SCIS 2018 2018 Symposium on
Cryptography and Information Security
Niigata, Japan, Jan. 23 - 26, 2018
The Institute of Electronics,
Information and Communication Engineers

Pairing Type Optimization Problem and Its Hardness

Fumitaka Hoshino [1,2] Masayuki Abe [1,3] Miyako Ohkubo [4]

Abstract. An asymmetric pairing requires asymmetric input pair, namely there exist two input data types for an asymmetric pairing. This means crypto designers must choose a data type for each variables consistently in their cryptographic schemes. For some cases, it is actually impossible to satisfy such data type assignments. Even if it is possible, their choice drastically impacts on the efficiency of their schemes. Therefore it is interesting how to satisfy and optimize this assignment, but it becomes a complicated task when the scheme is large. Pairing type satisfiability and optimization problems are formalizations of such tasks. It is known that there exists a polynomial-time algorithm to solve the pairing type satisfiability problem. However it has been unclear how hard the pairing type optimization problem is. In this work, we provide a comprehensive theory of pairing type optimization problem, and show that there exists no algorithm to solve it in the worst case in time polynomial in the size of input, if $\mathbf{P} \neq \mathbf{NP}$.

Keywords: Pairing Type Conversion, Pairing Type Optimization Problem, $\mathbf{P} \neq \mathbf{NP}$.

1 Introduction

1.1 Background

Pairing is a cryptographic primitive frequently employed for elaborate cryptographic schemes like functional encryption, broadcast encryption, searchable encryption, proxy re-encryption, non-interactive zero knowledge proof, and so on [28]. In 1990s, it was mostly used to analyze elliptic curve cryptography, but around 2000, some pioneering cryptographers initiated the research for constructive side of pairing [39, 22], and discovered an elegant solution for a long-standing open problem [29]. Following these works, Boneh and others extensively exploited the possibility of applying pairings to cryptography [11, 12, 10, 9]. Soon after that, it was widely spread among crypto community in the blink of an eye, and sparked a revolution in crypto design. [27]

Informally pairing is defined as a triple of computational Diffie-Hellman groups $(\mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_T)$ together with an efficient bihomomorphism $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_T$. Pairings are classified into the following two classes based on the type of the input pair [19].

Symmetric pairing: There exist non-degenerate efficient homomorphisms between $\mathbb{G}_0$ and $\mathbb{G}_1$ bidirectionally.

Asymmetric pairing: (It is conjectured that) there exists no non-degenerate efficient homomorphism between $\mathbb{G}_0$ and $\mathbb{G}_1$, or even if there exists one, it is at least one-way.

In symmetric pairing, elements on $\mathbb{G}_0$ and $\mathbb{G}_1$ are convertible each other through the efficient homomorphisms. Therefore, in design of cryptographic schemes, $\mathbb{G}_0$ and $\mathbb{G}_1$ are not distinguished, thus they are written simply as $\mathbb{G}$. Symmetric pairing has been a popular choice in early days, but it is known that (candidate) asymmetric pairing can be implemented much more efficiently than symmetric one. Furthermore crypto designers are prompted to employ asymmetric one by the recent progress in crypto analysis for small characteristic finite fields [23, 24, 20, 6, 7, 25].

However an asymmetric pairing requires asymmetric input pair in $\mathbb{G}_0 \times \mathbb{G}_1$, i.e. there exist two input data types for an asymmetric pairing. This means crypto designers must choose a data type for each variables consistently in their cryptographic schemes. For some cases, it is actually impossible to satisfy such data type assignments. Even if it is possible, their choice drastically impacts on the efficiency of their schemes. Therefore it is interesting how to satisfy and optimize this assignment, but it becomes a complicated task when the scheme is large. Such a task is called pairing type conversion. Pairing type satisfiability and optimization problems are formalizations of it.

[1] Secure Platform Laboratories, NTT Corporation, Japan
[2] School of Computing, Tokyo Institute of Technology, Japan
[3] Graduate School of Informatics, Kyoto University, Japan
[4] Security Fundamentals Laboratory, CSR, NICT, Japan

1.2 Related Works

In the literature, there are some results for pairing type conversion. Early works on type conversion, e.g. [34, 15, 14, 16], study and suggest heuristic guidelines for when a scheme allows or resists conversion. To our best knowledge, AutoGroup introduced by Akinyele, Green and Hohenberger in [5] is the first automated conversion system that converts schemes from symmetric pairing to asymmetric one. Given a target scheme described in their scheme description language, the system finds set of 'valid' solutions that satisfy constraints over pairings by using a satisfiability modulo theory solver [17]. It then search for the 'optimal' solution that conforms to other mathematical constraints and ones preferences. When there are number of possible solutions, the performance gets lower. In this pioneering work, the security of the resulting converted scheme was not guaranteed. In [1], Abe et al., established a theoretical ground for preserving security during conversion. Their framework, reviewed in section 2, provides useful theorems for security guarantee. But their conversion algorithm is basically a brute-force search over all possible conversions and it requires exponential time in the number of pairings. In [36], Tango et al. pointed out that there are decision and search versions of pairing type satisfiability problem, and proposed a polynomial time algorithm to solve the decision version based on graph coloring. In [4], Akinyele, Garman, and Hohenberger introduced an upgraded system called AutoGroup+ that integrates the framework of [1] to AutoGroup. Though the system becomes more solid in terms of security, their approach for finding an optimal solution remains the same as before. They cover only small scale cryptographic schemes. In [2], Abe, Hoshino, and Ohkubo proved that there exists a polynomial time algorithm to solve search version of pairing type satisfiability problem. Moreover, they introduced an algorithm using 0-1 Integer Programming (IP) to solve pairing type optimization problem. Instantiated with a widely available IP solver, it instantly converts existing intricate schemes, and can process large scale schemes that involves more than a thousand variables and hundreds of pairings.

1.3 Our Contribution

Although conversion for large scale schemes becomes realistic by [2], there is no guarantee that the exactly optimal solution can be found in practical time for any inputs. Unlike the pairing type satisfiability problem, it is unclear whether one can solve pairing type optimization problem in polynomial time or not. In this work, we provide a comprehensive theory of pairing type optimization problem, and show that if $\mathbf{P} \neq \mathbf{NP}$, there exists no algorithm to solve it in the worst case in time polynomial in the size of input. Under the assumption of $\mathbf{P} \neq \mathbf{NP}$, this fact means that if the exactly optimal solution must be found efficiently, we must modify the model of pairing type optimization to exclude

essentially difficult problems.

2 Preliminary

In this section, we define some notions and notations. Some of them are so standard that they may feel tedious for most of the readers, but we restate them to avoid confusion or ambiguity.

2.1 General Notions and Notations

In accordance with the customs of theoretical computer science and cryptography, computation is modeled by a probabilistic interactive oracle Turing machine. We identify the word Turing machine with algorithm. In this work, we use the following terminology.

polynomial-time algorithm: a deterministic polynomial-time Turing machine.

efficient algorithm: a probabilistic polynomial-time Turing machine, which includes polynomial-time algorithm.

Let T be a probabilistic Turing machine. $T(X)$ denotes that T takes X as an input tape. If T has an output value, $T(X)$ also denotes the output value of T when T takes X as an input tape. To simplify the description, we will often omit some input tapes that are not relevant, e.g. security parameter, common reference string, or random tape. To specify omitted tapes, e.g. we write $T(X)$ as $T(X; 1^\lambda, \mathtt{crs}, r)$ where 1^λ, $\mathtt{crs}$ and r are security parameter, common reference string and random tape, respectively. When random tape r is specified, T should be regarded as a deterministic Turing machine.

Definition 1 ($\wedge, \vee, \oplus, \Rightarrow, \Leftrightarrow, \neg, \bar{\cdot}$)**.** *For two logical statements x, y, we write its logical conjunction, disjunction, exclusive disjunction, implication, and equivalence as $x \wedge y$, $x \vee y$, $x \oplus y$, $x \Rightarrow y$, and $x \Leftrightarrow y$ respectively. For a statement x, we write its logical complement as $\neg x$ or $\bar{x}$.*

In this work, we identify the set of integers $\{0, 1\}$ with the set of truth values. Namely 0 is interpreted as false and 1 as truth. We assume that statements can be deduced as in classical logic.

Definition 2 (Directed Graph). *A directed graph G is defined as*

$$G = (V, E),$$

where V is a finite set called vertex set, and E is a finite multiset of elements in $V \times V$ called edge set. In this work, we assume all directed graphs are simple, which means their edge sets have nether self-loop nor multiedge. Therefore, E is a subset of $V \times V$ in this work. For a given graph G, its vertex set and edge set are written as

$$V(G) \quad \text{and} \quad E(G),$$

respectively. An edge $(x, y) \in V \times V$ is often written as

$$(x \xrightarrow{G} y) \quad \text{or} \quad (y \xleftarrow{G} x)$$

to clear its direction.

Definition 3 (Parent). *If $(x \xrightarrow{G} y) \in E(G)$, x is called a parent of y. If x is a parent of y, we write $x \xrightarrow{G} y$ instead of $(x \xrightarrow{G} y) \in E(G)$.*

Definition 4 (Child). *If $(x \xleftarrow{G} y) \in E(G)$, x is called a child of y. If x is a child of y, we write $x \xleftarrow{G} y$ instead of $(x \xleftarrow{G} y) \in E(G)$.*

Definition 5 (Ancestor). *A vertex $y \in V(G)$ which can reach $x \in V(G)$ (and $y \neq x$) is called an ancestor of x. If y is an ancestor of x, we write $y \overset{G}{\rightsquigarrow} x$. If y is an ancestor of x or x itself, we write $y \overset{G}{\rightsquigarrow} x$.*

Definition 6 (Descendant). *A vertex $y \in V(G)$ reachable from $x \in V(G)$ (and $y \neq x$) is called a descendant of x. If y is a descendant of x, we write $y \overset{G}{\leftsquigarrow} x$. If y is a descendant of x or x itself, we write $y \overset{G}{\lesssim} x$.*

Definition 7 (Strongly Connected). *For any vertices $x, y \in V(G)$, if $x \overset{G}{\lesssim} y$ and $x \overset{G}{\gtrsim} y$, we say x and y are strongly connected to each other, and write $x \overset{G}{\sim} y$.*

The following list is an overview of typical arrow expressions which we use to describe algorithms.

- $X \leftarrow Y$ The value of the expression Y is assigned to the variable X.
- $X \overset{\circ}{\leftarrow} Y$ A new value $X \circ Y$ is assigned to the variable X, for any binary operator $\circ$, e.g. $X \overset{\cup}{\leftarrow} Y$ means $X \leftarrow X \cup Y$.
- $X \overset{\$}{\leftarrow} Y$ X is uniformly selected from Y, assuming that Y is a set.
- $X \overset{\$}{\leftarrow} Y()$ X is randomly selected from the output space of the probabilistic Turing machine Y according to the distribution of Y's output when Y's random tape is uniformly selected.
- $X \overset{\$}{\mapsto} Y$ A probabilistic Turing machine which takes X as an input tape and outputs Y.
- $X \to Y$ All of maps from X to Y, assuming that X and Y are sets, which is identical with Y^X.
- $X \mapsto Y$ the map which returns Y for X.
- $X \Rightarrow Y$ X implies Y assuming that X and Y are logical statements.
- $X \Leftrightarrow Y$ X if and only if Y assuming that X and Y are logical statements.
- $X \xrightarrow{G} Y$ X is a parent of Y in the graph G.
- $X \xleftarrow{G} Y$ X is a child of Y in the graph G.
- $X \overset{G}{\rightsquigarrow} Y$ X is an ancestor of Y in the graph G.
- $X \overset{G}{\leftsquigarrow} Y$ X is a descendant of Y in the graph G.

2.2 HashToPoint

Many cryptographic schemes require a special associated algorithm called `HashToPoint` [11, 13, 16]. In short words, `HashToPoint` is an efficient random sampling over $\mathbb{G}_b$ without knowing nontrivial discrete logarithms, whose inputs and outputs are defined as

$$\mathtt{HashToPoint}_b : \{0,1\}^* \to \mathbb{G}_b.$$

It is known that there exists efficient algorithms to implement $\mathtt{HashToPoint}_b$ [11, 21, 30, 33, 31, 38, 18]. In symmetric pairing, $\mathtt{HashToPoint}_0 = \mathtt{HashToPoint}_1$, because $\mathbb{G}_0 = \mathbb{G}_1 = \mathbb{G}$, hence we remove the suffix and write as `HashToPoint`. While in asymmetric pairing, $\mathtt{HashToPoint}_0 \neq \mathtt{HashToPoint}_1$. This is problematic for secure type conversion, since some variables, e.g. generators of source group $\mathbb{G}$, must be represented as $\mathbb{G}_0 \times \mathbb{G}_1$ in the converted scheme. That is, if such a variable g is defined as $g \leftarrow \mathtt{HashToPoint}(r)$ for some $r \in \{0,1\}^*$ in the original scheme, $\mathtt{HashToPoint}_0(r)$ and $\mathtt{HashToPoint}_1(r)$ in the converted scheme must share some non-trivial discrete logarithms without knowing them. If obvious trapdoors are acceptable for the scheme, such $\mathtt{HashToPoint}_b$ can be implemented by using Water's hash [40]. However it is unacceptable for "generic secure" conversions, because trapdoors may ruin the security arguments completely. For typical candidates of type 3 pairing, such $\mathtt{HashToPoint}_b$ without obvious trapdoors is unknown. Therefore if a scheme to convert involves `HashToPoint`, its output must be represented as ether $\mathbb{G}_0$ or $\mathbb{G}_1$ but not both in the converted one.

2.3 Dependency Graph

For a pairing-based cryptographic scheme, its dependency graph is defined as a directed graph which represents data flow w.r.t. group operations in all algorithms in the scheme. Each vertex represents a variable storing a group element in the algorithms, and each edge represents its dependency w.r.t. group operations, that is head depends on tail. In figure 1, we show an example of a dependency graph for a program that computes some group operations over Type 1 pairing. The left of figure 1 is an example of an algorithm (in so called pidgin ALGOL) which takes source group elements A, B and D as input, compute C and E via group operations, and outputs a result of pairing $e(C, E)$. The right of figure 1 is a dependency graph that corresponds to the left algorithm.

In a dependency graph, just the relations between source group elements via group operations are described, and all other things are dropped, e.g. the structures of the program like "if-then-

```
Sample(a, A, B, D):
  a ∈ ℤ/pℤ;
  A, B, C, D, E ∈ 𝔾;
  if a = 0 then
    C ← A · B; E ← D;
  else
    C ← D^a; E ← D^3;
  endif
  Output e(C, E);
```

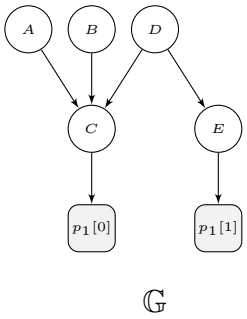

Figure 1: An example of dependency graph.

else", variables storing other than the source group elements like $a \in \mathbb{Z}/p\mathbb{Z}$, and operations in the target group.

In Abe et al.'s framework, one may derive two sub-graphs from a dependency graph which represents a cryptographic scheme. We call deriving the pair of the sub-graphs or the pair itself "split". A split represents a converted scheme, and each sub-graph becomes a dependency graph w.r.t. $\mathbb{G}_0$ or $\mathbb{G}_1$. The converted scheme based on asymmetric pairing will be reconstructed according to the split. In figure 2, we show an example of conversion for the program of figure 1.

```
Sample(a, A, B, (D_0, D_1)):
  a ∈ ℤ/pℤ;
  A, B, C, D_0 ∈ 𝔾_0; D_1, E ∈ 𝔾_1;
  if a = 0 then
    C ← A · B; E ← D_1;
  else
    C ← D_0^a; E ← D_1^3;
  endif
  Output e(C, E);
```

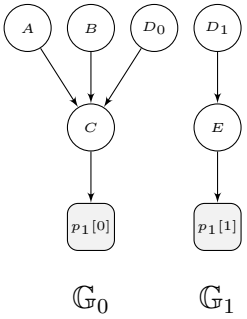

Figure 2: An example of conversion by split.

Definition 8 ($\mathbb{G}, \mathbb{G}_0, \mathbb{G}_1$). *In previous sections, we defined $\mathbb{G}, \mathbb{G}_0, \mathbb{G}_1$ as source groups of symmetric or asymmetric pairing, but hereinafter, we redefine $\mathbb{G}$ as a dependency graph to be splitted or its vertex set as in the figure 1. In the same manner, we redefine $\mathbb{G}_0$ and $\mathbb{G}_1$ as the split i.e. sub-graphs of $\mathbb{G}$ or their vertex sets as in the figure 2. For simplicity, we assume the notation $\mathbb{G}$, $\mathbb{G}_0$, and $\mathbb{G}_1$ include all relevant information about the scheme e.g. which vertices correspond inputs of pairings, outputs of hash functions, and so on. Through this work we assume $\mathbb{G}$ is finite.*

Notice that this abuse of notation may confuse some experts of the cryptographic pairing, e.g. the statement $P \in \mathbb{G}_0 \wedge P \in \mathbb{G}_1$ may mislead them as $P = \mathcal{O}$ which is the identity element of $\mathbb{G}_0$ and $\mathbb{G}_1$, but this is a fallacy because the statement means nothing more than that the variable P is represented as $\mathbb{G}_0 \times \mathbb{G}_1$ in the converted scheme.

Definition 9 (Duplicated node). *If a vertex x satisfies*

$$x \in \mathbb{G}_0 \quad \wedge \quad x \in \mathbb{G}_1,$$

x is called duplicated node, or simply we say x is duplicated.

Definition 10 (Duplicatable/Non-duplicatable). *The node which can be duplicated is called duplicatable. Similarly the node which cannot be duplicated by some reasons is called non-duplicatable.*

Vertices of the outputs of `HashToPoint()` operations must be configured as non-duplicatable nodes, and some nodes may be by crypto designers due to reducing the data size of converted scheme.

Definition 11 (Pairing). *A vertex which represents an input of pairing is called a pairing node.*

We assume that variables storing the inputs of a pairing are declared and defined implicitly. For example, in the right of figure 1 the vertices $p_1[0]$ and $p_1[1]$ correspond to the implicit variables. If there are many pairings in a scheme, we assume that i-th pairing has implicit variables $p_i[0]$ and $p_i[1]$. In a dependency graph, a pairing is described as a pair of leaves, because it is a function which takes two source group elements and outputs one target group element. Therefore a pairing node must be a leaf. Moreover, later we will show that, we can treat all pairing nodes as non-duplicatable.

In general, data flow of a single algorithm (i.e. chains of effective assignments) compose a directed acyclic graph (DAG), thus it has no cycle. However in our cases, the dependency graph consists of multiple algorithms in a scheme. In such a case, it is possible that dependency graph have a cycle, hence, a vertex x can be an ancestor and a descendant of a vertex y simultaneously. Each vertex in a cycle is an ancestor and a descendant of all other vertices in the cycle, therefore we can identify all vertices in the cycle with a single vertex w.r.t. dependency. Considering this property, we define the following notion.

Definition 12 (Reduced Dependency Graph). *A reduced dependency graph is a directed acyclic graph defined by identifying all vertices in each cycle of a dependency graph with a single vertex (and removing all loop edges). We assume each vertex in the reduced dependency graph inherits its properties from the original graph. If a cycle has a non-duplicatable node, we regard the corresponding vertex in the reduced dependency graph as non-duplicatable.*

Reduced dependency graph is nothing other than the quotient graph of $\mathbb{G}$ by the equivalence relation $\overset{\mathbb{G}}{\sim}$, which is often referred to as the strongly connected components quotient graph or the condensation of $\mathbb{G}$. Efficient algorithms are well known for strongly connected components decomposition [37, 3, 32, 8]. For most of our problem, it is enough to consider the reduced dependency graph instead of the original dependency graph. Therefore we assume $\mathbb{G}$ as a DAG hereinafter.

Definition 13 (Leaf). *A leaf is defined as a vertex without out-edge (in the reduced dependency graph).*

By the definition, the inputs of a pairing must be leaves. For reasons stated later we can treat all leaves as non-duplicatable nodes.

In [1, 35], Abe et al. defined a class of split called valid split which guarantees the functionalities and the security of the converted scheme.

Definition 14 (Valid Split). *Let $\mathbb{G}$ be a dependency graph for a cryptographic scheme, $(\mathbb{G}_0, \mathbb{G}_1)$ be a split of $\mathbb{G}$. We say the split $(\mathbb{G}_0, \mathbb{G}_1)$ is valid iff it satisfies all of the following properties:*

1. *merging $\mathbb{G}_0$ and $\mathbb{G}_1$ recovers $\mathbb{G}$,*
2. *for all $b \in \{0, 1\}$, for all vertices $x, y \in \mathbb{G} : y \overset{\mathbb{G}}{\rightsquigarrow} x$, if $x \in \mathbb{G}_b$ then $y \in \mathbb{G}_b$,*
3. *for each pairing, its input nodes $p_i[0]$ and $p_i[1]$ are separately included in $\mathbb{G}_0$ and $\mathbb{G}_1$,*
4. *No node in $\mathbb{G}_0 \cap \mathbb{G}_1$ is non-duplicatable.*

Assume that a split $(\mathbb{G}_0, \mathbb{G}_1)$ satisfies

$$V(\mathbb{G}_0) \cup V(\mathbb{G}_1) = V(\mathbb{G})$$

and the above property 2. For any edge $(x \overset{\mathbb{G}}{\to} y)$ in $E(\mathbb{G})$, we can create a corresponding edge on $\mathbb{G}_0$ or $\mathbb{G}_1$ by the following algorithm.

Algorithm 1 (Edge Interpolation).

Input. $V(\mathbb{G}_b)$ *and* $E(\mathbb{G})$.
Output. $E(\mathbb{G}_b)$.
Step 1. $E(\mathbb{G}_b) \leftarrow \varnothing$;
2. $\forall (x \overset{\mathbb{G}}{\to} y) \in E(\mathbb{G})$,

If $(x \in V(\mathbb{G}_b)) \wedge (y \in V(\mathbb{G}_b))$ *then*

$$E(\mathbb{G}_b) \overset{\cup}{\leftarrow} \{(x \overset{\mathbb{G}}{\to} y)\} ;$$

3. Output $E(\mathbb{G}_b)$ *and halt.*

This means we don't have to care about the edges of a valid split except for ancestor-descendant relationship. Therefore, in the rest of this paper, we often regard $\mathbb{G}$, $\mathbb{G}_0$, and $\mathbb{G}_1$ just as vertex sets. We can always interpolate appropriate edges into $\mathbb{G}_0$ and $\mathbb{G}_1$ by using $E(\mathbb{G})$ if necessary.

3 Pairing Type Optimization Problem

3.1 Abstract Crypto Schemes

In this section, we will define some notions and notations to treat pairing type conversion formally. In previous section, we define reduced dependency graph which represents the data flow w.r.t. group operations in all algorithms in an actual cryptographic scheme. However, for the moment, we will ignore whether a directed graph is derived from a real cryptographic scheme or not.

Definition 15 (Abstract Crypto Scheme). *A directed acyclic graph* $\mathbb{G}$ *with*

$\texttt{NoDup} \subset V(\mathbb{G})$: *set of non-duplicatable nodes, and*

$\texttt{Pair} \subset \{\{x, y\} \mid x, y \in L_{\mathbb{G}}\}$: *set of pairings*

is called an abstract symmetric-pairing-based crypto scheme, or just an abstract crypto scheme. Here $L_{\mathbb{G}}$ *is all of leaves in* $\mathbb{G}$ *and* $\#\bigcup \texttt{Pair} = 2\,\#\texttt{Pair}$.

The word "abstract crypto scheme" is just a rewording for the (reduced) dependency graph. Therefore the notions of split $(\mathbb{G}_0, \mathbb{G}_1)$ and its validity are naturally defined in the same manner. To avoid complicated notation, hereinafter, we will often omit `NoDup` and `Pair`, and simply write $\mathbb{G}$ to express an abstract crypto scheme, except when they are necessary.

Later in this work some logical statements on vertices of abstract crypto schemes will be treated algebraically. To ease translation between logical statements and algebraic relations, we define the following notation.

Definition 16 (Assignment Variable). *For* $x \in V(\mathbb{G})$ *and* $b \in \{0, 1\}$*, we interpret the expression* $(x \in \mathbb{G}_b)$ *as a propositional variable which represents its truth value. We call* $(x \in \mathbb{G}_b)$ *an assignment variable, and define the set of assignment variables* $V_{\mathbb{G}} := \{(x \in \mathbb{G}_b) \mid (x, b) \in V(\mathbb{G}) \times \{0, 1\}\} = V(\mathbb{G}) \times \{0, 1\}$*. To simplify the notation, we define* $\vec{x}$ *as*

$$\vec{x} := \begin{pmatrix} (x \in \mathbb{G}_0) \\ (x \in \mathbb{G}_1) \end{pmatrix}.$$

That is, we will express the Boolean variable which represents the truth-value of $x \in \mathbb{G}_b$ as $(x \in \mathbb{G}_b)$ instead of doing as $X_{x,b}$ or $X(x, b)$. In short,

$$(x \in \mathbb{G}_b) := \begin{cases} 1 & (\text{if } x \in \mathbb{G}_b), \\ 0 & (\text{if } x \notin \mathbb{G}_b). \end{cases}$$

A map $\delta : V_{\mathbb{G}} \to \{0, 1\}$ is called an assignment for the abstract crypto scheme $\mathbb{G}$, or simply an assignment. For $x \in V(\mathbb{G})$ and $b \in \{0, 1\}$, we often write an assignment $\delta(x, b)$ as

$$\delta(x \in \mathbb{G}_b).$$

If an assignment δ is specified, all assignment variables are assigned as

$$(x \in \mathbb{G}_b) := \delta(x \in \mathbb{G}_b),$$

and δ decides a split $(\mathbb{G}_0, \mathbb{G}_1)$ as

$$\mathbb{G}_b := \{x \in \mathbb{G} \mid \delta(x \in \mathbb{G}_b) = 1\}.$$

Similarly, if a split $(\mathbb{G}_0, \mathbb{G}_1)$ is specified, the corresponding assignment δ is uniquely decided. Therefore an assignment δ is also called a split. We will often omit the map δ from statements of assignments, e.g. we say

A valid split $(\mathbb{G}_0, \mathbb{G}_1)$ satisfies that $(x \in \mathbb{G}_b) \Rightarrow (y \in \mathbb{G}_b)$ if $y \overset{\mathbb{G}}{\rightsquigarrow} x$, instead of

A valid split δ satisfies that $\delta(x \in \mathbb{G}_b) \Rightarrow \delta(y \in \mathbb{G}_b)$ if $y \overset{\mathbb{G}}{\rightsquigarrow} x$.

Definition 17 (Abbreviated Assignment Variable). *If vertex* x *is a non-duplicatable node, we regard* x *as a variable over* $\{0, 1\}$*, and define*

$$x := (x \in \mathbb{G}_1).$$

That is for a non-duplicatable node x, we abbreviate $(x \in \mathbb{G}_1)$ as x. For $b \in \{0, 1\}$, if $x \in \mathbb{G}$ is a non-duplicatable node,

$$x = b \;\Leftrightarrow\; x \in \mathbb{G}_b \;\Leftrightarrow\; x \notin \mathbb{G}_{\bar{b}}.$$

If $x \in \mathbb{G}_b$ is a non-duplicatable node, x is equal to b.

When $\mathbb{G}$ has n vertices, there exists $2n$ assignment variables. Thus, the number of assignments i.e. that of splits is 2^{2n}. Some of such splits may be invalid for converting the scheme. The following algorithm efficiently decides whether an assignment is valid or not.

Algorithm 2 (Valid or Not).

Input. *An abstract crypto scheme* $(\mathbb{G}, \texttt{NoDup}, \texttt{Pair})$*, and an assignment* $\delta : V_{\mathbb{G}} \to \{0, 1\}$.

Output. 1 *if valid,* 0 *otherwise.*

Step 1. $\forall x \in V(\mathbb{G})$,

If $(x \in \mathbb{G}_0) \vee (x \in \mathbb{G}_1) \neq 1$ *then output* 0 *and halt* ;

2. $\forall (x \overset{\mathbb{G}}{\to} y) \in E(\mathbb{G})$, $\forall b \in \{0, 1\}$,

If $(y \in \mathbb{G}_b) \wedge \neg(x \in \mathbb{G}_b)$ *then output* 0 *and halt* ;

3. $\forall \{x, y\} \in \texttt{Pair}$, $\forall b \in \{0, 1\}$,

If $(x \in \mathbb{G}_b) \oplus (y \in \mathbb{G}_b) \neq 1$ *then output* 0 *and halt* ;

4. $\forall x \in \texttt{NoDup}$,

If $(x \in \mathbb{G}_0) \oplus (x \in \mathbb{G}_1) \neq 1$ *then output* 0 *and halt* ;

5. *Output 1 and halt.*

Our task which is referred to as pairing type conversion is just to satisfy and optimize this assignment. The following problems are some formalizations of this task.

Definition 18 (Pairing Type Satisfiability Problem).

Name. *Pairing Type Satisfiability Problem* (*abbr.* `PairSAT`)

Instance. *An abstract crypto scheme* $\mathbb{G}$.

Question. *Decide whether there exists a valid assignment in* $(V_{\mathbb{G}} \to \{0, 1\})$ *or not.*

Definition 19 (Search Version of `PairSAT`).

Name. *Search Version of* `PairSAT` (*abbr.* `SPairSAT`)

Instance. *An abstract crypto scheme* $\mathbb{G}$.

Question. *Find a valid assignment in* $(V_{\mathbb{G}} \to \{0, 1\})$ *if possible.*

Definition 20 (Pairing Type Optimization Problem).

Name. *Pairing Type Optimization Problem* (*abbr.* `PairOpt`)

Instance. *An abstract crypto scheme* $\mathbb{G}$*, and an evaluation function* $f : (V_{\mathbb{G}} \to \{0, 1\}) \to \mathbb{R}$.

Question. *Find a valid assignment in* $(V_{\mathbb{G}} \to \{0, 1\})$ *that minimizes* f.

The evaluation function f is assumed to be an index of some cryptographic interest like message length, circuit size, operation time, and so on, but f is not specified here for formal treatment.

Algorithms to solve `SPairSAT` or `PairOpt` can be diverted to `PairSAT` even if their output is nonsense for impossible cases, because we can decide efficiently whether a given split is valid or not. Therefore, w.r.t. hardness of the problems, we can easily derive

$$\texttt{PairSAT} \leq \texttt{SPairSAT} \leq \texttt{PairOpt}.$$

In the literature, an efficient algorithm to solve `SPairSAT` is known, but one to solve `PairOpt` is not. Just an efficient reduction from `PairOpt` to 0-1 integer programming problem is known [2]. It has been unknown whether there exists a separation between `SPairSAT` and `PairOpt` w.r.t. computational complexity.

4 Semi-Optimal Split

4.1 Valid Split

In [1, 35], Abe et al. introduced a set of conditions to guarantee functionalities and securities of the converted scheme as in the definition 14. But their conditions are not so informative for our problems because they don't exclude apparently inefficient splits. In this section, we introduce a new class of split which we call semi-optimal valid split and proof some trivial facts. Surprisingly most of relevant results in this work will be derived from such trivial facts.

To define some classes of splits or optimizations, we will treat conditions such in the definition 14 (Valid Split) axiomatically. First of all we write down the conditions in the definition 14 by using our notation. After that we will discuss their properties, and derive some lemmas.

A valid split $(\mathbb{G}_0, \mathbb{G}_1)$ of an abstract crypto scheme $(\mathbb{G}, \texttt{NoDup}, \texttt{Pair})$ satisfies the following 4 conditions.

Condition 1. $\forall x \in \mathbb{G},\ (x \in \mathbb{G}_0) \vee (x \in \mathbb{G}_1) = 1,$

Condition 2. $\forall x, y \in \mathbb{G} : x \overset{\mathbb{G}}{\rightsquigarrow} y,\ \forall b \in \{0,1\},\ (y \in \mathbb{G}_b) \Rightarrow (x \in \mathbb{G}_b),$

Condition 3. $\forall \{x, y\} \in \texttt{Pair},\ \forall b \in \{0,1\},\ (x \in \mathbb{G}_b) \oplus (y \in \mathbb{G}_b) = 1,$

Condition 4. $\forall x \in \texttt{NoDup},\ (x \in \mathbb{G}_0) \oplus (x \in \mathbb{G}_1) = 1.$

Proposition 1. *Any inputs of pairings are not duplicated in a valid split.*

Proof. Let $\{x, y\}$ be an input of a pairing. If x is duplicated in a split $(\mathbb{G}_0, \mathbb{G}_1)$, then $\forall b \in \{0,1\}$,

$$(y \in \mathbb{G}_b) = (x \in \mathbb{G}_b) \oplus 1 = 0$$

by condition 3. But it is contrary to condition 1. ■

Proposition 2. *Let $x \in \mathbb{G}$ be a leaf. If there exists a valid split s.t. x is duplicated, then there exists another valid split s.t. x is not duplicated.*

Proof. Let x be a duplicatable leaf node, and $(\mathbb{G}_0, \mathbb{G}_1)$ be a valid split where x is duplicated. By proposition 1, x is not an input of a pairing. No matter whether the assignment of $(x \in \mathbb{G}_b)$ is altered or not, condition 2 is satisfied, because x is a leaf. Similarly condition 3 and condition 4 are satisfied for any assignment of $(x \in \mathbb{G}_b)$, because x is nether pairing nor non-duplicatable. Therefore we can alter the assignment of $(x \in \mathbb{G}_b)$ as long as condition 1 is satisfied. ■

By proposition 1 and proposition 2, we can reduce the number of assignments to consider. Namely, to eliminate inconsiderable assignments, we will treat all leaves as non-duplicatable. To apply similar techniques to the upper nodes, we introduce some restatements of condition 2 and the concept of semi-optimal split.

Proposition 3. *Condition 2 $\Leftrightarrow \forall x, y \in \mathbb{G} : x \overset{\mathbb{G}}{\rightsquigarrow} y$, $\vec{y} \leq \vec{x}$, which means $\forall b \in \{0,1\},\ (y \in \mathbb{G}_b) \leq (x \in \mathbb{G}_b)$.*

Proof. By the following truth table, the proposition holds.

A	B	$A \Rightarrow B$	$A \leq B$
0	0	1	1
1	0	0	0
0	1	1	1
1	1	1	1

■

Before providing the most important (but trivial) proposition in this section, we give the following to explain its essence.

Proposition 4. *Let $(\mathbb{G}_0, \mathbb{G}_1)$ be a split satisfying condition 2. If $x \in \mathbb{G}$ has two descendants y_1 and y_2, $((y_1 \in \mathbb{G}_b) \vee (y_2 \in \mathbb{G}_b)) \leq (x \in \mathbb{G}_b)$ for all $b \in \{0,1\}$.*

Proof. By proposition 3,

$$((y_1 \in \mathbb{G}_b) \leq (x \in \mathbb{G}_b)) \wedge ((y_2 \in \mathbb{G}_b) \leq (x \in \mathbb{G}_b)).$$

This means

$$\max((y_1 \in \mathbb{G}_b), (y_2 \in \mathbb{G}_b)) \leq (x \in \mathbb{G}_b).$$

By the following truth table, the proposition holds.

A	B	$\max(A, B)$	$A \vee B$
0	0	0	0
1	0	1	1
0	1	1	1
1	1	1	1

■

Proposition 5. *Let D_x be all of descendants of x. Condition 2 $\Leftrightarrow \forall x \in \mathbb{G},\ \forall b \in \{0,1\},\ (x \in \mathbb{G}_b) \geq \bigvee_{y \in D_x} (y \in \mathbb{G}_b)$. Here we define $\bigvee_{y \in D_x} (y \in \mathbb{G}_b) := 0$ if $D_x = \varnothing$.*

Proof. If $D_x = \varnothing$, we define

$$\bigwedge_{y \in D_x} ((y \in \mathbb{G}_b) \leq (x \in \mathbb{G}_b)) := 1,$$

and

$$\max_{y \in D_x} ((y \in \mathbb{G}_b)) := 0.$$

$$\text{Condition 2} \Leftrightarrow \forall x \in \mathbb{G},\ \forall b \in \{0,1\},\ \bigwedge_{y \in D_x} ((y \in \mathbb{G}_b) \leq (x \in \mathbb{G}_b))$$

$$\Leftrightarrow \forall x \in \mathbb{G},\ \forall b \in \{0,1\},\ \max_{y \in D_x} ((y \in \mathbb{G}_b)) \leq (x \in \mathbb{G}_b).$$

By applying

$$\max_{A \in S}(A) = \bigvee_{A \in S} A,$$

where S is a finite set of binary variables, the proposition holds. ■

4.2 Semi-Optimal Split

Definition 21 (Semi-Optimal Split). *We define that an assignment for the variable $(x \in \mathbb{G}_b)$ is semi-optimal iff*

$$(x \in \mathbb{G}_b) = \begin{cases} \neg(x \in \mathbb{G}_{\bar{b}}) & (\text{if } D_x = \varnothing), \\ \bigvee_{y \in D_x} (y \in \mathbb{G}_b) & (\text{if } D_x \neq \varnothing). \end{cases}$$

A split is also called semi-optimal iff all assignments in the split are semi-optimal.

Proposition 6. *If all assignments for the variables $(x \in \mathbb{G}_b)$ and $(y \in \mathbb{G}_b) \in D_x$ are semi-optimal,*

$$(x \in \mathbb{G}_b) = \begin{cases} \neg(x \in \mathbb{G}_{\bar{b}}) & (\text{if } L_x = \varnothing), \\ \bigvee_{y \in L_x} (y \in \mathbb{G}_b) & (\text{if } L_x \neq \varnothing). \end{cases}$$

where L_x is all of descendant leaves of x.

Proof. Because the proposition is trivial in the case of $L_x = \varnothing$, we suppose $L_x \neq \varnothing$. Assume that we fix the assignment of leaves. Since $\mathbb{G}$ is a DAG, we can decide all semi-optimal assignments of upper nodes as

$$(x \in \mathbb{G}_b) = \bigvee_{y \in D_x} (y \in \mathbb{G}_b),$$

by going upstream repeatedly. This means the semi-optimal assignments of $(x \in \mathbb{G}_b)$ depends only on the leaves if all assignments of its descendants are semi-optimal. By using the commutative and idempotent property of $\vee$, i.e. the identities $A \vee B = B \vee A$ and $A \vee A = A$, the proposition holds. ■

Proposition 7. *If a split $(\mathbb{G}_0, \mathbb{G}_1)$ is valid and $(x \in \mathbb{G}_b)$ is not semi-optimal, x is duplicated in the split.*

Proof. If $D_x = \varnothing$ the proposition holds by definition and condition 1. Therefore we assume $D_x \neq \varnothing$ in the following. Because $(x \in \mathbb{G}_b)$ is not semi-optimal,

$$0 \leq \bigvee_{y \in L_x} (y \in \mathbb{G}_b) \lneq (x \in \mathbb{G}_b) \leq 1.$$

Therefore $(x \in \mathbb{G}_b) = 1$ and $\bigvee_{y \in L_x} (y \in \mathbb{G}_b) = 0$. This means $\forall y \in L_x, (y \in \mathbb{G}_b) = 0$. By condition 1, $\forall y \in L_x, (y \in \mathbb{G}_{\bar{b}}) = 1$.

$$1 = \bigvee_{y \in L_x} (y \in \mathbb{G}_{\bar{b}}) \leq (x \in \mathbb{G}_{\bar{b}}) \leq 1.$$

Therefore $(x \in \mathbb{G}_{\bar{b}}) = 1$, i.e. x is duplicated in the split $(\mathbb{G}_0, \mathbb{G}_1)$. ■

Proposition 8. *If there exists a valid split $(\mathbb{G}_0, \mathbb{G}_1)$ s.t. $(x \in \mathbb{G}_b)$ is not semi-optimal, then there exists another valid split s.t. $(x \in \mathbb{G}_b)$ is semi-optimal.*

Proof. By proposition 7, x is duplicated in the original split. This means x is nether a pairing nor a non-duplicatable node. Therefore both of condition 3 and 4 are satisfied regardless of the value of $(x \in \mathbb{G}_b)$. If we alter the assignment of $(x \in \mathbb{G}_b)$, condition 1 is also satisfied because x is originally duplicated i.e. $(x \in \mathbb{G}_{\bar{b}}) = 1$. Finally, condition 2 is true as long as

$$(x \in \mathbb{G}_b) \geq \bigvee_{y \in D_x} (y \in \mathbb{G}_b).$$

Therefore we can alter the assignment of $(x \in \mathbb{G}_b)$ to be semi-optimal. ■

By using the following `NonSemiToSemi` algorithm, we can convert from a non-semi-optimal assignment to a semi-optimal one without losing its validity.

Algorithm 3 (`NonSemiToSemi`).

Input. *An abstract crypto scheme* $(\mathbb{G}, \texttt{NoDup}, \texttt{Pair})$ *and a valid assignment* $\delta : V_{\mathbb{G}} \to \{0,1\}$.

Output. *A semi-optimal valid assignment* $\delta' : V_{\mathbb{G}} \to \{0,1\}$.

Step 1. $\forall v \in V(\mathbb{G})$, $\texttt{visited}[v] \leftarrow 0$;

Step 2. `foreach` $v \in V(\mathbb{G})$, $\texttt{DFSearch}(v)$;

Step 3. *Output* δ' *and halt.*

Subroutine. $\texttt{DFSearch}(v)$

Side effects. δ' *and* `visited` *will be updated.*

Steps. *If* $(\neg \texttt{visited}[v])$ *then*

$\texttt{visited}[v] \leftarrow 1$;

If $\{w \mid w \overset{\mathbb{G}}{\leftarrow} v\} = \varnothing$ *then*

If $\delta(v \in \mathbb{G}_0) \wedge \delta(v \in \mathbb{G}_1)$ *then*

$\delta'(v \in \mathbb{G}_0) \overset{\$}{\leftarrow} \{0,1\}$;

$\delta'(v \in \mathbb{G}_1) \leftarrow 1 \oplus \delta'(v \in \mathbb{G}_0)$;

else `foreach` $b \in \{0,1\}$,

$\delta'(v \in \mathbb{G}_b) \leftarrow \delta(v \in \mathbb{G}_b)$;

else

`foreach` $w : w \overset{\mathbb{G}}{\leftarrow} v$, $\texttt{DFSearch}(w)$;

`foreach` $b \in \{0,1\}$, $\delta'(v \in \mathbb{G}_0) \leftarrow \bigvee_{w \,:\, w \overset{\mathbb{G}}{\leftarrow} v} \delta'(w \in \mathbb{G}_0)$;

`return` ;

As a consequence of proposition 8, we can drastically reduce the number of assignments to consider w.r.t satisfiability problem. Namely, it is enough to consider just semi-optimal assignments to solve `PairSAT` or `SPairSAT`. To apply this technique to optimization problem, we introduce the following condition which is natural because the evaluation function is assumed to be something like size of data, cost of implementation or efficiency of operations.

Condition 5. *Let* $(\mathbb{G}_0, \mathbb{G}_1)$ *and* $(\mathbb{G}'_0, \mathbb{G}'_1)$ *be two splits of* $\mathbb{G}$, f *be the evaluation function of splits. If* $x \in \mathbb{G}$ *is duplicated in the split* $(\mathbb{G}'_0, \mathbb{G}'_1)$ *but not duplicated in the split* $(\mathbb{G}_0, \mathbb{G}_1)$, *and all other assignments are the same in the both split, then*

$$f(\mathbb{G}_0, \mathbb{G}_1) \leq f(\mathbb{G}'_0, \mathbb{G}'_1).$$

Proposition 9. *Let* $(\mathbb{G}_0, \mathbb{G}_1)$ *and* $(\mathbb{G}'_0, \mathbb{G}'_1)$ *be two valid splits of* $\mathbb{G}$, f *be the evaluation function of splits. If* $(x \in \mathbb{G}_b)$ *is semi-optimal but* $(x \in \mathbb{G}'_b)$ *is not, and all other assignments are the same in the both split, then*

$$f(\mathbb{G}_0, \mathbb{G}_1) \leq f(\mathbb{G}'_0, \mathbb{G}'_1).$$

Proof. By proposition 7 and condition 5, the proposition holds. ■

Proposition 10. *If a split is semi-optimal, it satisfies condition 1 and condition 2.*

Proof. By proposition 5, all vertices in a semi-optimal split satisfies condition 2. Therefore we concentrate condition 1. Let x be a vertex of $\mathbb{G}$. If $D_x = \varnothing$ then x satisfies condition 1 by definition. Therefore we assume $D_x \neq \varnothing$ in the following. If $(x \in \mathbb{G}_b) = 1$ then x satisfies condition 1 by definition. Therefore we assume $(x \in \mathbb{G}_b) = 0$ in the following. Because $(x \in \mathbb{G}_b)$ is semi-optimal,

$$\bigvee_{y \in L_x} (y \in \mathbb{G}_b) = (x \in \mathbb{G}_b) = 0.$$

This means $\forall y \in L_x, (y \in \mathbb{G}_b) = 0$. Because $\forall y \in L_x$ is semi-optimal, $\forall y \in L_x, (y \in \mathbb{G}_{\bar{b}}) = 1$. Therefore

$$(x \in \mathbb{G}_{\bar{b}}) = \bigvee_{y \in L_x} (y \in \mathbb{G}_{\bar{b}}) = 1.$$

Therefore $(x \in \mathbb{G}_{\bar{b}}) = 1$, i.e. x satisfies condition 1. ■

Based on the above arguments, it is enough to consider only semi-optimal valid splits to solve `PairSAT`, `SPairSAT`, or `PairOpt`. By proposition 10 we can replace condition 1 and condition 2 with the following condition.

Condition 6. *Split must be semi-optimal.*

Proposition 11. *Assume that all leaves are non-duplicatable. If a split satisfies condition 2 and condition 4,*

$$\forall x \in \mathbb{G},\ (x \text{ is non-duplicatable} \Rightarrow \forall y \in L_x,\ x \oplus y = 0).$$

in the split.

Proof. Because the proposition is trivial in the case of $L_x = \varnothing$, we suppose $L_x \neq \varnothing$. If there exists $y_0 \in L_x$ s.t. $x \oplus y_0 \neq 0$ then $(x \in \mathbb{G}_1) \oplus (y_0 \in \mathbb{G}_1) = 1$ by the definition of assignment variable of non-duplicatable node. Because y_0 is a leaf, $(y_0 \in \mathbb{G}_0) \oplus (y_0 \in \mathbb{G}_1) = 1$. Therefore $(y_0 \in \mathbb{G}_0) = (x \in \mathbb{G}_1)$. Because the split satisfies condition 2,

$$(x \in \mathbb{G}_0) \geq \bigvee_{y \in L_x} (y \in \mathbb{G}_0) \geq (y_0 \in \mathbb{G}_0).$$

Therefore $(x \in \mathbb{G}_0) \geq (x \in \mathbb{G}_1)$. Similarly, we can derive $(x \in \mathbb{G}_1) \geq (x \in \mathbb{G}_0)$ from $(y_0 \in \mathbb{G}_1) = (x \in \mathbb{G}_0)$ and

$$(x \in \mathbb{G}_1) \geq \bigvee_{y \in L_x} (y \in \mathbb{G}_1) \geq (y_0 \in \mathbb{G}_1).$$

Therefore

$$(x \in \mathbb{G}_0) = (x \in \mathbb{G}_1).$$

But this is contrary to non-duplicatability of x. ■

5 Hardness of the Problems

Theorem 1 (Abe et al.[2]). *There exists a polynomial time algorithm to solve* `SPairSAT`.

Proof. We show this constructively by giving the following algorithm which solve `SPairSAT` in time polynomial in the size of input. ■

Algorithm 4 (`SPairSAT` **Solver**).

Input. *An abstract crypto scheme* $(\mathbb{G}, \texttt{NoDup}, \texttt{Pair})$.

Output. *A split* $(\mathbb{G}_0, \mathbb{G}_1)$ *which satisfies condition 6, 3, and 4, if possible.* $\perp$ *otherwise.*

Step 1. $\texttt{NoDup} \overset{\cup}{\leftarrow} L_{\mathbb{G}}$;

2. *Let* $Q \leftarrow \varnothing$ *be a variable of an equation system* ;

3. $\forall x \in \texttt{NoDup}$, $\forall y \in L_x$, $Q \overset{\cup}{\leftarrow} \{x \oplus y = 0\}$;

4. $\forall \{x, y\} \in \texttt{Pair}$, $Q \overset{\cup}{\leftarrow} \{x \oplus y = 1\}$;

5. *Establish an echelon form of linear equation system* $Q/\mathbb{F}_2$ *with Gaussian elimination and decide its consistency ;*

6. *If* $Q/\mathbb{F}_2$ *is inconsistent, output* $\perp$ *and halt. Otherwise* $\forall x \in \texttt{NoDup}$ *s.t. x is independent in the above echelon form, assign x to any in* $\{0,1\}$, *and decide all dependent variables in* `NoDup` ;

7. $\forall x \in \texttt{NoDup}$, $(x \in \mathbb{G}_0) \leftarrow \neg x$;
// $(x \in \mathbb{G}_1) = x$ *by definition 17.*

8. $\forall x \in \mathbb{G} \backslash \texttt{NoDup},\ \forall b \in \{0,1\},\ (x \in \mathbb{G}_b) \leftarrow \bigvee_{y \in L_x} (y \in \mathbb{G}_b)$;
9. *Establish* $(\mathbb{G}_0, \mathbb{G}_1)$ *according to the assignment, output it, and halt.*

Corollary 1. `PairSAT` *is in* **P**.

Tango et al. gave another proof of this corollary using graph coloring [36].

In [2], Abe et al. introduce a reduction from `PairOpt` to `0-1 IP`. The following is a simplified version of it.

Algorithm 5 (`PairOpt` to `0-1 IP`).

Input. *An instance of* `PairOpt`*: an abstract crypto scheme* $(\mathbb{G}, \texttt{NoDup}, \texttt{Pair})$*, and an evaluation function* $f : (V(\mathbb{G}) \times \{0,1\} \to \{0,1\}) \to \mathbb{R}$.

Output. *An instance of* `0-1 IP`*: .*

Step 1. $\texttt{NoDup} \xleftarrow{\cup} L_{\mathbb{G}}$;
2. *Let* $Q \leftarrow \varnothing$ *be a variable of an inequality system* ;
3. $\forall x \in \texttt{NoDup},\ Q \xleftarrow{\cup} \{(x \in \mathbb{G}_0) + (x \in \mathbb{G}_1) = 1\}$;
4. $\forall \{x, y\} \in \texttt{Pair},\ Q \xleftarrow{\cup} \{(x \in \mathbb{G}_1) + (y \in \mathbb{G}_1) = 1\}$;
5. $\forall (x \xrightarrow{\mathbb{G}} y) \in E(\mathbb{G}),\ \forall b \in \{0,1\},\ Q \xleftarrow{\cup} \{(x \in \mathbb{G}_b) \geq (y \in \mathbb{G}_b)\}$;
6. *Output* Q *and* f*, and halt.*

In the above algorithm, the evaluation function f is not specified, but we assume that it is an index of some cryptographic interest. For example, if f is chosen to be the total bit size of some group elements,

$$f(\delta) = \sum_{x,b} |\mathbb{G}_b|\, \delta(x \in \mathbb{G}_b),$$

where $|\mathbb{G}_b|$ is that of $\mathbb{G}_b$. For another example, if f is the total operation cost of computing some group elements,

$$f(\delta) = \sum_{x,b} c\,|\mathbb{L}| \cdot |\mathbb{G}_b|^2\, \delta(x \in \mathbb{G}_b),$$

where $c\,|\mathbb{L}| \cdot |\mathbb{G}_b|^2$ is that of an exponentiation, assuming that x's are derived by it. For the usual interest of crypto design, the evaluation function f becomes a linear function of $(x \in \mathbb{G}_b)$'s, because $(x \in \mathbb{G}_b)$'s are binary variables.

In general, if f contains an expression $X \wedge Y$ for distinct binary variables X and Y, we can replace it with a new binary variable Z by introducing a new constraints

$$\{Z - X - Y + 1 \geq 0,\ X - Z \geq 0,\ Y - Z \geq 0\}$$

to the inequality system Q. Similarly, if f contains $\neg X$, we can replace it with a new variable Z by introducing

$$\{Z = 1 - X\}$$

to Q. Therefore any linear combination of any binary logic can be easily linearized by introducing new variables and constraints.

In any case, for a large class of computable evaluation function, we can establish a linear inequality system Q and a linear evaluation function f, i.e. an instance of 0-1 integer "linear" programming problem.

Let A be the coefficient matrix of Q, and $\vec{b}$ be constant terms of Q. It is known that, if Q has a special property called totally dual integral, there exists a polynomial-time algorithm to solve such a 0-1 integer linear programming problem. If A has a special form called totally unimodular, and all elements in $\vec{b}$ are integer, then Q is guaranteed to be totally dual integral. In fact, all elements in $\vec{b}$ are integer, and A has several characteristics that a totally unimodular matrix should have.

Therefore even if A is not totally unimodular, one may think that some techniques like Lagrangian relaxation may derive a deterministic polynomial-time algorithm to solve it. However, later results in this section will show that such a hope is almost gone for exact optimization of abstract crypto scheme without any restriction.

Definition 22 (`MINSAT` [26]).

Name. *Minimum Satisfiability Problem* (*abbr.* `MINSAT`)

Instance. *A set of binary variables* $U = \{u_1, \ldots, u_k\}$*, and a set of clauses* $C = \{c_1, \ldots, c_n\}$ *over* U *(where a clause is a disjunction of literals).*

Question. *Find a truth assignment :* $U \to \{0,1\}$ *to minimize number of clauses in* C *which is satisfied by the assignment.*

We just refer the following.

Theorem 2 (Kohli et al. [26]). `MINSAT` *is* **NP***-hard.*

Theorem 3. `PairOpt` *is* **NP***-hard.*

Proof. The following algorithm reduces a `MINSAT` instance to a `PairOpt` instance in time polynomial in the size of input.

Algorithm 6 (`MINSAT` to `PairOpt`).

Input. *An instance of* `MINSAT`*:* $U = \{u_1, \ldots, u_k\}$ *and* $C = \{c_1, \ldots, c_n\}$.

Output. *An instance of* `PairOpt`*: an abstract crypto scheme* $(\mathbb{G}, \texttt{NoDup}, \texttt{Pair})$*, and an evaluation function* $f : (V(\mathbb{G}) \times \{0,1\} \to \{0,1\}) \to \mathbb{R}$.

Step 1. $(V(\mathbb{G}), E(\mathbb{G}), \texttt{NoDup}, \texttt{Pair}) \leftarrow (\varnothing, \varnothing, \varnothing, \varnothing)$;
2. $\forall u \in U,$

$$V(\mathbb{G}) \xleftarrow{\cup} \{u, \neg u\},\ \texttt{NoDup} \xleftarrow{\cup} \{u, \neg u\},\ \texttt{Pair} \xleftarrow{\cup} \{\{u, \neg u\}\}$$;

3. $\forall c \in C,$

$$V(\mathbb{G}) \xleftarrow{\cup} \{c\}$$;

For all literal $\ell \in c$, $E(\mathbb{G}) \xleftarrow{\cup} \{(c \xrightarrow{\mathbb{G}} \ell)\}$;

4. *Let* $f(\delta) := \sum_{c \in C} \delta(c \in \mathbb{G}_1)$;
5. *Output* $(\mathbb{G}, \texttt{NoDup}, \texttt{Pair})$ *and* f*, and halt.*

Given an optimal valid split $(\mathbb{G}_0, \mathbb{G}_1)$ of the above `PairOpt` instance. We can derive a semi-optimal valid split efficiently from a given valid split, by altering assignments repeatedly as in the proof of proposition 8. Therefore we can find its semi-optimal version of $(\mathbb{G}_0, \mathbb{G}_1)$ efficiently. Moreover the semi-optimal version also satisfies the optimality of f, because f satisfies condition 5. In the semi-optimal split, $(c \in \mathbb{G}_1)$ satisfies

$$(c \in \mathbb{G}_1) = \bigvee_{\ell \in c} (\ell \in \mathbb{G}_1) = \bigvee_{\ell \in c} \ell.$$

Therefore the assignments of $(u_i \in \mathbb{G}_1) = u_i$ is the solution to minimize $\sum_{c \in C} \bigvee_{\ell \in c} \ell$, i.e. the solution of `MINSAT` instance (U, C). ■

References

[1] Masayuki Abe, Jens Groth, Miyako Ohkubo, and Takeya Tango. Converting cryptographic schemes from symmetric to asymmetric bilinear groups. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 241–260. Springer, 2014.

[2] Masayuki Abe, Fumitaka Hoshino, and Miyako Ohkubo. Design in Type-I, Run in Type-III: Fast and Scalable Bilinear-Type Conversion Using Integer Programming. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part III*, volume 9816 of *Lecture Notes in Computer Science*, pages 387–415. Springer, 2016.

[3] Alfred V. Aho, John E. Hopcroft, and Jeffrey Ullman. Kosaraju's algorithm. In *Data Structures and Algorithms*, pages 222–229. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1983. The authors credit the algorithm of Section 6.7 to an unpublished paper from 1978 by S. Rao Kosaraju.

[4] Joseph A. Akinyele, Christina Garman, and Susan Hohenberger. Automating Fast and Secure Translations from Type-I to Type-III Pairing Schemes. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*, pages 1370–1381. ACM, 2015.

[5] Joseph A. Akinyele, Matthew Green, and Susan Hohenberger. Using SMT solvers to automate design tasks for encryption and signature schemes. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 399–410. ACM, 2013.

[6] Razvan Barbulescu, Pierrick Gaudry, Antoine Joux, and Emmanuel Thomé. A quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic. *IACR Cryptology ePrint Archive*, 2013:400, 2013.

[7] Razvan Barbulescu, Pierrick Gaudry, Antoine Joux, and Emmanuel Thomé. A heuristic quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2014.

[8] Roderick Bloem, Harold N. Gabow, and Fabio Somenzi. An algorithm for strongly connected component analysis in n log n symbolic steps. *Formal Methods in System Design*, 28(1):37–56, 2006.

[9] Dan Boneh and Xavier Boyen. Secure identity based encryption without random oracles. In Matthew K. Franklin, editor, *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *Lecture Notes in Computer Science*, pages 443–459. Springer, 2004.

[10] Dan Boneh and Xavier Boyen. Short Signatures Without Random Oracles. In Christian Cachin and Jan Camenisch, editors, *Proc. of EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 56–73. Springer, 2004.

[11] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In Joe Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer, 2001.

[12] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In *Proc. of ASIACRYPT 2001*, pages 514–532, 2001.

[13] Dan Boneh and Hovav Shacham. Group signatures with verifier-local revocation. In Vijayalakshmi Atluri, Birgit Pfitzmann, and Patrick Drew McDaniel, editors, *Proceedings of the 11th ACM Conference on Computer and Communications Security, CCS 2004, Washington, DC, USA, October 25-29, 2004*, pages 168–177. ACM, 2004.

[14] Sanjit Chatterjee, Darrel Hankerson, Edward Knapp, and Alfred Menezes. Comparing two pairing-based aggregate signature schemes. *Des. Codes Cryptography*, 55(2-3):141–167, 2010.

[15] Sanjit Chatterjee and Alfred Menezes. On cryptographic protocols employing asymmetric pairings - the role of Ψ revisited. *IACR Cryptology ePrint Archive*, 2009:480, 2009.

[16] Sanjit Chatterjee and Alfred Menezes. On cryptographic protocols employing asymmetric pairings - the role of Ψ revisited. *Discrete Applied Mathematics*, 159(13):1311–1322, 2011.

[17] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.

[18] Pierre-Alain Fouque and Mehdi Tibouchi. Indifferentiable hashing to barreto-naehrig curves. In Alejandro Hevia and Gregory Neven, editors, *Progress in Cryptology - LATINCRYPT 2012 - 2nd International Conference on Cryptology and Information Security in Latin America, Santiago, Chile, October 7-10, 2012. Proceedings*, volume 7533 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2012.

[19] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.

[20] Faruk Göloglu, Robert Granger, Gary McGuire, and Jens Zumbrägel. On the function field sieve and the impact of higher splitting probabilities: Application to discrete logarithms in $\mathbb{F}_{2^{1971}}$. *IACR Cryptology ePrint Archive*, 2013:74, 2013.

[21] Thomas Icart. How to hash into elliptic curves. In Shai Halevi, editor, *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, volume 5677 of *Lecture Notes in Computer Science*, pages 303–316. Springer, 2009.

[22] Antoine Joux. A one round protocol for tripartite diffie-hellman. In Wieb Bosma, editor, *ANTS*, volume 1838 of *Lecture Notes in Computer Science*, pages 385–394. Springer, 2000.

[23] Antoine Joux. Faster index calculus for the medium prime case application to 1175-bit and 1425-bit finite fields. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, pages 177–193. Springer, 2013.

[24] Antoine Joux. A new index calculus algorithm with complexity $L(1/4 + o(1))$ in very small characteristic. *IACR Cryptology ePrint Archive*, 2013:95, 2013.

[25] Antoine Joux. Discrete logarithms in small characteristic finite fields: a survey of recent advances (invited talk). In Heribert Vollmer and Brigitte Vallée, editors, *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, volume 66 of *LIPIcs*, pages 3:1–3:1. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

[26] Rajeev Kohli, Ramesh Krishnamurti, and Prakash Mirchandani. The minimum satisfiability problem. *SIAM J. Discrete Math.*, 7(2):275–283, 1994.

[27] Tatsuaki Okamoto. On pairing-based cryptosystems. In Phong Q. Nguyen, editor, *Progressin Cryptology - VIETCRYPT 2006, First International Conferenceon Cryptology in Vietnam, Hanoi, Vietnam, September 25-28, 2006, Revised Selected Papers*, volume 4341 of *Lecture Notes in Computer Science*, pages 50–66. Springer, 2006.

[28] Paulo S. L. M. Barreto. The Pairing-Based Crypto Lounge. https://web.archive.org/web/20150408080445/www.larc.usp.br/~pbarreto/pblounge.html, 2002, 2008.

[29] Ryuichi Sakai, Kiyoshi Ohgishi, and Masao Kasahara. Cryptosystems based on pairing. In *Proc. of SCIS 2000 2000 Symposium on Cryptography and Information Security Okinawa, Japan, Jan. 26 - 29, 2000*. IEICE, 2000.

[30] Hisayoshi Sato and Keisuke Hakuta. An efficient method of generating rational points on elliptic curves. *Journal of Math-for-Industry*, 1:33–44, 2009.

[31] Andrew Shallue and Christiaan van de Woestijne. Construction of rational points on elliptic curves over finite fields. In Florian Hess, Sebastian Pauli, and Michael E. Pohst, editors, *Algorithmic Number Theory, 7th International Symposium, ANTS-VII, Berlin, Germany, July 23-28, 2006, Proceedings*, volume 4076 of *Lecture Notes in Computer Science*, pages 510–524. Springer, 2006.

[32] M. Sharir. A strong-connectivity algorithm and its applications in data flow analysis. *Computers & Mathematics with Applications*, 7(1):67 – 72, 1981.

[33] Mariusz Skałba. Points on elliptic curves over finite fields. *Acta Arithmetica*, 117:293–301, 2005.

[34] Nigel P. Smart and Frederik Vercauteren. On computable isomorphisms in efficient asymmetric pairing-based systems. *Discrete Applied Mathematics*, 155(4):538–547, 2007.

[35] Takeya Tango, Masayuki Abe, and Tatsuaki Okamoto. Implementation of Automated Translation for Schemes on Symmetric Bilinear Groups. In *Proc. of SCIS 2014 The 31st Symposium on Cryptography and Information Security kagoshima, Japan, Jan. 21 - 24, 2014*. IEICE, 2014.

[36] Takeya Tango, Masayuki Abe, Tatsuaki Okamoto, and Miyako Ohkubo. Polynomial-Time Algorithm for Deciding Possibility of Converting Cryptographic Schemes from Type-I to III Pairing Groups. In *Proc. of SCIS 2015 The 32nd Symposium on Cryptography and Information Security Kokura, Japan, Jan. 20 - 23, 2015*. IEICE, written in japanese, 2015.

[37] Robert Endre Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972.

[38] Mehdi Tibouchi. A Note on Hashing to BN Curves. In *Proc. of SCIS 2012 The 29th Symposium on Cryptography and Information Security Kanazawa, Japan, Jan. 30 - Feb. 2, 2012*. IEICE, 2012.

[39] Eric R. Verheul. Self-blindable Credential Certificates from the Weil Pairing. In Colin Boyd, editor, *Proc. of ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 533–551. Springer, 2001.

[40] Brent Waters. Efficient identity-based encryption without random oracles. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*, pages 114–127. Springer, 2005.