2a_0 - 3a_1 + 2a_2 = 0$
$\Sigma_0(A_2, A_{13}, A_{22}) = A_2 \oplus A_{13} \oplus A_{22}$	$A_2 + A_{13} - A_{22} + \Sigma_0 - 2a_0 = 0$
$\Sigma_1(A_{-6}, A_{-11}, A_{-25}) = A_{-6} \oplus A_{-11} \oplus A_{-25}$	$A_{-6} + A_{-11} - A_{-25} + \Sigma_1 - 2a_0 = 0$
$\mathrm{Ma(A,B,C)} = (\mathrm{A} \wedge \mathrm{B}) \oplus (\mathrm{A} \wedge \mathrm{C}) \oplus (\mathrm{B} \wedge \mathrm{C})$	$\mathrm{A} + \mathrm{B} + \mathrm{C} - 2\mathrm{Ma} - a_0 = 0$

The same is with the section about the MD5 (Section 3.2), where we can write the MILP for one round as follows:

$$\begin{cases} L_{Ch}(E_{k-1}, F_{k-1}, G_{k-1}, CH_k, \alpha_{k,1}) = 0, \\ L_{\Sigma_1}(E_{k-1}, SIG_{k,1}, \alpha_{k,2}) = 0, \\ L_{\Sigma_0}(A_{k-1}, SIG_{k,0}, \alpha_{k,3}) = 0, \\ L_{Ma}(A_{k-1}, B_{k-1}, C_{k-1}, MA_k, \alpha_{k,4}) = 0, \\ \Sigma(H_{k-1}, SIG_{k,1}, CH_k, W_k, K_k, S_{k,1}, \xi_{k,1}) = 0, \\ \Sigma(D_{k-1}, S_{k,1}, E_k, \xi_{k,2}) = 0, \\ \Sigma(S_{k,1}, MA_k, A_k, \xi_{k,3}) = 0, \\ B_k = A_{k-1}, C_k = B_{k-1}, D_k = C_{k-1}, F_k = E_{k-1}, G_k = F_{k-1}, H_k = G_{k-1}\,, \end{cases} \tag{44}$$

where $L_{Ch}, L_{\Sigma_1}, L_{\Sigma_0}, L_{Ma}$ are linear systems for nonlinear functions from Table 3, Σ is a system for the addition modulo operation, all α and ξ are vectors of binary ancillas for nonlinear and addition transformations, respectively, W_k is a block of extended an input message, K_k is a given round constant, and $A_{k-1}, B_{k-1}, C_{k-1}, D_{k-1}, E_{k-1}, F_{k-1}, G_{k-1}$ and H_{k-1} are blocks of intermediate hash values from the previous round, whereas $A_k, B_k, C_k, D_k, E_k, F_k, G_k$ and H_k are hash values of the current round.

For extending an input message SHA-256, we use the iterative process, where the first 16 blocks are the same as the input message $W_k = M_k$ $(k \in 1 \dots 16)$, and the next W_k $(k \in 17 \dots 64)$ is computed as follows:

$$\beta_{k,0} = (W_{k-15} >>> 7) \oplus (W_{k-15} >>> 18) \oplus (W_{k-15} >> 3) \tag{45}$$

$$\beta_{k,1} = (W_{k-2} >>> 17) \oplus (W_{k-2} >>> 19) \oplus (W_{k-2} >> 10) \tag{46}$$

$$W_k = W_{k-16} + \beta_{k,0} + W_{k-7} + \beta_{k,1} \Big(\mathrm{mod}\ 2^{32} \Big) \tag{47}$$

Denote by L_{ω_0} and L_{ω_1} the linear system for operation in Equations (45) and (46), respectively. They are built as Σ_0 or Σ_1 in Table 3. $\gamma_{k,0}$ and $\gamma_{k,1}$ are ancilla vector bits for these linear systems. So let us write the full linear system for SHA-256:

$$\begin{cases} W_1 = M_1, W_2 = M_2, \dots, W_{16} = M_{16}, \\ L_{\omega_0}(W_2, \beta_{17,0}, \gamma_{17,0}) = 0, \\ L_{\omega_1}(W_{15}, \beta_{17,1}, \gamma_{17,1}) = 0, \\ \Sigma(W_1, \beta_{17,0}, W_{10}, \beta_{17,1}, W_{17}, \xi_{17,0}) = 0, \\ \dots \\ L_{\omega_0}(W_{49}, \beta_{64,0}, \gamma_{64,0}) = 0, \\ L_{\omega_1}(W_{62}, \beta_{64,1}, \gamma_{64,1}) = 0, \\ \Sigma(W_{48}, \beta_{64,0}, W_{57}, \beta_{64,1}, W_{64}, \xi_{64,0}) = 0, \\ \\ L_{Ch}(E_0, F_0, G_0, CH_1, \alpha_{1,1}) = 0, \\ L_{\Sigma_1}(E_0, SIG_{1,1}, \alpha_{1,2}) = 0, \\ L_{\Sigma_0}(A_0, SIG_{1,0}, \alpha_{1,3}) = 0, \\ L_{Ma}(A_0, B_0, C_0, MA_1, \alpha_{1,4}) = 0, \\ \Sigma(H_0, SIG_{1,1}, CH_1, W_1, K_1, S_{1,1}, \xi_{1,1}) = 0, \\ \Sigma(D_0, S_{1,1}, E_1, \xi_{1,2}) = 0, \\ \Sigma(S_{1,1}, MA_1, A_1, \xi_{1,3}) = 0, \\ B_1 = A_0, C_1 = B_0, D_1 = C_0, F_1 = E_0, G_1 = F_0, H_1 = G_0, \\ \dots \\ L_{Ch}(E_{63}, F_{63}, G_{63}, CH_{64}, \alpha_{64,1}) = 0, \\ L_{\Sigma_1}(E_{63}, SIG_{64,1}, \alpha_{64,2}) = 0, \\ L_{\Sigma_0}(A_{63}, SIG_{64,0}, \alpha_{64,3}) = 0, \\ L_{Ma}(A_{63}, B_{63}, C_{63}, MA_{64}, \alpha_{64,4}) = 0, \\ \Sigma(H_{63}, SIG_{64,1}, CH_{64}, W_{64}, K_{64}, S_{64,1}, \xi_{64,1}) = 0, \\ \Sigma(D_{63}, S_{64,1}, E_{64}, \xi_{64,2}) = 0, \\ \Sigma(S_{64,1}, MA_{64}, A_{64}, \xi_{64,3}) = 0, \\ B_{64} = A_{63}, C_{64} = B_{63}, D_{64} = C_{63}, F_{64} = E_{63}, G_{64} = F_{63}, H_{64} = G_{63}, \end{cases} \tag{48}$$

where union of $A_{64}, B_{64}, C_{64}, D_{64}, E_{64}, F_{64}, G_{64}, H_{64}$ is final hash value of SHA-256.

In a similar way as in the previous section (Section 3.2), we calculate the number of bits in MILP. For SHA-256, the linear system (Equation (48)) has 46,080 bits.

3.4. The AES

The AES is a symmetric-key cryptography algorithm, meaning that the same key is used for both encrypting and decrypting data [21]. The AES has three different sizes of the key (128, 192 and 256 bits) and the fixed block size of an input message (128 bits). Depending on the key size, the algorithm has a different number of rounds (10, 12 or 14). Then, the protocol decomposes into several parts:

1. **KeyExpansion**—round keys are derived from the cipher key using the AES key schedule. AES requires a separate 128-bit round key block for each round plus one more.
2. Initial round key addition:
   (a) **AddRoundKey**—each byte of the state is combined with a byte of the round key using bitwise XOR.
3. 9, 11 or 13 rounds:
   (a) **SubBytes**—a non-linear substitution step where each byte is replaced with another according to a lookup table.
   (b) **ShiftRows**—a transposition step where the last three rows of the state are shifted cyclically a certain number of steps.
   (c) **MixColumns**—a linear mixing operation which operates on the columns of the state, combining the four bytes in each column.
   (d) **AddRoundKey**
4. Final round (making 10, 12 or 14 rounds in total):
   (a) **SubBytes**;
   (b) **ShiftRows**;
   (c) **AddRoundKey**.

The first part prepares the key for encoding. Two to four parts are applied sequentially to an input message. An input message has 128 bits and is represented as 4×4 with one byte element, termed the state. For instance, let b be the input message with 128 bits or equivalently 16 bytes, and $b_0, b_1, \ldots, b_{15}$ are represented as this two-dimensional array:

$$\begin{bmatrix} b_0 & b_4 & b_8 & b_{12} \\ b_1 & b_5 & b_9 & b_{13} \\ b_2 & b_6 & b_{10} & b_{14} \\ b_3 & b_7 & b_{11} & b_{15} \end{bmatrix} \tag{49}$$

As we mentioned before, we divide the algorithm into smaller parts, transform to linear equations (or QUBO) and merge all parts. Note that the mentioned steps consist of XOR, shifting and operations in Rijndael's finite field. The first two transformations are considered in Section 3.2. Therefore, let us take a closer look at the last one.

3.4.1. Rijndael's Finite Field

The several steps are based on the arithmetic in Galois field $\mathrm{GF}(2^8) = \mathrm{GF}(2)[x]/(x^8 + x^4 + x^3 + x + 1)$. In AES, they use two operations in this field:

1. Inverse operation in $\mathrm{GF}(2^8)$;
2. Multiplication by 1, 2 and 3.

The first operation was implemented in Ref. [22]. The calculation of the inverse operation in the $\mathrm{GF}(2^8)$ is a hard problem, but it is easy in the $\mathrm{GF}(2)$, where multiplication is the AND operation; therefore, $1^{-1} = 1, 0^{-1} = 0$. (Overall, 0 does not have the inverse element but for definiteness, we put $0^{-1} = 0$. This agreement is valid for any $\mathrm{GF}(2^p)$, $p \geqslant 1$.) In Refs. [22], the authors reduce sequentially the inversion in $\mathrm{GF}(2^8)$ into multiplication and inversion in $\mathrm{GF}(2^4)$, then into $\mathrm{GF}(2^2)$ and, finally, into $\mathrm{GF}(2)$. For example, the G element in $\mathrm{GF}(2^8)$ can be represented as $G = \gamma_1 y + \gamma_0$ with a multiplication modulo, an irreducible polynomial $r(y) = y^2 + \tau y + \nu$ and $\gamma_0, \gamma_1 \in \mathrm{GF}(2^4)$. Making a similar transformation with step-by-step (more details in [22]) arithmetical operations, only $\mathrm{GF}(2)$ is left. So, the given logic gates are enough for making an inversion in $\mathrm{GF}(2^8)$: the XOR, NAND and NOR. All these operations can be transformed into linear equations (QUBO) using our approach from the previous Sections 2.1 and 2.2. One can see the corresponding

linear equation in Table 4. For one byte, it requires 180 gates (XOR, NAND, and NOR) [22]. Each gate is decomposed into one additional equation and two additional bits (z and a).

Table 4. Linear equation for elementary operations (linear equations were found in the previous patent; also there was the method for their generation).

	Formula	Linear Equation
AND	$x \wedge y = z$	$E_{\wedge}(x,y,a,z) \doteq \{x+y-2z-a=0\}$
OR	$x \vee y = z$	$E_{\vee}(x,y,a,z) \doteq \{x+y-2z+a=0\}$
XOR	$x \oplus y = z$	$E_{\oplus}(x,y,a,z) \doteq \{x+y-z-2a=0\}$
NOR	$\overline{x \vee y} = z$	$E_{\overline{\vee}}(x,y,a,z) \doteq \{x+y+2z-a=1\}$
NAND	$\overline{x \wedge y} = z$	$E_{\overline{\wedge}}(x,y,a,z) \doteq \{x+y+2z-a=2\}$

Multiplications by 1, 2 and 3 in the $\mathrm{GF}(2^8)$ are also easy to implement. Multiplying by 1 is exactly the same number ($a \times 1 = a (a \in \mathrm{GF}(2^8))$. Multiplying by 2 is equivalent to shifting the number left by one, and then XOR'ing the value 0x1B if the high bit was one; otherwise it has to do nothing. Let us build the linear system by steps. $x = [x_0, x_1, \ldots, x_7]$ is the initial number. In the beginning, we make the shifting and XOR with 0x1B (or 00011011 as the binary number). After that, we obtain $z = [z_0, z_1, \ldots, z_7]$. As 0x1B is constant, the XOR gate can be simplified ($a \oplus 0 = a, a \oplus 1 = \overline{a}$). Therefore, after shifting, we have to flip the value if the highest bit of x is equal to 1 and the corresponding bit of 0x1B is also equal to 1. Therefore, the full multiplication is the following linear system:

$$\begin{cases} z_0 = x_7 \\ z_1 = x_7 + x_0 - 2a_0 \\ z_2 = x_1 \\ z_3 = x_7 + x_2 - 2a_1 \\ z_4 = x_7 + x_3 - 2a_2 \\ z_5 = x_4 \\ z_6 = x_5 \\ z_7 = x_6 \end{cases} \tag{50}$$

The equation with two variables can be eliminated because it is just the substitution. So, multiplication by 2 requires 3 additional equations and 6 additional bits. Multiplication by 3 can be considered as follows:

$$x \times 3 = x \times (2 \oplus 1) = x \times 2 \oplus x \tag{51}$$

The XOR operation requires one additional equation and one additional bit per bit. So, multiplication by 3 requires $3 + 8 = 11$ additional equations and 22 additional bits.

3.4.2. The Full MILP for the AES

After handling the step with Rijndael's finite field (Section 3.4.1), other operations are quite easy and consist of only simple gates from Table 4. So we can build the full MILP for AES as in the MD5 case (Section 3.2).

The AES uses a key which can be longer than the message (AES-256 has 256 bits for key and 128 bits for the message). Therefore, one raw message and its cipher are not enough for restoring the full key. However, two known messages (with their ciphers) have a sufficient total length. Therefore, we built two copies of the same linear system (QUBO) for the AES with different known messages (and known) ciphers but linked with the same

key. After optimization of these problem, we obtain the key. However, there exists another approach based on the system of quadratic equations; see [23] and the references therein.

3.5. Summary about Cryptography

In Table 5, we calculate the required number of bits in the MILP and qubits in the QUBO, and they are about the same. More precisely, we show the number of variables for the system of linear equations. For inequalities, the numbers of variables differ but remain close. For the AES, the estimates are made with the order of magnitude accuracy.

Table 5. Comparing table of problems size.

MD5	SHA-256	AES-128	AES-192	AES-256
17,280	47,808	≈90,000	≈2 · 100,000	≈2 · 125,000

Unfortunately, the current D-Wave quantum annealer has limited connectivity between the variables. Therefore, before running an experiment at the D-Wave, a user has to embed the original graph into any of the D-Wave's types of graphs: Chimera or Pegasus. There are methods for embedding an arbitrary graph into these graphs [5,24] which are implemented in the D-Wave software package. We tried to embed our graph from the hash function for one round because the QUBO in each round is the same and connects with a small number of qubits with the QUBO from other rounds. Therefore, embedding the full QUBO based on the merged embeddings is a promising approach since it uses a specific structure of the considered problem. In Table 6, one can see the difference between the number of qubits in the original problem and their number in the problem on the specific graphs of the D-Wave. Unfortunately, we did not implement the corresponding code for the AES yet; this will be a subject of the forthcoming publication.

Table 6. Average number of qubits after embedding for different graphs at D-Wave for one round of hash functions. In the SHA-256 case, D-Wave did not find embedding on the Chimera graph.

	MD5	SHA-256
Original	224	415
Chimera	943.2	-
Pegasus	461.1	1205.9

4. Discussion and Conclusions

We developed the methods for representing any Boolean function as a set of constraints in the mixed-integer linear programming and quadratic unconstrained binary optimization problem with the binary and continuous auxiliary variables. In particular, we investigated the possibility of applications of the developed method for the solution of the cryptographic problems known as a pre-image attack on the hash function, such as the MD5 and SHA256.

The first method represents the Boolean function as one linear equation in the original binary variables and, possibly, ancillary binary variables, which become additional variables of the obtained MILP problem. The method is based on the successive solving of a set of special integer CP problems until the coefficients of the desired linear equation are obtained. Any standard ICP solver may be used; we used the CPLEX solver.

The second method represents the Boolean function as a set of linear inequalities in the original binary variables and one additional continuous variable (representing the value of the function). The method is based on identifying the Boolean function (on n binary variables) with the kind of graphical representation, i.e., the set of 2^n points of the unit cube in $\mathbb{R}^{n+1}$. Then, these points are treated as vertices of a convex polyhedron in $\mathbb{R}^{n+1}$. To obtain the desired system of linear inequalities, the V-representation of this polyhedron should be

converted into *H*-representation, a set of affine half-spaces of polyhedron facets, i.e., a set of the affine inequalities (one inequality per each affine half-space). For this conversion, we used the LRS application [13], which is a general purpose tool for polyhedron handling.

Note that the first method gives fewer constraints (only one equation per Boolean function) but more ancillary binary variables, while the second method does not require any additional binary variables but instead requires a number of additional linear inequalities. The final decision on the choice between the either first or second method is a trade-off between the number of binary variables and the number of linear constraints in the obtained MILP problem.

Both methods can be used for reformulating the important cryptanalysis problems (pre-imaging of hash functions or breaking symmetric ciphers) based on the Boolean expressions for the MILP problems. Moreover, these methods are applied for formulating the QUBO, deriving it from the given Boolean function (see also [4,5]) and can be used as alternative methods for generating the QUBO with a low number of bits.

Note that for the quantum annealers, the first method is preferable, because most promising quantum annealers proceed along the special type of optimization problems, namely the quadratic unconstrained binary optimization one. The straightforward approach to transform the MILP problem into the QUBO is to convert the linear constraints into the quadratic penalties known as summands of the QUBO problem criterion.

Author Contributions: Conceptualization, A.I.P., V.V.V., V.M.V. and G.B.L.; methodology, A.I.P. and V.V.V.; software, A.I.P.; validation, A.I.P. and V.V.V.; formal analysis, A.I.P. and V.V.V.; investigation, A.I.P. and V.V.V.; resources, A.I.P. and V.V.V.; data curation, A.I.P., V.V.V.; writing—original draft preparation, A.I.P. and V.V.V.; writing—review and editing, A.I.P., V.V.V., V.M.V. and G.B.L.; visualization, A.I.P. and V.V.V.; supervision, V.V.V., V.M.V. and G.B.L.; project administration, V.M.V. and G.B.L.; funding acquisition, V.M.V. and G.B.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Our study does not report any data.

Acknowledgments: This work is supported by Terra Quantum AG.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Balas, E.; Mazzola, J.B. Nonlinear 0–1 programming: I. Linearization techniques. *Math. Program.* **1984**, *30*, 1–21. [CrossRef]
2. Milano, M.; Trick, M. Constraint and integer programming. In *Constraint and Integer Programming*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 1–31.
3. Hooker, J.N. Logic-based modeling. In *Handbook on Modelling for Discrete Optimization*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 61–102.
4. Bian, Z.; Chudak, F.; Israel, R.; Lackey, B.; Macready, W.G.; Roy, A. Mapping constrained optimization problems to quantum annealing with application to fault diagnosis. *arXiv* **2016**, arXiv:1603.03111.
5. Bian, Z.; Chudak, F.; Israel, R.; Lackey, B.; Macready, W.G.; Roy, A. Discrete optimization using quantum annealing on sparse Ising models. *Front. Phys.* **2014**, *2*, 56. [CrossRef]
6. Ramos-Calderer, S.; Bellini, E.; Latorre, J.I.; Manzano, M.; Mateu, V. Quantum search for scaled hash function preimages. *Quantum Inf. Processing* **2021**, *20*, 180. [CrossRef]
7. Grover, L.K. A fast quantum mechanical algorithm for database search. In Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing—STOC '96, New York, NY, USA, 24–26 May 1996. [CrossRef]
8. Rockafellar, R.T. *Convex Analysis*; Princeton University Press: Princeton, NJ, USA, 1970; p. 452.
9. Avis, D.; Fukuda, K. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Discret. Comput. Geom.* **1992**, *8*, 295–313. [CrossRef]
10. Fukuda, K. *Polyhedral Computation*; Department of Mathematics, Institute of Theoretical Computer Science ETH Zurich: Zurich, Switzerland, 2020 . [CrossRef]

11. Avis, D.; Fukuda, K.; Picozzi, S. On canonical representations of convex polyhedra. In *Mathematical Software*; World Scientific: Singapore, 2002; pp. 350–360. [CrossRef]
12. Fukuda, K. cdd, cddplus and cddlib Homepage. Swiss Federal Institute of Technology. 1997. Available online: http://www.inf.ethz.ch/personal/fukudak/cddhome/index.html (accessed on 12 July 2021).
13. Avis, D. lrs Homepage. McGill University. 2000. Available online: http://cgm.cs.mcgill.ca/~avis/C/lrs.htm (accessed on 12 July 2021).
14. Rosenberg, I. Reduction of bivalent maximization to the quadratic case. *Cah. Cent. D'etudes Rech. Oper.* **1975**, *17*, 71–74.
15. Sasaki, Y.; Aoki, K. Finding Preimages in Full MD5 Faster Than Exhaustive Search. In *Advances in Cryptology—EUROCRYPT 2009*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 134–152. [CrossRef]
16. Preimage Attack on MD4 Hash Function as a Problem of Parallel Sat-Based Cryptanalysis. *Bull. South Ural State Univ. Ser. Comput. Math. Softw. Eng.* **2017**, *6*, 16–27. [CrossRef]
17. Mironov, I.; Zhang, L. Applications of SAT Solvers to Cryptanalysis of Hash Functions. In *Lecture Notes in Computer Science*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 102–115. [CrossRef]
18. Dobbertin, H. The First Two Rounds of MD4 are Not One-Way. In *Fast Software Encryption*; Springer: Berlin/Heidelberg, Germany, 1998; pp. 284–292. [CrossRef]
19. Rivest, R.L. *The MD5 Message-Digest Algorithm*; RFC, Ed.; RFC 1321; IETF: Fremont, CA, USA, 1992. [CrossRef]
20. Dang, Q.H. *Secure Hash Standard*; Technical Report, National Institute of Standards and Technology: Gaithersburg, MD, USA, 2015. [CrossRef]
21. Daor, J.; Daemen, J.; Rijmen, V. AES Proposal: Rijndael. 1999. Available online: https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.36.640 (accessed on 24 June 2021).
22. Canright, D. A Very Compact S-Box for AES. In *Cryptographic Hardware and Embedded Systems—CHES 2005*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 441–455. [CrossRef]
23. Courtois, N.T. General Principles of Algebraic Attacks and New Design Criteria for Cipher Components. In *Advanced Encryption Standard—AES*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 67–83. [CrossRef]
24. Cai, J.; Macready, W.G.; Roy, A. A practical heuristic for finding graph minors. *arXiv* **2014**, arXiv:1406.2741.