PAPER

Fast and Scalable Bilinear-Type Conversion Method for Large Scale Crypto Schemes*

Masayuki ABE[†,††], ***Senior Member***, **Fumitaka HOSHINO**[†,†††a)], ***Nonmember***, ***and*** **Miyako OHKUBO**[††††], ***Member***

SUMMARY Bilinear-type conversion is to translate a cryptographic scheme designed over symmetric bilinear groups into one that works over asymmetric bilinear groups with small overhead regarding the size of objects concerned in the target scheme. In this paper, we address scalability for converting complex cryptographic schemes. Our contribution is threefold. Investigating complexity of bilinear-type conversion. We show that there exists no polynomial-time algorithm for worst-case inputs under standard complexity assumption. It means that bilinear-type conversion in general is an inherently difficult problem. Presenting a new scalable conversion method. Nevertheless, we show that large-scale conversion is indeed possible in practice when the target schemes are built from smaller building blocks with some structure. We present a novel conversion method, called *IPConv*, that uses 0-1 Integer Programming instantiated with a widely available IP solver. It instantly converts schemes containing more than a thousand of variables and hundreds of pairings. Application to computer-aided design. Our conversion method is also useful in modular design of middle to large scale cryptographic applications; first construct over simpler symmetric bilinear groups and run over efficient asymmetric groups. Thus one can avoid complication of manually allocating variables over asymmetric bilinear groups. We demonstrate its usefulness by somewhat counter-intuitive examples where converted DLIN-based Groth-Sahai proofs are more compact than manually built SXDH-based proofs. Though the early purpose of bilinear-type conversion is to save existing schemes from attacks against symmetric bilinear groups, our new scalable conversion method will find more applications beyond the original goal. Indeed, the above computer-aided design can be seen as a step toward automated modular design of cryptographic schemes.

key words: *pairing-based cryptography, bilinear-type conversion, integer programming, cryptographic scheme design, Groth-Sahai proofs*

1. Introduction

1.1 Background

Bilinear groups (also called pairing groups) are mathematical primitives that yield wide variety of advanced cryptographic applications. Informally, a bilinear group is defined by a triple of groups $(\mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_T)$ of same order q associated with an efficient bihomomorphism $e : \mathbb{G}_0 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$ called pairing. Among several types of bilinear groups in the literature, most frequently used ones in cryptography are those so-called Type-I and Type-III groups [2]. In Type-I groups, there exist non-degenerate efficient homomorphisms between $\mathbb{G}_0$ and $\mathbb{G}_1$ bidirectionally and hence it is regarded as $\mathbb{G}_0 = \mathbb{G}_1$ that is simply denoted by $\mathbb{G}$. In Type-III groups, it is assumed that there exists no non-degenerate efficient homomorphism between $\mathbb{G}_0$ and $\mathbb{G}_1$. Type-I groups has been a popular choice in cryptographic design in early days. For these days, however, crypto designers are prompted to employ Type-III groups due to the rapid progress in cryptanalysis for small characteristic finite fields that match to Type-I groups [3–8].

As vast number of schemes have been built over Type-I groups, e.g, [9–15], bilinear-type conversion methods that translate schemes designed for Type-I groups into ones that work over Type-III groups have been developed [16–22]. Recall that cryptographic schemes designed over Type-I groups do not necessarily work over Type-III groups due to the presence of symmetric pairings, $e(X, X)$. A workaround is to convert the algorithm by *duplicating* the variables. That is, the variable is represented by a pair $(X, X') \in \mathbb{G}_0 \times \mathbb{G}_1$. Duplication however clearly slows down the performance since all relevant computations are 'duplicated' in $\mathbb{G}_0$ and $\mathbb{G}_1$ as well. Besides, duplication is not always possible due to mathematical constraints or external requirements. For instance, it is not known how to pick random and consistent pair X and X' while retaining the hardness of the discrete logarithm problem on X and X'. An automated conversion finds the best allocation of variables over $\mathbb{G}_0$ and $\mathbb{G}_1$ that makes all group operations doable with minimal overhead.

Automated conversion methods in the literature [20–22] are only for small-scale schemes consisting of up to tens of group operations. To follow the secure conversion framework introduced in [21], one needs to convert not only the algorithms in the target scheme but those that appear in security proofs. It makes the target of conversion much larger than the scheme itself. Besides, applications often use several cryptographic schemes as building blocks. It is in particular common to use efficient non-interactive proof systems in constructions of advanced applications, which results in involving hundreds of variables and dozens of pairings. Existing methods thus fell short for converting middle to large-scale schemes.

Manuscript received March 12, 2018.
Manuscript revised August 17, 2018.

[†] The authors are with Secure Platform Laboratories, NTT Corporation, Musashino-shi, 180-8585 Japan.

[††] The author is with Graduate School of Informatics, Kyoto University, Kyoto-shi, 606-8501 Japan.

[†††] The author is with School of Computing, Tokyo Institute of Technology, Tokyo, 152-8550 Japan.

[††††] The author is with Security Fundamentals Laboratory, CSR, NICT, Koganei-shi, 184-8795 Japan.

[*] A preliminary version of this paper was published in the proceedings of CRYPTO 2016 [1]. This work was supported by JSPS KAKENHI Grant Number JP16K00027.

a) E-mail: hoshino.fumitaka@lab.ntt.co.jp

DOI: 10.1587/transfun.E102.A.251

1.2 Our Contribution

In this paper, we address the scalability issue in bilinear-type conversion. We investigate the complexity of the problem in general, show a practical solution, and present its application in cryptographic design. We elaborate our contributions as follows.

(1) Investigating complexity of bilinear-type conversion.

Though bilinear-type conversion has been studied for long and several heuristic methods have been explored, no theoretical argument has been given about its complexity. We for the first time investigate the complexity of bilinear-conversion and formally prove the difficulty of the problem. In the framework of [21] a scheme over Type-I group is represented by a graph called a dependency graph. It describes a flow of group operations in $\mathbb{G}$ over variables in the scheme. Conversion is explained as a problem of splitting a given dependency graph into two dependency graphs that represent flow of group operations in $\mathbb{G}_0$ and $\mathbb{G}_1$. These graphs must satisfy certain conditions so that the resulting scheme is executable. A split that satisfies the conditions is called a valid split. It has been shown that whether a valid split exists or not can be decided in polynomial-time [23]. Finding a valid split that brings the best efficiency in the converted scheme is an optimization problem. We show that there is no algorithm that solves the optimization problem in the worst case in time polynomial in the size of input, if $\mathbf{P} \neq \mathbf{NP}$. Therefore the scalable conversion is an essentially difficult problem in general.

(2) Presenting a new scalable conversion method.

The above negative result however does not mean absence of practical solution. The optimal solution may be found in practical time for realistic cryptographic schemes that have some structure in their dependency graph. We present a novel scalable conversion method, which we call 'IPConv', that uses 0-1 integer programming (IP). Given several kinds of constraints and a metric implemented into an objective function, it searches for a solution that minimizes the function value subject to the constraints. The idea of encoding computational constraints into an objective function follows from previous works [20, 22]. Our novelty is the encoding method that allows one to use Integer Programming that fits well to our optimization problem with various constraints. Besides, using such a tool is advantageous in the sense that there are publicly available (both commercial and non-commercial) software packages such as [24–29].

We demonstrate its scalability by applying it to large systems with thousands of variables and pairings are generated randomly subject to some reasonably looking structures. IPConv processed them in a few minutes to hours even with non-commercial IP solver SCIP [25] as an engine. The concrete figures of course become magnitude of better with a powerful commercial IP solver e.g. [24]. Scaling up to thousands of pairings may seem an overkill. However, for instance, schemes that include Groth-Sahai (GS) proof system [30] easily involve dozens or even hundreds of pairings when their security proofs are taken into account. Furthermore, tools such as [31–33] would allow automated synthesis that reach to or even exceed such a scale. Our method not only contributes to speedup the process of conversion but also opens the door to automated synthesis and optimization of large scale cryptographic applications over bilinear groups.

(3) Application to computer-aided design.

To show usefulness of IPConv beyond its original purpose of saving existing Type-I schemes, we use IPConv for conversion-assisted design of middle-scale schemes that involve GS-proofs. GS-proofs typically requires the Decision Linear assumption (DLIN) over Type-I groups or the Symmetric eXternal Diffie-Hellman assumption (SXDH) over Type-III groups. By conversion, the DLIN assumption is translated to the eXternal DLIN (XDLIN) assumption [34]. Though it is generally considered that schemes based on SXDH is more efficient than those based on XDLIN, we show some examples that converted schemes using XDLIN is more efficient than their direct instantiation based on SXDH.

Concretely, our first example is a scheme for showing in zero-knowledge ones possession of a correct structure-preserving signature [35] on a public message. We measure the concrete size of proofs when instantiated over KSS-16 curves at 128-bit security parameter and show that the scheme obtained by conversion yields proofs that are up to 56% shorter (asymptotic in the message length) than those generated by direct constructions based on SXDH. The construction uses a novel fine-tuning for zero-knowledge GS-proofs which may be of independent interest. Our second example is an automorphic blind signature scheme [35] that involves GS-proofs and is secure under SXDH assumption in asymmetric pairing. We show that the proofs can be replaced with the DLIN-based ones and it can be converted to work in asymmetric pairing under XDLIN assumption saving 41% of the signature size compared to the originally manufactured SXDH-based scheme.

Although our primary metric for optimization is the size of intended objects, we also compare their computational workload in the number of pairings in signature verification. Interestingly, the winner changes depending on the message size, acceptable duplication, and also the use of batch verification technique [36]. This unveils an open issue on optimization of schemes involving GS-proofs.

Documentation and source files used for the experiments in this paper are available in [37].

1.3 Related Works

Early works on bilinear-type conversion, e.g. [16–19], 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 [20] 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 [38]. 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 [22], Akinyele, Garman, and Hohenberger introduced an upgraded system called AutoGroup+ that integrates the framework of [21] 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 [21], Abe et al. established a theoretical ground for preserving security during conversion. Their framework, reviewed in Sect. 3, 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 [1], the authors proved that there exists a polynomial time algorithm to solve search version of pairing type satisfiability problem, and introduced an algorithm using 0-1 Integer Programming (IP) to solve the optimization problem. This is a preliminary version of this paper.

Regarding Groth-Sahai zero-knowledge proofs, the closest work is the one by Escala and Groth in [39]. They observe that commitment of $1_{\mathbb{Z}_p}$ can be seen as a commitment of the default generator G and uses the fact that a commitment of G can be equivocated to $1_{\mathbb{G}}$ to construct more efficient zero-knowledge proofs for pairing product equations (PPEs) with constant pairings of the form $e(G, A)$ in Type-III setting. Our fine-tuning technique uses the same property for the commitment of G but use it in a different manner that is most effective in Type-I setting. For details please refer to Section 5.1 in [40]. Another close work is [41] that presents a DLIN-based variant of GS-proof system over asymmetric bilinear groups. Their scheme bases on SDLIN assumption where *independent* DLIN in $\mathbb{G}_0$ and $\mathbb{G}_1$ are assumed as hard, and uses independently generated CRSes for commitments in $\mathbb{G}_0$ and $\mathbb{G}_1$. Thus their proof system is inherently asymmetric, which cannot exploit nice properties of symmetric setting as done in this work. Besides, SDLIN-based instantiation is less efficient than SXDH-based one. We therefore use the original SXDH-based instantiation for comparison in this paper. In [42, 43], a more efficient instantiation of GS-proofs by using recently introduced Matrix assumptions. Although DLIN-based GS-proofs are used throughout this paper, matrix-based assumption might be an alternative to further gain efficiency if the Type-III analogue of the assumption is acceptable.

2. Preliminary

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 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 $\bar{x}$.*

We will frequently use some elementary notions in graph theory. Since the same words often mean slightly different notions between different contexts in graph theory, to avoid confusion or ambiguity, in this paper we define them as follows.

Definition 2 (Directed Graph). *A directed graph G is defined as $G = (V, E)$, where V is a set called vertex set, and E is a 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 neither 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 $V(G)$ and $E(G)$ respectively. An edge $(x, y) \in V \times V$ is often written as $(x \overset{G}{\rightarrow} y)$ or $(y \overset{G}{\leftarrow} x)$ to clear its direction.*

Definition 3 (In-edge/Out-edge). *For a vertex x, $(x \overset{G}{\rightarrow} y)$ (resp. $(x \overset{G}{\leftarrow} y)$) is called its outgoing edge or out-edge (resp. incoming edge or in-edge).*

Definition 4 (Degree). *For any vertices $x \in V(G)$, its outdegree $\deg^+(x)$, indegree $\deg^-(x)$ and degree $\deg(x)$ are defined as $\deg^+(x) := \#\{(x \overset{G}{\rightarrow} y) \in E(G) \mid y \in G\}$, $\deg^-(x) := \#\{(x \overset{G}{\leftarrow} y) \in E(G) \mid y \in G\}$, and $\deg(x) := \deg^+(x) + \deg^-(x)$.*

Definition 5 (Leaf). *A leaf is defined as a vertex without outgoing edge, i.e. vertex x s.t. $\deg^+(x) = 0$.*

Definition 6 (Parent/Child). *Let x and y be vertices in $V(G)$. When $(x \overset{G}{\rightarrow} y) \in E(G)$, we simply denote $x \overset{G}{\rightarrow} y$, i.e. the binary relation $x \overset{G}{\rightarrow} y$ is a shorthand for "$(x \overset{G}{\rightarrow} y) \in E(G)$". In the same manner, when $(x \overset{G}{\leftarrow} y) \in E(G)$, we denote $x \overset{G}{\leftarrow} y$. If $x \overset{G}{\rightarrow} y$ (resp. $x \overset{G}{\leftarrow} y$), x is called a parent (resp. child) of y.*

Although it is ambiguous whether the expression $(x \overset{G}{\rightarrow} y)$ means an edge (x, y) or a binary relation $x \overset{G}{\rightarrow} y$ in parenthesis, in most cases we can distinguish between them by context. In this paper, we rarely use the latter.

Definition 7 (Ancestor/Descendant). *A vertex y which can reach (resp. reachable from) x is called an ancestor (resp. a descendant) of x. If y is an ancestor (resp. a descendant) of x or x itself, we write $y \overset{G}{\rightsquigarrow} x$ (resp. $y \overset{G}{\leftsquigarrow} x$).*

Definition 8 (Strongly Connected). *For any vertices $x, y \in V(G)$, if $x \overset{G}{\rightsquigarrow} y$ and $x \overset{G}{\leftsquigarrow} y$, we say x and y are strongly connected to each other, and write $x \overset{G}{\sim} y$.*

Definition 9 (Induced Subgraph). *Let G be a graph, S be a subset of $V(G)$. The (vertex) induced subgraph $G[S]$ is defined as $G[S] := (S, \{(x, y) \in E(G) \mid x, y \in S\})$.*

Definition 10 (Strongly Connected Component/Cycle). *For a graph G, its strongly connected components are defined as the induced subgraphs of equivalence classes of $V(G)$ w.r.t. $\overset{G}{\sim}$. When a strongly connected component has 2 or more vertices, we call it a cycle.*

3. Conversion Based on Dependency Graphs

3.1 Overview

In this section we review the framework in [21]. To guarantee the security of the resulting scheme, it converts not only algorithms that form the target scheme but also all algorithms that appear in the security proof as well as underlying assumptions. Namely, it assumes that the security is proven by the existence of reduction algorithms from some assumptions in Type-I, and converts the algorithms and assumptions into Type-III. This way, the security proof is preserved under the converted assumption. It is proven in [21] that if the original assumptions are valid in Type-I generic bilinear group model [44], the converted assumptions are valid in Type-III generic bilinear group model. Most typically, the DLIN assumption is converted to XDLIN.

In their framework relations among variables in target algorithms are described by using a graph called a dependency graph, and the central task of conversion is reduced to find a 'split' of the graph so that each graph implies variables and computations in each source group in the Type-III setting.

We follow the framework of [21] that consists of the following four steps.

1. Verify that the target scheme in Type-I and its security proof follows the abstraction of bilinear groups.
2. Describe the generic bilinear group operations over source group $\mathbb{G}$ by using a dependency graph as we shall explain later.
3. Split the dependency graph into two that satisfy some conditions. The resulting graphs imply variables and group operations in $\mathbb{G}_0$ and $\mathbb{G}_1$ respectively.
4. Describe the resulting scheme in Type-III as suggested by the graphs.

As well as [21], we focus on step 3 and propose a practical algorithm for the task of finding a split. Thus, when we conduct an experiment for demonstrating the performance, we start from a dependency graph as input and complete when a desirable split of the input graph is obtained.

3.2 Dependency Graph

A dependency graph is a directed graph that represents computational dependencies among variables storing source group elements in the target system. Each vertex represents a variable storing a group element in the algorithms e.g. some part or all of public key, cipher text, signature, commitment, zero-knowledge proof and so on. Each edge represents its dependency w.r.t. group operations, that is head depends on tail.

```
Sample($a, A, B, D$)
  $a \in \mathbb{Z}/p\mathbb{Z}$
  $A, B, C, D, E \in \mathbb{G}$
  if $a = 0$ then
    $C \leftarrow A \cdot B,\ E \leftarrow D$
  else
    $C \leftarrow D^a,\ E \leftarrow D^3$
  return $e(C, E)$
```

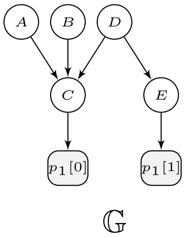

Fig. 1 An example of dependency graph.

In Fig. 1, we show an example of a dependency graph for a program that computes some group operations over Type-I pairing. The left of Fig. 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 Fig. 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-else", variables storing other than the source group elements like $a \in \mathbb{Z}/p\mathbb{Z}$, and operations in the target group. In a dependency graph, there may exist some nodes which do not appear explicitly in the description of algorithms. Such nodes are called implicit. The followings are the typical cases to appear implicit nodes.

- **Temporary nodes** represent temporary variables in $\mathbb{G}$. In a description of an algorithm, results of group operations are not necessary assigned to explicit variables. To bind the type of all operands, a new node is introduced.
- **Pairing nodes** represent inputs to pairing operations. Every pairing node has only one incoming edge and no outgoing edges. Each pairing node is paired with another pairing node so that the pair constitutes an input to a pairing operation.
- **Comparing nodes** represent element identification operations in $\mathbb{G}$ i.e. $=$ or $\neq$. In terms of dependency, element identification is nothing other than the group operation except for its output. To bind the type of both side, a new node who has two incoming edges but no outgoing one is introduced.
- **Control nodes** may be appended to the dependency graph to implement users' preferences defined independently of the description of algorithms. In Sect. 5.2, we will exemplify three typical cases, i.e. specific assignment, grouping, and exclusive assignment.

In Abe et al.'s framework, two sub-graphs are derived from a dependency graph to execute the conversion. 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

```
Sample($a, A, B, (D_0, D_1)$)
  $a \in \mathbb{Z}/p\mathbb{Z}$
  $A, B, C, D_0 \in \mathbb{G}_0;\ D_1, E \in \mathbb{G}_1$
  if $a = 0$ then
    $C \leftarrow A \cdot B,\ E \leftarrow D_1$
  else
    $C \leftarrow D_0^a,\ E \leftarrow D_1^3$
  return $e(C, E)$
```

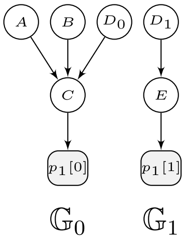

Fig. 2 An example of conversion by split.

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 Fig. 2, we show an example of conversion for the program of Fig. 1.

Definition 11 **($\mathbb{G}, \mathbb{G}_0, \mathbb{G}_1$).** *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 split or its vertex set as in the Fig. 1. In the same manner, we redefine $\mathbb{G}_0$ and $\mathbb{G}_1$ as sub-graphs of $\mathbb{G}$ or their vertex sets as in the Fig. 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 $P = O$ which is the identity element of $\mathbb{G}_0$ and $\mathbb{G}_1$, but this is a fallacy because the statement means nothing other than that the variable P is represented as $\mathbb{G}_0 \times \mathbb{G}_1$ in the converted scheme.

Definition 12 (Split). *Formally, for a dependency graph $\mathbb{G}$, any pairs of its sub-graphs are called its splits.*

Definition 13 (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 14 (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.*

It is known that vertices which represents the outputs of an operation called HashToPoint [15, 19, 45, 46] must be configured as non-duplicatable [21, 47]. Some nodes may be specified as non-duplicatable by crypto designers due to reducing the data size of converted scheme.

Definition 15 (Pairing node/Regular node). *A vertex which represents an input of pairing is called a pairing node. Vertices other than pairing nodes are called regular nodes.*

Every pairing node has just one in-edge and no out-edges, hence it must be a leaf. Each pairing node is paired with another pairing node so that the pair constitutes an input to a pairing operation. We assume that variables storing the inputs of a pairing are declared and defined implicitly. For example, in the right of Fig. 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 $p_i[1]$. Moreover, we will 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 the dependency graph has 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 can reduce a dependency graph to a DAG by identifying all vertices in each cycle 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.

The above reduced 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 [48–50].

For most of our problem, it is enough to consider the reduced dependency graph instead of the original one. Therefore we assume $\mathbb{G}$ as a DAG hereinafter, and define the following notion to treat bilinear-type conversion formally.

Definition 16 (Abstract Crypto Scheme). *A directed acyclic graph $\mathbb{G}$ with*

$$\begin{aligned} &\texttt{NoDup} \subset V(\mathbb{G}) : \textit{ set of non-duplicatable nodes, and} \\ &\texttt{Pair} \subset \{\{x, y\} \mid x, y \in L^1_{\mathbb{G}}\} : \textit{ set of pairings} \end{aligned}$$

is called an abstract symmetric-pairing-based crypto scheme, or just an abstract crypto scheme. Here $L^1_{\mathbb{G}}$ is all of leaves in $\mathbb{G}$ whose indegree is just 1, and $\# \bigcup \texttt{Pair} = 2\#\texttt{Pair}$.

The word "abstract crypto scheme" means the same as (reduced) dependency graph except for ignoring whether it is derived from a real crypto scheme or not. To avoid complicated notation, we will often omit `NoDup` and `Pair`, and simply write $\mathbb{G}$ to express an abstract crypto scheme, except when they are necessary.

3.3 Valid Split

In [21, 51], Abe et al. defined a class of split called valid split which guarantees the functionalities and the security of the converted scheme. It has been shown in [21] that if a dependency graph is split into two graphs that satisfy four conditions below then the converted scheme derived from the graphs works over Type-III bilinear groups and is secure in the same sense as the original scheme but based on converted assumptions. Such a pair of graphs is called a valid split.

Definition 17 (Valid Split). *Let* $(\mathbb{G}, \texttt{NoDup}, \texttt{Pair})$ *be an abstract crypto 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. *if* $y \overset{\mathbb{G}}{\rightsquigarrow} x \wedge x \in \mathbb{G}_b$ *then* $y \in \mathbb{G}_b$*, for all* $b \in \{0, 1\}$*,*
3. *if* $\{x, y\} \in \texttt{Pair}$ *then* x *and* y *are separately included in* $\mathbb{G}_0$ *and* $\mathbb{G}_1$*, and*
4. $\mathbb{G}_0 \cap \mathbb{G}_1 \cap \texttt{NoDup} = \varnothing$.

The first condition guarantees that all variables and computations are preserved during conversion. The second condition guarantees that all variables needed to compute a variable belong to the same source group.

Let $(\mathbb{G}_0, \mathbb{G}_1)$ be a split which satisfies $V(\mathbb{G}_0) \cup V(\mathbb{G}_1) = V(\mathbb{G})$ and the above property 2. For any edge $(y \overset{\mathbb{G}}{\rightarrow} x)$ in $E(\mathbb{G})$, there exist $b \in \{0, 1\}$ s.t. $x \in V(\mathbb{G}_b)$ because $x \in V(\mathbb{G}_0) \cup V(\mathbb{G}_1)$. Therefore $y \in V(\mathbb{G}_b)$ by the property 2, i.e. for any edge $e \in E(\mathbb{G})$ there exist $b \in \{0, 1\}$ s.t. $e \in E(\mathbb{G}_b)$ if we can identify the induced subgraph $\mathbb{G}[V(\mathbb{G}_b)]$ and $\mathbb{G}_b$.

This means we do not 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 restore $\mathbb{G}_b$ to the induced subgraph $\mathbb{G}[V(\mathbb{G}_b)]$ when $E(\mathbb{G})$ is known.

The following algorithm efficiently decides whether a split is valid or not.

Algorithm 1 (Valid or Not).

Input. *An abstract crypto scheme* $(\mathbb{G}, \texttt{NoDup}, \texttt{Pair})$*, and a split* $(\mathbb{G}_0, \mathbb{G}_1)$.
Output. 1 *if valid,* 0 *otherwise.*
Steps.

```
1. $\forall x \in \mathbb{G}$ if $(x \in \mathbb{G}_0) \vee (x \in \mathbb{G}_1) \neq 1, E(\mathbb{G}),\ \forall b \in \{0, 1\}$
       if $(y \in \mathbb{G}_b) \wedge \neg(x \in \mathbb{G}_b)$ return 0,
3. $\forall \{x, y\} \in \texttt{Pair},\ \forall b \in \{0, 1\}$
       if $(x \in \mathbb{G}_b) \oplus (y \in \mathbb{G}_b) \neq 1$ return 0,
4. $\forall x \in \texttt{NoDup}$, if $(x \in \mathbb{G}_0) \oplus (x \in \mathbb{G}_1) \neq 1$ return 0,
5. return 1.
```

According to this algorithm, Abe et al. proposed a bilinear-type conversion algorithm [21,51], but it is basically a brute-force search over all possible conversions and requires exponential time in the number of nodes.

Note that a valid split as defined above only meets the mathematical constraint over the pairings and those given by NoDup. There could be large number of valid splits for a dependency graph and it is another issue how to pick the optimal one according the metric and constraints given by the user.

4. Theory of Bilinear-Type Conversion

In this section, we introduce a comprehensive theory of bilinear-type conversion. In this theory, 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 notations.

Definition 18 (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. For an abstract crypto scheme* $\mathbb{G}$*, we define* $V_{\mathbb{G}}$ *as the set of assignment variables, i.e.*

$$V_{\mathbb{G}} := \{(x \in \mathbb{G}_b) \mid (x, b) \in V(\mathbb{G}) \times \{0, 1\}\} = V(\mathbb{G}) \times \{0, 1\}.$$

Definition 19 (Shorthand Notation for Assignment Variables). *If vertex* x *is a non-duplicatable node, we regard* x *as a variable over* $\{0, 1\}$*, and define* $x := (x \in \mathbb{G}_1)$.

Definition 20 (Assignment). *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 split $(\mathbb{G}_0, \mathbb{G}_1)$ satisfies $(x \in \mathbb{G}_b) \Rightarrow (y \in \mathbb{G}_b)$ if $y \overset{\mathbb{G}}{\rightsquigarrow} x$," instead of "A split δ satisfies $\delta(x \in \mathbb{G}_b) \Rightarrow \delta(y \in \mathbb{G}_b)$ if $y \overset{\mathbb{G}}{\rightsquigarrow} x$."

Definition 21 (Order of Assignments). *Let* δ *and* δ' *be two assignments. We write* $\delta' \geq \delta$ *iff* $\forall (x \in \mathbb{G}_b) \in V_{\mathbb{G}}, \delta(x \in \mathbb{G}_b) \Rightarrow \delta'(x \in \mathbb{G}_b)$.

4.1 Problems on Bilinear-Type Conversion

A theoretical background that distinguish this work from previous ones is a separation between satisfiability and optimization problems on bilinear-type conversion. In the previous works, satisfiability problem is primarily focused on, and optimization is regarded as a subsidiary issue. However, in this work, we will show that the problem to tackle is just the optimization, since the satisfiability is easy. In this section, we formalize these problems to discuss the hardness of them.

Definition 22 (ConvSAT)**.**

Name. *Satisfiability of Bilinear-Type Conversion.*
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 23 (sConvSAT)**.**

Name. *Search Version of* ConvSAT.
Instance. *An abstract crypto scheme* $\mathbb{G}$.

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

Definition 24 (ConvOpt)**.**

Name. *Optimization of Bilinear-Type Conversion.*

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 a metric 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 sConvSAT or ConvOpt can be diverted to ConvSAT even if their output is nonsense for impossible cases, because by using Algorithm 1, we can decide whether a given split is valid or not efficiently. Therefore, w.r.t. hardness of the problems, we can easily derive ConvSAT $\leq$ sConvSAT $\leq$ ConvOpt. In the literature, an efficient algorithm for ConvSAT is known [23], while one for sConvSAT is unknown. In this work, we show that there exists polynomial-time algorithms to solve sConvSAT, but no algorithm to solve ConvOpt in the worst case in time polynomial in the size of input, if $\mathbf{P} \neq \mathbf{NP}$.

4.2 Semi-Optimal Split

In [21,51], Abe et al. introduced a set of conditions to guarantee all functionalities and securities of the converted scheme as in the Definition 17 (Valid Split). But their conditions are not so informative for our problems because they do not exclude apparently poorly-optimized splits. In this section, we introduce a special class of valid split which we call semi-optimal split and state some trivial facts. Surprisingly most of relevant results in this work will be derived from such trivial facts. We omitted some proofs of propositions due to space. See [47] for them.

Definition 25 (Semi-Optimal Split). *We define that an assignment for the variable* $(x \in \mathbb{G}_b)$ *is semi-optimal iff*

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

where D_x *is all of descendants of* x*. A node* x *is called semi-optimal iff both* $(x \in \mathbb{G}_b)$*'s are semi-optimal. A valid split is also called semi-optimal iff all assignments in the split are semi-optimal. Semi-optimal split is short for semi-optimal valid split.*

Proposition 1. *If there exists a valid split s.t.* $(x \in \mathbb{G}_b)$ *is not semi-optimal for some* $x \in \mathbb{G}$ *and some* $b \in \{0,1\}$*, there exists another valid split s.t.* $(x \in \mathbb{G}_b)$ *is semi-optimal.*

Corollary 1. *For an abstract crypto scheme, there exists a valid split iff there exists a semi-optimal one.*

By using the following algorithm, we can convert a valid assignment into a semi-optimal one efficiently.

Algorithm 2 (Valid To Semi-Optimal).

```
Input. An abstract crypto scheme $(\mathbb{G}, \mathtt{NoDup}, \mathtt{Pair})$ and
    a valid assignment $\delta : V_{\mathbb{G}} \to \{0,1\}$.
Output. A semi-optimal assignment $\delta' : V_{\mathbb{G}} \to \{0,1\}$.
Steps.
 1. $\forall v \in V(\mathbb{G})$, $\mathtt{visited}[v] \leftarrow 0$,
 2. $\forall v \in V(\mathbb{G})$, $\mathtt{DFSearch}(v)$,
 3. return $\delta'$.

Subroutine. $\mathtt{DFSearch}(v)$
SideEffects. $\delta'$ and $\mathtt{visited}$ will be updated.
Steps.
 if $(\neg \mathtt{visited}[v])$ then
   $\mathtt{visited}[v] \leftarrow 1$,
   if $\{w \mid w \overset{\mathbb{G}}{\leftarrow} v\} = \emptyset$ 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 $\forall b \in \{0,1\}$,
       $\delta'(v \in \mathbb{G}_b) \leftarrow \delta(v \in \mathbb{G}_b)$,
   else
     $\forall w : w \overset{\mathbb{G}}{\leftarrow} v$, $\mathtt{DFSearch}(w)$,
     $\forall b \in \{0,1\}$, $\delta'(v \in \mathbb{G}_0) \leftarrow \bigvee_{w : w \overset{\mathbb{G}}{\leftarrow} v} \delta'(w \in \mathbb{G}_0)$,
 return.
```

As a consequence of Corollary 1, we can drastically reduce the number of assignments to consider w.r.t the satisfiability problems. Namely, it is enough to consider just semi-optimal assignments to solve ConvSAT or sConvSAT. To apply this technique to the optimization problem, we introduce the following condition which is quite natural since the evaluation function is assumed to be something like size of data, cost of implementation or efficiency of operations.

Condition 1. *Let* f *be the evaluation function,* δ *and* δ' *be two valid splits, where all vertices but* x *have the same assignments in the both splits. If* x *is not duplicated in* δ *but in* δ'*, then* $f(\delta) \leq f(\delta')$*.*

Proposition 2. *Let* f *be the evaluation function which satisfies Condition 1,* δ, δ' *and* x *be the same as in Condition 1. If* x *is not semi-optimal in* δ' *but in* δ*, then* $f(\delta) \leq f(\delta')$*.*

Corollary 2. *Let* f *be the evaluation function which satisfies Condition 1. For any valid split* δ_1 *of* $\mathbb{G}$*, there exists a descending chain of valid splits terminated by a semi-optimal split* δ_n*, i.e.* $\delta_1 \geq \cdots \geq \delta_n$*, which satisfies* $f(\delta_1) \geq \cdots \geq f(\delta_n)$*.*

4.3 Hardness of the Problems

Theorem 1. *There exists a polynomial time algorithm to solve* sConvSAT.

***Proof*.** We show this constructively by giving the following algorithm which solve sConvSAT deterministically in time polynomial in the size of input. In the followings, we call a set of linear equations a linear equation system.

Algorithm 3 (sConvSAT **Solver).**

Input. *An abstract crypto scheme* $(\mathbb{G}, \mathtt{NoDup}, \mathtt{Pair})$.
Output. *A valid split* $(\mathbb{G}_0, \mathbb{G}_1)$ *if possible.* $\perp$ *otherwise.*
Steps.

```
1. $\mathtt{NoDup} \overset{\cup}{\leftarrow} L_{\mathbb{G}}$, where $L_{\mathbb{G}}$ is all of leaves in $\mathbb{G}$.
2. Let $Q \leftarrow \emptyset$ be a variable storing a linear equation system,
3. $\forall x \in \mathtt{NoDup},\ \forall y \in L_x,\ Q \overset{\cup}{\leftarrow} \{x \oplus y = 0\}$,
                      where $L_x$ is all of descendant leaves of $x$.
4. $\forall \{x, y\} \in \mathtt{Pair},\ Q \overset{\cup}{\leftarrow} \{x \oplus y = 1\}$,
5. Establish an echelon form of linear equation system
   $Q/\mathbb{F}_2$ with Gaussian elimination.
6. If the last non-zero row of the echelon form is $\vec{0} \cdot \vec{x} = 1$,
   where $\vec{x}$ is the vector of the variables in $Q/\mathbb{F}_2$, return $\perp$
   (which means $Q/\mathbb{F}_2$ is inconsistent).
7. Otherwise decide the assignment of the leading variable
   (the first variable who has non-zero coefficient from the
   left, also called the dependent variable) of the row to
   satisfy the equation of the row by assigning all following
   variables to any in $\{0, 1\}$. In the same way, decide the
   dependent variables in the upper rows from bottom to
   top by assigning all following variables consistently.
8. $\forall x \in \mathtt{NoDup}, (x \in \mathbb{G}_0) \leftarrow \neg x$,
          // $(x \in \mathbb{G}_1) = x$ by Definition 19.
9. $\forall x \in \mathbb{G} \setminus \mathtt{NoDup},\ \forall b \in \{0, 1\},\ (x \in \mathbb{G}_b) \leftarrow \bigvee_{y \in L_x}$
   $(y \in \mathbb{G}_b)$,
10. Establish $(\mathbb{G}_0, \mathbb{G}_1)$ according to the assignment, and
    return it.
```

□

By using Algorithm 3, we can exactly know how many feasible solutions i.e. semi-optimal splits exist, namely 2^n where n is the number of independent variables.

Algorithm 4 (Sanity Checking). *Sanity checking is a variant of Algorithm 3 which just returns the consistency of the linear equation system* $Q/\mathbb{F}_2$ *at Step 6.*

Corollary 3. ConvSAT *is in* **P**.

Tango et al. gave another proof of this corollary using graph coloring [23].

To prove the hardness of ConvOpt, we just refer the following definition and theorem due to Kohli, Krishnamurti and Mirchandani. See [52] for proof of the theorem.

Definition 26 (MinSAT **[52]).**

Name. *Minimum Satisfiability Problem.*

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.*

Theorem 2 (Kohli et al. [52]). MinSAT *is* **NP***-hard.*

Theorem 3. ConvOpt *is* **NP***-hard.*

***Proof*.** The following algorithm reduces a MinSAT instance to a ConvOpt instance in time polynomial in the size of input.

Algorithm 5 (MinSAT **to** ConvOpt**).**

Input. *An instance of* MinSAT*:*
$U = \{u_1, \ldots, u_k\}$ *and* $C = \{c_1, \ldots, c_n\}$.
Output. *An instance of* ConvOpt*:*
an abstract crypto scheme $(\mathbb{G}, \mathtt{NoDup}, \mathtt{Pair})$, *and*
an evaluation function $f : (V_{\mathbb{G}} \to \{0, 1\}) \to \mathbb{R}$.
Steps.

```
1. $(V(\mathbb{G}), E(\mathbb{G}), \mathtt{NoDup}, \mathtt{Pair}) \leftarrow (\emptyset, \emptyset, \emptyset, \emptyset)$,
2. $\forall u \in U$,
      define new symbol $p$,
      $V(\mathbb{G}) \overset{\cup}{\leftarrow} \{u, \neg u, p, \neg p\},\ E(\mathbb{G}) \overset{\cup}{\leftarrow} \{(u \xrightarrow{\mathbb{G}} p), (\neg u \xrightarrow{\mathbb{G}} \neg p)\}$,
      $\mathtt{NoDup} \overset{\cup}{\leftarrow} \{u, \neg u, p, \neg p\},\ \mathtt{Pair} \overset{\cup}{\leftarrow} \{\{p, \neg p\}\}$,
3. $\forall c \in C$,
      $V(\mathbb{G}) \overset{\cup}{\leftarrow} \{c\}$,
      For all literal $\ell \in c,\ E(\mathbb{G}) \overset{\cup}{\leftarrow} \{(c \xrightarrow{\mathbb{G}} \ell)\}$,
4. Let $f(\delta) := \sum_{c \in C} \delta(c \in \mathbb{G}_1)$,
5. return $(\mathbb{G}, \mathtt{NoDup}, \mathtt{Pair})$ and $f$.
```

Given an optimal valid split $(\mathbb{G}_0, \mathbb{G}_1)$ of the above ConvOpt instance. We can derive a semi-optimal split efficiently from a given valid split by using Algorithm 2. Thus 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 1. 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). □

5. Finding Optimal Valid Split with IP

In previous section, we show that there exists no algorithm to solve ConvOpt in the worst case in time polynomial in the size of input, if **P** ≠ **NP**.

However this negative result never means that there exist no practical bilinear-type conversion algorithm. The optimal solution may be found in practical time for practical cases, if it includes no hard structure. In the preliminary version of this paper [1], we propose such a conversion algorithm, which we call 'IPConv', based on 0-1 integer programming (IP). In this section, we will introduce a simplified version of it.

Proposition 3. *Let* $(\mathbb{G}_0, \mathbb{G}_1)$ *be a valid split of* $\mathbb{G}$.

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

Namely the map : $V(\mathbb{G}) \to \{0, 1\},\ x \mapsto \delta(x \in \mathbb{G}_b)$ can be regarded as order-preserving for a valid split δ. Regarding this property, we can extend Algorithm 3 to the next one, which reduces an instance of ConvOpt to that of 0-1 IP efficiently. In the followings, we call a set of linear equations and inequalities a linear inequality system.

Algorithm 6 (ConvOpt **to 0-1 IP).**

```
Input. An abstract crypto scheme $(\mathbb{G}, \mathtt{NoDup}, \mathtt{Pair})$.
Output. A linear inequality system $Q$.
Steps.
  1. $\mathtt{NoDup} \overset{\cup}{\leftarrow} L_{\mathbb{G}}$
  2. Let $Q \leftarrow \emptyset$ be a variable storing a linear inequality
     system,
  3. $\forall x \in \mathtt{NoDup}, Q \overset{\cup}{\leftarrow} \{(x \in \mathbb{G}_0) + (x \in \mathbb{G}_1) = 1\}$,
  4. $\forall \{x, y\} \in \mathtt{Pair}, Q \overset{\cup}{\leftarrow} \{(x \in \mathbb{G}_1) + (y \in \mathbb{G}_1) = 1\}$,
  5. $\forall (x \xrightarrow{\mathbb{G}} y) \in E(\mathbb{G}), \forall b \in \{0, 1\}, Q \overset{\cup}{\leftarrow} \{(x \in \mathbb{G}_b) \geq$
     $(y \in \mathbb{G}_b)\}$,
  6. return $Q$.
```

In the above algorithm, the evaluation function f is not specified, but we assume that it is a metric of some cryptographic interest.

In general, if f contains an expression $X \wedge Y$ (or $X \times Y$) for distinct binary variables X and Y, we can replace it with a new binary variable Z by introducing a new linear 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 and efficiently linearized by introducing new variables and linear constraints. Consequently, 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.

However, for simplicity, we assume that all evaluation functions are monotonic (order-preserving) and non-negative in the rest of this paper, i.e. for any assignments δ and δ', $\delta' \geq \delta \Rightarrow f(\delta') \geq f(\delta)$ and $f(\delta) \geq 0$. Such evaluation functions are natural for most of cryptographic interest like message length, circuit size, operation time, and so on. Hence, ignoring the validity of the splits, we can easily estimate the maximum and the minimum values of the evaluation functions as $\max f = f(\delta_1)$ and $\min f = f(\delta_0) \geq 0$, where δ_1 and δ_0 are the assignments which assign all assignment variables to 1 and 0 respectively.

5.1 IPConv Procedure

We present a new method, which we call 'IPConv' for finding an optimal valid split. IPConv takes the task in the third step of the conversion procedure mentioned in Sect. 3.1. It takes as input a dependency graph $\mathbb{G}$ of a Type-I scheme and users' preferences, and outputs a split $(\mathbb{G}_0, \mathbb{G}_1)$ corresponding to a converted Type-III scheme. IPConv consists of the following stages.

1. **Preprocessing on the graph.** The input dependency graph is modified to implement some user-specified preferences. When the dependency graph has a cycle, it is reduced into a DAG by using a strongly connected components decomposition algorithm [48–50]. The output of this stage is an abstract crypto scheme $(\mathbb{G}, \mathtt{NoDup}, \mathtt{Pair})$.
2. **Translating into a linear inequality system.** Assignment variable $(x \in \mathbb{G}_b)$'s are placed on each node x's in $V(\mathbb{G})$ for all $b \in \{0, 1\}$. Although the feasibility of the instance can be detected in later stages, we use Algorithm 4 in this stage to assure the existence of a solution, since it is overwhelmingly faster (deterministic polynomial-time). After that, the abstract crypto scheme is translated into a linear inequality system Q by using Algorithm 6.
3. **Establishing the objective function.** According to user's preferences, the objective function f is composed with possibly modifying Q and introducing new variables. We will discuss this in more detail in the next section.
4. **Running Integer Programming.** Run 0-1 Integer Programming for finding an assignment to the variables that minimizes the objective function f subject to the constraints Q.
5. **Composing the final split.** The assignment decides which constraint nodes belong to which source group, and further decides on other nodes. Thus a valid split is composed from the assignment.

5.2 Users' Preferences

One may want to avoid duplication regarding specific set of variables as much as possible. Typical practical demands would be to look for the minimal duplication in the public-key elements, or the smallest possible duplication in the instance of assumptions. In general, by manipulating inputs and outputs of Algorithm 6 and the objective function, i.e. $\mathbb{G}$, $\mathtt{NoDup}$, $\mathtt{Pair}$, Q and f, we can handle various requirements to the split. In this section we show in the following several types of preferences that can be handled in our conversion procedure.

- **Priority.** We allow users to give a priority to some nodes so that they avoid duplication as much as possible than other nodes. Concretely, a priority is given by a list of sets of nodes. Let $(I_1, I_2, \cdots)$ be a sequence of non-empty sets of nodes where every set consists of arbitrary number of nodes and the sets are pairwise disjoint. It is considered that nodes in I_i are given more priority for non-duplication than those in I_{i+1}. For instance, suppose that I_1 includes nodes representing a public-key and I_2 includes nodes representing a signature. By specifying (I_1, I_2) as a priority, a solution that includes less duplication in a public-key is preferred. If only one node in a public-key is duplicated in solution A, and all nodes in a signature are duplicated in solution B, then solution B will be taken. Unspecified nodes are given the least priority. For example, we can implement a evaluation function supporting priorities, based on the space to store, as follows. Let $g_i(\delta)$ be a evaluation function s.t. $g_i(\delta) := \sum_{x\in I_i,\ b\in\{0,1\}} |\mathbb{G}_b|(x \in \mathbb{G}_b)$, which means the space to store all nodes in I_i. We can compose a sequence of evaluation functions s.t.

$$f_i(\delta) = \begin{cases} 0 & (i = 0), \\ g_i(\delta) + (\#I_i + 1) f_{i-1}(\delta) & (i > 0), \end{cases} \tag{1}$$

which means in the evaluation function f_i, any one of nodes in f_{i-1} has a greater impact than all of nodes in g_i. When I_n is the least priority, clearly f_n implements all priorities. Evaluation functions supporting priorities based on other metrics can be also implemented by applying the same technique, e.g. $g_i(\delta) := \sum_{x\in I_i} (x \in \mathbb{G}_0) \wedge (x \in \mathbb{G}_1)$, which means the number of duplicated nodes in I_i.

- **Magnification factor.** Often a node represents multiple of variables treated in the same manner in the converting program. For instance, a message m consisting of several group elements $m = (m[1], \ldots, m[k])$ with constant k can be represented by a node referred to by m[i]. Such a node should have a magnification factor of k. It must be equal or larger than one. Magnification factor can be implemented simply by multiplying k to the term of the node m[i], e.g. $f(\delta) := k \times \sum_{b\in\{0,1\}} |\mathbb{G}_b|(\mathtt{m[i]} \in \mathbb{G}_b) + \cdots$. When a dependency graph is reduced into a DAG, a vertex in the DAG may represent multiple vertices in the original graph. In such a case, the representing vertex may have a magnification factor which can be calculated automatically. When a node with a magnification factor k belongs to a priority I_i, it may be natural to modify the meaning of $\#I_i$ in Eq. (1) to count up k group elements in the node.
- **Prohibiting duplication.** By specifying a node as 'prohibited', the node will never be duplicated. We can implement this simply by appending the node to `NoDup`.
- **Specific assignment.** By specifying a particular group to a particular node, the group is assigned to the node. (But the node may still be duplicated unless it is specified as 'prohibited' as well.) A specific assignment to a specific node, say n, is handled by appending a new implicit non-duplicatable node c and a new edge $(n \overset{\mathbb{G}}{\to} c)$ to the graph $\mathbb{G}$. To assign a user specified group $\mathbb{G}_b$ to c, just introduce a constraint $\{c = b\}$ to the inequality system Q, otherwise either $\mathbb{G}_0$ or $\mathbb{G}_1$ will be chosen automatically. As the specific group is assigned to c, the same group must be assigned to n as well since n is an ancestor of c.
- **Grouping.** By specifying a set of nodes, they are assigned to the same group. (But it does not solely mean no duplication for individual node.) Grouping of nodes $n_1, \ldots, n_k$ is handled in the same manner as in the case of specific assignment, i.e. by appending a new implicit non-duplicatable node c and edges $(n_1 \overset{\mathbb{G}}{\to} c), \ldots, (n_k \overset{\mathbb{G}}{\to} c)$ to the graph $\mathbb{G}$.
- **Exclusive assignment.** By specifying two nodes, different groups are assigned to each node. The specified nodes are implicitly specified as prohibited so that the exclusive assignment holds. This option, together with the prohibition, allows one to describe schemes designed in Type-III without concretely specifying groups to every variable. Exclusive assignment of two nodes x and y can be implemented by appending the nodes x and y to `NoDup`, and introducing a constraint $\{x + y = 1\}$ to the inequality system Q.

5.3 Optimality of the Output

According to our implementation of the objective function, IPConv outputs a solution whose variables given the top priority have minimal space to store. That is, those variables avoid duplication and are allocated in $\mathbb{G}_0$ as much as possible. Then, subject to the allocation in the top priority, variables in the second priority are allocated to have minimal space to store, and so forth. Concrete meaning of optimality is defined by the variables specified in the order of priority. If one's target is a public-key encryption scheme, for instance, and elements in a public-key are set as the top priority, the outcome is a scheme whose public-key has the shortest representation possible. (But it never reduces the number of group elements in the public-key, which is left for the designers' work.) To see the balance between several options in the order of priority, one may repeat the conversion to the same scheme with different preferences. Each result of conversion is optimal with respect to the given preference.

In the context of bilinear-type conversion, optimizing the size of objects is a reasonable choice for better efficiency as avoiding duplication not only saves the space but also saves relevant computation. Yet extending the objective function to implement more elaborate metrics is a potential direction for further research. For instance, it is desirable to incorporate the cost of computation each variable is involved in. It requires the dependency graph to carry more information than the relations by group operations. We leave it for future development.

6. Performance

Throughout the paper, experiments are done on a standard PC: CPU: Intel Core i5-3570 3.40GHz, OS: Linux 3.16.0-34-generic #47-Ubuntu. For Integer Programming, we use SCIP [25] (non-commercial) and GUROBI [24] (commercial). In this experiments, we assume $|\mathbb{G}_1| = 2|\mathbb{G}_0|$ according to Barreto-Naehrig curves [53].

6.1 Processing Time for Real Schemes

(1) Small-scale schemes.

In the first two rows of Table 1, we show the processing time of IPConv for converting Boneh-Boyen HIBE [54] with $\ell = 9$ hierarchy, and Waters' Dual-system encryption [13]. Their dependency graphs are relatively small but have number of possible splits. A comparison to AutoGroup+ is done in the same environment. For fair comparison, we need to offset the overhead for processing high-level I/O format in

Table 1 Processing time of IPConv with SCIP. Figures in parenthesis are those of AutoGroup+ in the same environment. The upper half is small-scale monolithic schemes and the lower half is middle-scale schemes consisting of several building blocks. (# vertices) counts all nodes including the pairing nodes in the input graph. (# pairings) counts pairs of pairing nodes.

Target	Graph Size		Processing	Notes
Scheme	#vertices	#pairings	Time	
Waters' DSE [13]	95	13	146 ms	(4639 ms)
BBS HIBE [54]	283	56	262 ms	(15667 ms)
BlindAutoSIG [35]	339	116	142 ms	-
AHO [35]+GSZK [30]	597	222	463 ms	-
Trace. Group Enc. [55]	1604	588	6306 ms	-

AutoGroup+. According to [22], it takes about 500ms to handle the smallest case in their experiments. Even after offsetting similar amount as an overhead, the speedup with IPConv is obvious.

(2) Middle-scale schemes.

We also conduct experiments on middle scale schemes that involve GS-proofs and other building blocks. The results are summarized in Table 1.

AHO Signature + GSZK: Our first experiment is for a structure-preserving signature scheme in [35], a.k.a. AHO signature scheme, combined with zero-knowledge proof of a correct signature on a public message. We set the message length for AHO signatures to $n = 4$ and instantiate the zero-knowledge proof with the DLIN-based GS-proofs and convert the entire scheme to Type-III. More details appear in Sect. 7.

Blind Automorphic Signature Scheme: The second experiment is for the automorphic blind signature scheme from [35]. This experiment is to demonstrate that our framework can handle schemes that is already in Type-III. Overall structure of the target scheme is the same as the first one; a combination of a signature scheme and a NIWI GS-proof of a correct signature. Unlike the first one, however, the scheme is constructed under SXDH assumption that holds only in the Type-III setting. We describe a dependency graph for the scheme using exclusive assignment directive so that SXDH assumption is consistently incorporated to the framework. It may be interesting to see that assumptions are the only part that need to set constraints originated from the asymmetry of groups. Constraints in all upper layer algorithms are automatically taken from the assumptions. More details appear in Section 5.3 in [40].

Traceable Group Encryption: Our last experiment is for a traceable group encryption scheme from [55] that is more intricate involving several building blocks such as a tag-based encryption [56], AHO signatures, and one-time signatures, and GS-proofs. Taking reduction algorithms in the security proofs of each building block, the corresponding dependency graph becomes as large as consisting of 1604 nodes including 588 × 2 pairing nodes, which is beyond the scale that existing automated conversion can process within a practical time.

6.2 Scalability

Though the experiment in the previous section already demonstrates the scalability of IPConv to some extent, we would like to see overall behavior of IPConv against the size of inputs. Generally it is exponential due to the nature of IP. Yet it is worth to know the threshold for the practical use.

(1) On Random Graphs.

To measure the performance and the tolerance in the scale, it is necessary to sample dependency graphs from reasonable and scalable distribution. However, it is indeed impossible to consider the distribution over all constructable cryptographic schemes. It does not make sense to consider it over all possible graphs, either, since most of them do not correspond to meaningful cryptographic schemes. We therefore use some heuristics to define the distribution. Through the experiments in the previous section, we have observed that dependency graphs for real cryptographic schemes follow some structure. We simulate it in a scalable manner in the following way: Let N be the number of regular nodes, P be the number of pairings, and k be the maximum fan-in to a regular node. Every regular node is indexed by $i \in \{1, \dots, N\}$. Pairing nodes $p_{ij}[0]$ and $p_{ij}[1]$ represent a pairing with nodes i and j as input.

Algorithm 7 (Random Dependency Graph Generation).

```
Input. Graph parameters:
  the number of regular nodes $N$,
  the number of pairings $P$, and
  the maximum fan-in to a regular node $k$.
Output. A dependency graph $\mathbb{G}$.
Steps.
  1. Generate regular nodes: $V(\mathbb{G}) \leftarrow \{1, \dots, N\}$, $E(\mathbb{G}) \leftarrow \emptyset$.
  2. For every regular node $i \in \{1, \dots, N\}$,
       select $k' \xleftarrow{\$} \{1, \dots, k\}$, and
       repeat the following $k'$ times:
         Select $j \xleftarrow{\$} \{1, \dots, i-1\}$.
         Generate an edge: $E(\mathbb{G}) \xleftarrow{\cup} \{(j \xrightarrow{\mathbb{G}} i)\}$.
  3. Repeat the following $P$ times:
       Randomly select two regular nodes $i$ and $j (\geq i)$
       (discard and redo if the pair has been chosen before).
       Generate pairing nodes: $V(\mathbb{G}) \xleftarrow{\cup} \{p_{ij}[0], p_{ij}[1]\}$,
       and edges: $E(\mathbb{G}) \xleftarrow{\cup} \{(i \xrightarrow{\mathbb{G}} p_{ij}[0]), (j \xrightarrow{\mathbb{G}} p_{ij}[1])\}$.
```

Our preliminary experiment shows that large k results in so dense graphs that do not well simulate the graphs for real schemes in the previous section. Throughout our experiments, we set $k = 6$ and $N = P$ as they are close to the average for those in the real examples. With such a heuristic parameter setting we are not able to claim theoretical rigorousness to the result of our experiments. But they do show some tendency in the scalability. For the purpose of comparison,

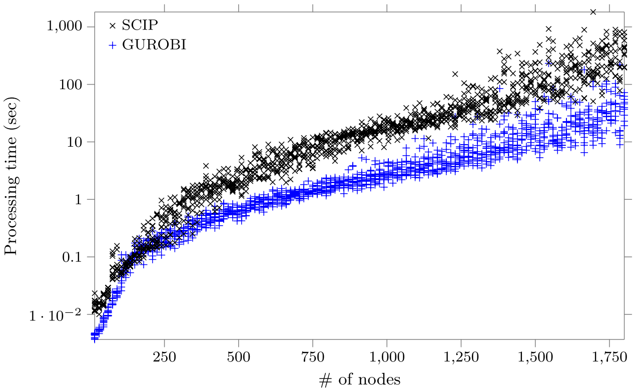

Fig. 3 Processing time in the semi-log scale for random dependency graphs.

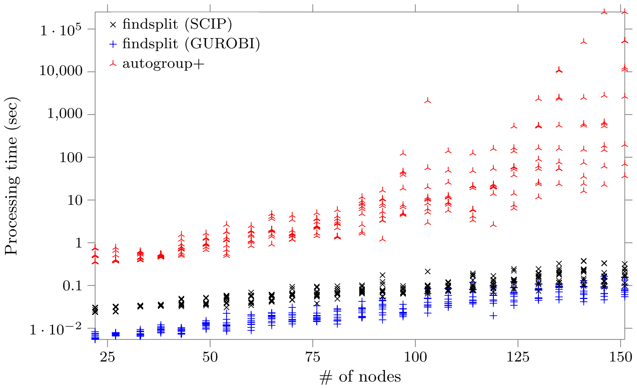

Fig. 4 Comparison between IPConv and AutoGroup+ regarding stability of processing time.

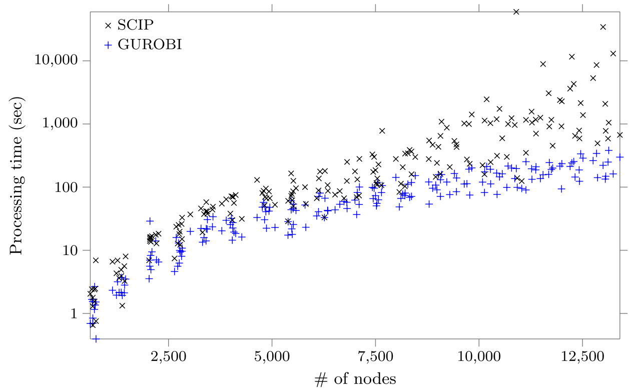

Fig. 5 Processing time in the semi-log scale for cluster dependency graphs.

we show a real dependency graph for a tagged one-time signature scheme [11] in Fig. A· 1 and a random dependency graph that has the same number of pairings in Fig. A· 2. The square-shaped nodes placed in the bottom of the graph are the pairing nodes. Other nodes are represented by a circle. The node at the top represents the default generators.

We first examine the permissible scale of IPConv by measuring its processing time for random dependency graphs having up to 600 pairings and equal number of regular nodes. Fig. 3 illustrates the results for 1200 inputs. IPConv finds an optimal solution in well affordable time up to around $N = P = 600$. But after that point, the processing time gets more dispersed depending on the input.

We next compare the performance with AutoGroup+. The result is illustrated in Fig. 4 that includes 250 samples for each AutoGroup+ and IPConv.

Around 150 nodes, the SMT solver used in AutoGroup+ rarely fails for some unidentified reason. With graphs containing 150 nodes, the processing time between two conversion methods differ 100 to 10^6 times. This result shows that middle to large scale conversion is out of the scope of AutoGroup+. Comparing the absolute processing time based on Fig. 4 is not perfectly fair as IPConv only takes the task of finding an optimal split whereas AutoGroup+ deals with higher-level inputs and outputs. But from the figure, one can see less dispersion in the processing time with IPConv, and its scalability is well observed.

(2) On Cluster Graphs.

We next evaluate the performance for more structured dependency graphs based on a prospect that large scale systems over bilinear groups are built in a modular fashion by combining several building blocks and GS-proofs. How would dependency graphs for such systems look like? Observe that, 1) only a small number of objects will be passed from one building block to others, 2) every building block would be used only through the legitimate interface during security proofs, and 3) the default generator is connected to a number of nodes in each building blocks. We thus foresee that a dependency graph for a modularly-built large-scale system would form sparsely connected clusters of dependency graphs with a single node that has relatively dense connection to nodes in every cluster.

We generate random cluster dependency graphs in a way that each cluster has similar volume and structure as that of AHO signature plus GS zero-knowledge proof appeared in the previous experiment (see Fig. A· 3 for its dependency graph). Namely, a cluster consists of a randomly connected thirty six regular nodes and some of the nodes are involved in two random PPEs for GS zero-knowledge proofs whose dependency is automatically encoded to the graph. Then every two clusters are randomly connected each other with a fixed number of edges. The resulting graph with five clusters is shown in Fig. A· 4. The performance of IPConv for the random cluster graphs are measured up to $n = 19$ clusters.

The experiment is repeated 10 times for each n. At $n = 19$, a graph consists of 13046 nodes and 5182 pairings in average. Comparing Fig. 5 with Fig. 3, there is a clear stretch in the handleable number of vertices. If there are no connections between the clusters (except for those from the node representing the default generator), the processing time will be linear in the number of the clusters assuming that the processing time for each cluster is the same. We can thus see that the sparse connection among the clusters did not add much complexity.

7. Using Conversion in Cryptographic Design

In this section we show an example of how conversion plays the role in designing cryptographic schemes by showing combination of the GS ZK and the AHO signature scheme. We then show another example that demonstrates conversion of an automorphic blind signature scheme designed originally in Type-III.

(1) Fine-Tuned GS Proof of Correct Commitment via Conversion

In the Groth-Sahai NIZK for PPE relations, it is often needed to prove that $[X]$ is a correct commitment of a public constant A in such a way that the proof can be simulated with $X = 1_{\mathbb{G}}$. In the original paper [30], it is done by proving a relation represented by a general multi-scalar multiplication equation (MSE). We present a technique that does the job with a less costly linear pairing product equation (PPE).

The Original Construction. Recall that, in the symmetric setting under the DLIN assumption, committing to a scalar value $a \in \mathbb{Z}_p$ requires two random values, say r_1 and r_2, in $\mathbb{Z}_p$, and committing to a group element $A \in \mathbb{G}$ uses three random values, $s_1, s_2, s_3 \in \mathbb{Z}_p$. We denote the commitment by $[a; r_1, r_2]$, and $[A; s_1, s_2, s_3]$, respectively. The genuine prover algorithm computes a default commitment of $1_{\mathbb{Z}_p}$ as $[1_{\mathbb{Z}_p}; 0, 0]$, and a proof for multi-scalar multiplication equation

$$[X]^1 \cdot A^{-[1_{\mathbb{Z}_p}]} = 1_{\mathbb{G}}. \tag{2}$$

Fine-Tuning in Type-I. Instead of using default $[1_{\mathbb{Z}_p}]$, the prover algorithm uses default commitment $[G^1; 0, 0, 0]$. Then prove a PPE

$$e([X], G)\, e(A^{-1}, [G^1]) = 1_{\mathbb{G}_T}. \tag{3}$$

instead of (2). Since we are considering the DLIN-based instantiation for now, (3) is a *linear* PPE that costs only 3 group elements whereas proof of (2) requires 9 elements.

Converting to Type-III. By converting the above proof system, we have an analogue proof system in the asymmetric setting based on the XDLIN assumption [34]. While the security is guaranteed by the conversion framework of [21], the quality of the resulting proof system must be examined.

Speaking from the conclusion, we have a clean split of its dependency graph without duplication except for the nodes representing the CRS. Thus, with duplicated CRS in $\mathbb{G}_0$ and $\mathbb{G}_1$, every group operation is done in either $\mathbb{G}_0$ or $\mathbb{G}_1$ and asymmetric pairing computation can be performed consistently. More importantly, the proof remains consisting of 3 group elements (and they are all in $\mathbb{G}_0$). Full details are presented in Section 5.1 of [40].

(2) AHO Signature + GSZK

AHO signature scheme in Type-I setting is summarized as follows. Let $gk := (p, \mathbb{G}, \mathbb{G}_T, e, G)$ be a symmetric bilinear groups. A public-key is $(gk, A_0, A_1, A_2, A_3, B_0, B_1, B_2, B_3, G_z, G_r, H_z, H_u, G_1, \ldots, G_n, H_1, \ldots, H_n)$ for the message space of $\mathbb{G}^n$. A signature for message $(M_1, \ldots, M_n)$ is $\sigma = (Z, R, S, T, U, V, W) \in \mathbb{G}^7$. To prove possession of a correct signature for a message in the clear, a prover randomizes (S, T, V, W) into (S', T', V', W') in a way that $e(S, T) = e(S', T')$ and $e(V, W) = e(V', W')$ hold and then proves that pairing product equations

$$e(A_0, [A_1])\, e(A_2, [A_3]) = e(G_z, [Z]) \times e(G_r, [R]) e(S', [T']) \prod_{i=1}^{n} e(G_i, [M_i]), \quad \text{and} \tag{4}$$

$$e(B_0, [B_1])\, e(B_2, [B_3]) = e(H_z, [Z]) \times e(H_u, [U]) e(V', [W']) \prod_{i=1}^{n} e(H_i, [M_i]) \tag{5}$$

hold with respect to committed variables in the brackets. Additionally, relation (3) for every public value $X \in \{A_1, A_3, B_1, B_3, M_1, \ldots, M_n\}$ is proved by using our fine-tuning technique to show the correctness of the commitments.

We then consider four approaches to obtain Type-III counterpart of the above scheme. Table 2 summarizes the performance of the resulting schemes in Type-III in terms of the proof size and number of pairings in verification. Sizes in bits are estimated assuming the use of KSS-16 curves [57] where $|\mathbb{G}_1|/|\mathbb{G}_0| = 4$.

Conversion: By converting the above scheme we obtain a scheme in Type-III. Details for the proof part are presented in Sect. A.1. In the resulting scheme, CRS is entirely duplicated but elements in the proofs, public-keys, and messages are assigned to either $\mathbb{G}_0$ or $\mathbb{G}_1$ without duplication. It is particularly important to point out that X and $[X]$ in (3) are assigned to the same group without duplicating X while proving (3) as a linear PPE. This approach is the most efficient in the proof size since most of commitments and proofs can be allocated in $\mathbb{G}_0$.

Direct instantiation 1 (with duplicated messages): Next we consider instantiating the GS-proofs directly over Type-III groups based on the SXDH assumption. The fine-tuned construction is only possible when public constants paired with committed variables are duplicated. Therefore, elements $\{A_1, A_3, B_1, B_3, M_1, \ldots, M_n\}$ have to be duplicated. Duplicated key elements, A_1, A_3, B_1, and B_3 will be a part of the public-key. On the other hand, duplicated message $M_1, \ldots, M_n$ must be sent to the verifier as a part of the proof.

Direct instantiation 2 (with duplicated keys): When duplicating M_i is prohibiting, a workaround would be to commit to public-key elements G_i and H_i instead. Duplicated G_i and H_i can be included in the public-key (thus we do not count it in the proof size). Unfortunately, this approach is not efficient in terms of proof size since the proofs of correct commitment for both G_i and H_i doubles the proof length. On the other hand, it allows efficient

Table 2 Comparison of proof size and number of pairings between conversion-aided and three direct constructions. The message is in $\mathbb{G}_0$. Proof size counts number of group elements in relevant GS commitments and proofs. The size in bits is estimated assuming KSS-16 curve with base field size of 340 bits, i.e., $\lambda := |\mathbb{G}_0| = 340$. Column "naive" counts the number of pairings literally in the verification equations, and "batched" counts the number of pairings in batch verification.

Construction	Duplicated	Proof Size			# of Pairings	
	Object	$\mathbb{G}_0$	$\mathbb{G}_1$	in bits	naive	batched
Conversion	crs	$6n + 39$	6	$(6n + 63)\lambda$	$18n + 90$	$2n + 20$
Direct (1)	msg	$2n + 18$	$3n + 12$	$(14n + 66)\lambda$	$12n + 60$	$2n + 17$
Direct (2)	pk	$4n + 26$	$4n + 16$	$(20n + 90)\lambda$	$20n + 84$	$n + 23$
Direct (3)	-	$4n + 26$	$4n + 20$	$(20n + 106)\lambda$	$22n + 100$	$2n + 22$

Table 3 Comparison of the signature size and number of pairings in verification between conversion-aided and direct instantiations of verifier's algorithm for the automorphic blind signature scheme [35]. The message is $(M, N) \in \mathbb{G}_0 \times \mathbb{G}_1$. Duplication of $\tilde{D}$ is needed for computing proofs but not for verification.

Construction	Duplicated	Size of Blind Sig.			# of Pairings	
	Objects	$\mathbb{G}_0$	$\mathbb{G}_1$	in bits	naive	batched
Conversion	crs, $\tilde{D}$	24	6	48 λ	64	13
Original [35]	-	18	16	82 λ	68	13

batch verification. The reason is that pairings corresponding to $e([G_i], M_i)$ and $e([H_i], M_i)$ in the verification can be merged into one pairing associated to M_i while at least two pairings are needed to deal with $e(G_i, [M_i])$ and $e(H_i, [M_i])$ in the above approaches.

Direct instantiation 3 (without duplication): Finally, we consider avoiding duplication at all in the direct instantiation of GS proofs in Type-III by following the original approach using MSE (2). As expected, both proof size and number of pairings increase due to the MSEs. Use of batch verification is not quite effective, either.

As we see from Table 2, the scheme obtained by conversion has advantage in the proof size as it includes less elements from $\mathbb{G}_1$, whose representation is 4 times larger than those from $\mathbb{G}_0$ in the case of KSS-16 curves. Regarding the computational workload, when batch verification is taken into account, there is not much difference for small n whichever approach is taken. But for large n, direct instantiation in Type-III with duplicated public-key is more advantageous.

(3) Automorphic Blind Signature Scheme

Examples so far deals with schemes designed purely in Type-I. Now we show that schemes designed originally in Type-III are also incorporated into our framework for finding optimal deployment of source groups and perhaps finding more efficient GS-proofs used there.

We show a converted scheme converted from the automorphic blind signature scheme in [35], a blind signature is a GS-proof for one's possession of a correct (plain) automorphic signature on a clear message. In total, a signature that a verifier receive except for the message is of size $24|\mathbb{G}_0| + 6|\mathbb{G}_1|$, which is 48 λ bits at the same parameter setting (KSS-16 at base field size of $\lambda = 340$ bits) as in the previous case. It compares to the original construction that requires $6|\mathbb{G}_0|(= 3 \cdot 2|\mathbb{G}_0|)$ for committing to (A, B, R), $4|\mathbb{G}_1|(= 2 \cdot 2|\mathbb{G}_1|)$ for $(\tilde{D}, \tilde{S})$, and $3 \cdot (4|\mathbb{G}_0| + 4|\mathbb{G}_1|)$ for the proof. This sums up to $18|\mathbb{G}_0| + 16|\mathbb{G}_1|$ and it turns out 82 λ bits as above.

8. Conclusion

We have shown that scaling bilinear-type conversion in general is an essentially difficult problem. We at the same time develop a practical conversion method based on 0-1 Integer Programming and demonstrate its performance and scalability through experiments over real and randomly generated targets of conversion. Usefulness of the conversion method has been shown also in its application to conversion-aided cryptographic scheme design. It can be seen as a step toward realizing automated modular design of cryptographic schemes and protocols. Yet, depending on the target schemes, direct instantiation in Type-III based on SXDH can be better than converted schemes. We conclude that it is an interesting research and engineering target to develop an automated conversion method that takes such options into its optimization.

Acknowledgments

The authors thank Susan Hohenberger Waters and co-authors of [20, 22] for their help to understand AutoGroup. We also thank to Takeya Tango for an alternative sanity checking method. Comments from Jens Groth in early stage of this research are appreciated. Special thanks to the developers of SCIP [25] for their quality software.

References

[1] M. Abe, F. Hoshino, and M. Ohkubo, "Design in Type-I, run in Type-III: Fast and scalable bilinear-type conversion using integer programming," Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, Aug. 2016, Proceedings, Part III, M. Robshaw and J. Katz, eds., Lecture Notes in Computer Science, vol.9816, pp.387–415, Springer, 2016. doi:10.1007/978-3-662-53015-3_14.

[2] S.D. Galbraith, K.G. Paterson, and N.P. Smart, "Pairings for cryptographers," Discrete Appl. Math., vol.156, no.16, pp.3113–3121, 2008. doi:10.1016/j.dam.2007.12.010.

[3] A. Joux, "Faster index calculus for the medium prime case application to 1175-bit and 1425-bit finite fields," Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 2013. Proceedings, T. Johansson and P.Q. Nguyen, eds., Lecture Notes in Computer Science, vol.7881, pp.177–193, Springer, 2013. doi:10.1007/978-3-642-38348-9_11.

[4] A. Joux, "A new index calculus algorithm with complexity $L(1/4 + o(1))$ in very small characteristic," IACR Cryptology ePrint Archive, vol.2013, p.95, 2013. URL: http://eprint.iacr.org/2013/095

[5] F. Göloglu, R. Granger, G. McGuire, and J. 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, vol.2013, p.74, 2013. URL: http://eprint.iacr.org/2013/074

[6] R. Barbulescu, P. Gaudry, A. Joux, and E. Thomé, "A quasi-polynomial algorithm for discrete logarithm in finite fields of small

characteristic," IACR Cryptology ePrint Archive, vol.2013, p.400, 2013. URL: http://eprint.iacr.org/2013/400

[7] R. Barbulescu, P. Gaudry, A. Joux, and E. Thomé, "A heuristic quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic," Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 2014. Proceedings, P.Q. Nguyen and E. Oswald, eds., Lecture Notes in Computer Science, vol.8441, pp.1–16, Springer, 2014. doi:10.1007/978-3-642-55220-5_1.

[8] A. Joux, "Discrete logarithms in small characteristic finite fields: A survey of recent advances (invited talk)," 34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 2017, Hannover, Germany, H. Vollmer and B. Vallée, eds., LIPIcs, vol.66, pp.3:1–3:1, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.STACS.2017.3.

[9] B. Libert, M. Joye, M. Yung, and T. Peters, "Secure efficient history-hiding append-only signatures in the standard model," in Katz [58], pp.450–473. doi:10.1007/978-3-662-46447-2_20.

[10] B. Libert and M. Joye, "Group signatures with message-dependent opening in the standard model," Topics in Cryptology - CT-RSA 2014 - The Cryptographer's Track at the RSA Conference 2014, San Francisco, CA, USA, Feb. 2014. Proceedings, J. Benaloh, ed., Lecture Notes in Computer Science, vol.8366, pp.286–306, Springer, 2014. doi:10.1007/978-3-319-04852-9_15.

[11] M. Abe, B. David, M. Kohlweiss, R. Nishimaki, and M. Ohkubo, "Tagged one-time signatures: Tight security and optimal tag size," Public-Key Cryptography - PKC 2013 - 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, Feb–March 2013. Proceedings, K. Kurosawa and G. Hanaoka, eds., Lecture Notes in Computer Science, vol.7778, pp.312–331, Springer, 2013. doi:10.1007/978-3-642-36362-7_20.

[12] M. Backes, D. Fiore, and R.M. Reischuk, "Verifiable delegation of computation on outsourced data," in A. Sadeghi, V.D. Gligor, and M. Yung, eds., [59], pp.863–874. doi:10.1145/2508859.2516681.

[13] B. Waters, "Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions," Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, Aug. 2009. Proceedings, S. Halevi, ed., Lecture Notes in Computer Science, vol.5677, pp.619–636, Springer, 2009. doi:10.1007/978-3-642-03356-8_36.

[14] B. Waters, "Efficient identity-based encryption without random oracles," Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 2005, Proceedings, R. Cramer, ed., Lecture Notes in Computer Science, vol.3494, pp.114–127, Springer, 2005. doi:10.1007/11426639_7.

[15] D. Boneh and H. Shacham, "Group signatures with verifier-local revocation," Proc. 11th ACM Conference on Computer and Communications Security, CCS 2004, Washington, DC, USA, Oct. 2004, V. Atluri, B. Pfitzmann, and P.D. McDaniel, eds., pp.168–177, ACM, 2004. doi:10.1145/1030083.1030106.

[16] N.P. Smart and F. Vercauteren, "On computable isomorphisms in efficient asymmetric pairing-based systems," Discrete Appl. Math., vol.155, no.4, pp.538–547, 2007. doi:10.1016/j.dam.2006.07.004.

[17] S. Chatterjee and A. Menezes, "On cryptographic protocols employing asymmetric pairings - The role of Ψ revisited," IACR Cryptology ePrint Archive, vol.2009, p.480, 2009. URL: http://eprint.iacr.org/2009/480

[18] S. Chatterjee, D. Hankerson, E. Knapp, and A. Menezes, "Comparing two pairing-based aggregate signature schemes," Des. Codes Cryptogr., vol.55, no.2-3, pp.141–167, 2010. doi:10.1007/s10623-009-9334-7.

[19] S. Chatterjee and A. Menezes, "On cryptographic protocols employing asymmetric pairings - The role of Ψ revisited," Discrete Appl. Math., vol.159, no.13, pp.1311–1322, 2011. doi:10.1016/j.dam.2011.04.021.

[20] J.A. Akinyele, M. Green, and S. Hohenberger, "Using SMT solvers to automate design tasks for encryption and signature schemes," in A. Sadeghi, V.D. Gligor, and M. Yung, eds., [59], pp.399–410. doi:10.1145/2508859.2516718.

[21] M. Abe, J. Groth, M. Ohkubo, and T. Tango, "Converting cryptographic schemes from symmetric to asymmetric bilinear groups," in J.A. Garay and R. Gennaro, eds., [60], pp.241–260. doi:10.1007/978-3-662-44371-2_14.

[22] J.A. Akinyele, C. Garman, and S. Hohenberger, "Automating fast and secure translations from Type-I to Type-III pairing schemes," Proc. 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, Oct. 2015, I. Ray, N. Li, and C. Kruegel, eds., pp.1370–1381, ACM, 2015. doi:10.1145/2810103.2813601.

[23] T. Tango, M. Abe, T. Okamoto, and M. Ohkubo, "Polynomial-time algorithm for deciding possibility of converting cryptographic schemes from Type-I to III pairing groups," Proc. SCIS 2015 The 32nd Symposium on Cryptography and Information Security, IEICE, Kokura, Japan, Jan. 2015 (in japanese).

[24] Gurobi Optimization, Inc., "Gurobi optimizer reference manual," In http://www.gurobi.com/. URL: http://www.gurobi.com/documentation/6.5/refman.pdf

[25] T. Achterberg, "CIP: Solving constraint integer programs," Mathematical Programming Computation, vol.1, no.1, pp.1–41, 2009. URL: http://mpc.zib.de/index.php/MPC/article/view/4

[26] G. Gamrath and M.E. Lübbecke, "Experiments with a generic dantzig-wolfe decomposition for integer programs," Experimental Algorithms, 9th International Symposium, SEA 2010, Ischia Island, Naples, Italy, May 2010. Proceedings, P. Festa, ed., Lecture Notes in Computer Science, vol.6049, pp.239–252, Springer, 2010. doi:10.1007/978-3-642-13193-6_21

[27] T. Koch, Rapid Mathematical Prototyping, Ph.D. thesis, Technische Universität Berlin, 2004. URL: http://nbn-resolving.de/urn:nbn:de:0297-zib-8346

[28] M. Melnick, "LiPS." URL: http://lipside.sourceforge.net/

[29] LINDO Systems, "LINDO." URL: http://www.lindo.com/

[30] J. Groth and A. Sahai, "Efficient noninteractive proof systems for bilinear groups," SIAM J. Comput., vol.41, no.5, pp.1193–1232, 2012. doi:10.1137/080725386.

[31] B. Blanchet, "Cryptoverif: A computationally sound mechanized prover for cryptographic protocols," Dagstuhl seminar Formal Protocol Verification Applied, Oct. 2007. URL: http://prosecco.gforge.inria.fr/personal/bblanche/talks/Dagstuhl07.pdf

[32] G. Barthe, E. Fagerholm, D. Fiore, J.C. Mitchell, A. Scedrov, and B. Schmidt, "Automated analysis of cryptographic assumptions in generic group models," in J.A. Garay and R. Gennaro, eds., [60], pp.95–112. doi:10.1007/978-3-662-44371-2_6.

[33] G. Barthe, E. Fagerholm, D. Fiore, A. Scedrov, B. Schmidt, and M. Tibouchi, "Strongly-optimal structure preserving signatures from type II pairings: Synthesis and lower bounds," in J. Katz, ed., [58], pp.355–376. doi:10.1007/978-3-662-46447-2_16.

[34] M. Abe, M. Chase, B. David, M. Kohlweiss, R. Nishimaki, and M. Ohkubo, "Constant-size structure-preserving signatures: Generic constructions and simple assumptions," Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, Dec. 2012. Proceedings, X. Wang and K. Sako, eds., Lecture Notes in Computer Science, vol.7658, pp.4–24, Springer, 2012. doi:10.1007/978-3-642-34961-4_3.

[35] M. Abe, G. Fuchsbauer, J. Groth, K. Haralambiev, and M. Ohkubo, "Structure-preserving signatures and commitments to group elements," J. Cryptol., vol.29, no.2, pp.363–421, 2016. doi:10.1007/s00145-014-9196-7.

[36] O. Blazy, G. Fuchsbauer, M. Izabachène, A. Jambert, H. Sibert, and D. Vergnaud, "Batch groth-sahai," Applied Cryptography and Network Security, 8th International Conference, ACNS 2010, Beijing, China, June 2010. Proceedings, J. Zhou and M. Yung, eds., Lecture Notes in Computer Science, vol.6123, pp.218–235, 2010. doi:10.1007/978-3-642-13708-2_14.

[37] M. Abe, F. Hoshino, and M. Ohkubo, "IPConv," on GitHub, 2018. URL: https://github.com/security-kouza/IPConv.git

[38] L.M. de Moura and N. Bjørner, "Z3: An efficient SMT solver," 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–April 2008. Proceedings, C.R. Ramakrishnan and J. Rehof, eds., Lecture Notes in Computer Science, vol.4963, pp.337–340, Springer, 2008. doi:10.1007/978-3-540-78800-3_24.

[39] A. Escala and J. Groth, "Fine-tuning groth-sahai proofs," in H. Krawczyk, ed., [61], pp.630–649. doi:10.1007/978-3-642-54631-0_36.

[40] M. Abe, F. Hoshino, and M. Ohkubo, "Design in Type-I, run in Type-III: Fast and scalable bilinear-type conversion using integer programming," Cryptology ePrint Archive: 2016/570, 2016. URL: http://eprint.iacr.org/2016/570

[41] E. Ghadafi, N.P. Smart, and B. Warinschi, "Groth-sahai proofs revisited," Public Key Cryptography - PKC 2010, 13th International Conference on Practice and Theory in Public Key Cryptography, Paris, France, May 2010. Proceedings, P.Q. Nguyen and D. Pointcheval, eds., Lecture Notes in Computer Science, vol.6056, pp.177–192, Springer, 2010. doi:10.1007/978-3-642-13013-7_11.

[42] A. Escala, G. Herold, E. Kiltz, C. Ràfols, and J.L. Villar, "An algebraic framework for diffie-hellman assumptions," Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, Aug. 2013. Proceedings, Part II, R. Canetti and J.A. Garay, eds., Lecture Notes in Computer Science, vol.8043, pp.129–147, Springer, 2013. doi:10.1007/978-3-642-40084-1_8.

[43] G. Herold, J. Hesse, D. Hofheinz, C. Ràfols, and A. Rupp, "Polynomial spaces: A new framework for composite-to-prime-order transformations," in J.A. Garay and R. Gennaro, eds., [60], pp.261–279. doi:10.1007/978-3-662-44371-2_15.

[44] D. Boneh, X. Boyen, and H. Shacham, "Short group signatures," Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, Aug. 2004, Proceedings, M.K. Franklin, ed., Lecture Notes in Computer Science, vol.3152, pp.41–55, Springer, 2004. doi:10.1007/978-3-540-28628-8_3.

[45] D. Boneh and M.K. Franklin, "Identity-based encryption from the weil pairing," Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, Aug. 2001, Proceedings, J. Kilian, eds., Lecture Notes in Computer Science, vol.2139, pp.213–229, Springer, 2001. doi:10.1007/3-540-44647-8_13.

[46] P. Fouque and M. Tibouchi, "Indifferentiable hashing to barreto-naehrig curves," Progress in Cryptology - LATINCRYPT 2012 - 2nd International Conference on Cryptology and Information Security in Latin America, Santiago, Chile, Oct. 2012. Proceedings, A. Hevia and G. Neven, eds., Lecture Notes in Computer Science, vol.7533, pp.1–17, Springer, 2012. doi:10.1007/978-3-642-33481-8_1.

[47] F. Hoshino, M. Abe, and M. Ohkubo, "Pairing type optimization problem and its hardness," Proc. SCIS 2018 2018 Symposium on Cryptography and Information Security 2018, IEICE, 2018.

[48] R.E. Tarjan, "Depth-first search and linear graph algorithms," SIAM J. Comput., vol.1, no.2, pp.146–160, 1972. doi:10.1137/0201010.

[49] A.V. Aho, J.E. Hopcroft, and J. Ullman, "Kosaraju's algorithm," in Data Structures and Algorithms, pp.222–229, Addison-Wesley Longman Publishing, Boston, MA, USA, 1st ed., 1983. The authors credit the algorithm of Section 6.7 to an unpublished paper from 1978 by S. Rao Kosaraju.

[50] M. Sharir, "A strong-connectivity algorithm and its applications in data flow analysis," Computers & Mathematics with Applications, vol.7, no.1, pp.67–72, 1981. doi:10.1016/0898-1221(81)90008-0.

[51] T. Tango, M. Abe, and T. Okamoto, "Implementation of automated translation for schemes on symmetric bilinear groups," Proc. SCIS 2014 The 31st Symposium on Cryptography and Information Security, IEICE, Kagoshima, Japan, Jan. 2014.

[52] R. Kohli, R. Krishnamurti, and P. Mirchandani, "The minimum satisfiability problem," SIAM J. Discrete Math., vol.7, no.2, pp.275–283, 1994. doi:10.1137/S0895480191220836.

[53] P.S.L.M. Barreto and M. Naehrig, "Pairing-friendly elliptic curves of prime order," Selected Areas in Cryptography, 12th International Workshop, SAC 2005, Kingston, ON, Canada, Aug. 2005, Revised Selected Papers, B. Preneel and S.E. Tavares, eds., Lecture Notes in Computer Science, vol.3897, pp.319–331, Springer, 2005. doi:10.1007/11693383_22.

[54] D. Boneh and X. Boyen, "Efficient selective-ID secure identity-based encryption without random oracles," Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2004, Proceedings, C. Cachin and J. Camenisch, eds., Lecture Notes in Computer Science, vol.3027, pp.223–238, Springer, 2004. doi:10.1007/978-3-540-24676-3_14.

[55] B. Libert, M. Yung, M. Joye, and T. Peters, "Traceable group encryption," in H. Krawczyk, ed., [61], pp.592–610. doi:10.1007/978-3-642-54631-0_34.

[56] E. Kiltz, "Chosen-ciphertext security from tag-based encryption," Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 2006, Proceedings, S. Halevi and T. Rabin, eds., Lecture Notes in Computer Science, vol.3876, pp.581–600, Springer, 2006. doi:10.1007/11681878_30.

[57] R. Barbulescu and S. Duquesne, "Updating key size estimations for pairings," J. Cryptol., pp.1–39, Jan. 2018. doi:10.1007/s00145-018-9280-5.

[58] J. Katz, ed., Public-Key Cryptography - PKC 2015 - 18th IACR International Conference on Practice and Theory in Public-Key Cryptography, Gaithersburg, MD, USA, March–April 2015, Proceedings, Lecture Notes in Computer Science, vol.9020, Springer, 2015. doi:10.1007/978-3-662-46447-2.

[59] A. Sadeghi, V.D. Gligor, and M. Yung, eds., 2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, Nov. 2013, ACM, 2013. URL: http://dl.acm.org/citation.cfm?id=2508859

[60] J.A. Garay and R. Gennaro, eds., Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, Aug. 2014, Proceedings, Part I, Lecture Notes in Computer Science, vol.8616, Springer, 2014. doi:10.1007/978-3-662-44371-2.

[61] H. Krawczyk, ed., Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 2014. Proceedings, Lecture Notes in Computer Science, vol.8383, Springer, 2014. doi:10.1007/978-3-642-54631-0.

Appendix A: Details of Converted Schemes in Sect. 7

A.1 Converted GSZK for AHO signature

Let parameters for AHO signature scheme be asymmetric bilinear groups $gk := (p, \mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_T, e, G, \tilde{G})$, verification-key $pk := (gk, \tilde{G}_z, \tilde{G}_r, \tilde{H}_z, \tilde{H}_u, \{\tilde{G}_i, \tilde{H}_i\}_{i=1}^{n}, \tilde{A}_0, A_1, \tilde{A}_1, A_2, \tilde{B}_0, B_1, \tilde{B}_1, B_2)$, message $msg := (M_1, \ldots, M_n)$, and signature $\sigma := (Z, R, U, \tilde{S}, T, \tilde{V}, W)$. CRS $\vec{u} \in \mathbb{G}_0^3$ and $\vec{\tilde{u}} \in \tilde{\mathbb{G}}_1^3$ are generated by using our fine-tuning technique. The relations to prove are PPEs (4), (5), and (3) re-numbered as follows.

$$\hat{e}(\tilde{A}_0, [A_1])\, \hat{e}(\tilde{A}_2, [A_3]) = \hat{e}(\tilde{G}_z, [Z]) \times \hat{e}(\tilde{G}_r, [R])\, \hat{e}(\tilde{S}', [T'']) \prod_{i=1}^{n} \hat{e}(\tilde{G}_i, [M_i]), \quad (\text{A}\cdot 1)$$

$$\hat{e}(\tilde{B}_0,[B_1])\,\hat{e}(\tilde{B}_2,[B_3]) = \hat{e}(\tilde{H}_z,[Z])\times \hat{e}(\tilde{H}_u,[U])\,\hat{e}(\tilde{V}',[W'])\prod_{i=1}^{n}\hat{e}(\tilde{H}_i,[M_i]), \qquad (\text{A}\cdot 2)$$

$$\hat{e}(\tilde{G},[X])\,\hat{e}([\tilde{G}],X^{-1}) = 1_{\mathbb{G}_T} \qquad (\text{A}\cdot 3)$$

for each $X \in \{A_1, A_3, B_1, B_3, M_i\}$. Here pairing $\hat{e}$ is defined as

$$\hat{e}(X,Y) = \begin{cases} e(X,Y) & (X \in \mathbb{G}_0 \wedge Y \in \mathbb{G}_1), \\ e(Y,X) & (Y \in \mathbb{G}_0 \wedge X \in \mathbb{G}_1), \\ \perp & (\text{otherwise}). \end{cases} \qquad (\text{A}\cdot 4)$$

The relations can be regarded as linear PPEs. In the rest of this section, we switch to additive notation for convenience of presenting GS-proofs.

[Prover Algorithm]

For each $Y \in \{Z, R, U, T', W', A_1, A_3, B_1, B_3, M_i\}$, commit Y by computing

$$[Y] := (\mathcal{O},\mathcal{O},Y) + \mathcal{S}_Y\,\vec{u} = (C_{1,Y}, C_{2,Y}, C_{3,Y}) \in \mathbb{G}_0^3.$$

with independently uniform $\mathcal{S}_Y \overset{\$}{\leftarrow} \mathbb{Z}_p^{1\times 3}$ where $\mathcal{S}_Y\,\vec{u}$ denotes elementwise scalar multiplication. Let $\mathcal{S}_{\tilde{G}} := (0,0,0) \in \mathbb{Z}_p^3$,

$$\mathcal{S}^{\top}_{(\text{A}\cdot 1)} := (\mathcal{S}_{A_1}, \mathcal{S}_{A_3}, \mathcal{S}_Z, \mathcal{S}_R, \mathcal{S}_{T'}, \mathcal{S}_{M_i}),$$
$$\mathcal{S}^{\top}_{(\text{A}\cdot 2)} := (\mathcal{S}_{B_1}, \mathcal{S}_{B_3}, \mathcal{S}_Z, \mathcal{S}_U, \mathcal{S}_{W'}, \mathcal{S}_{M_i}),\ \text{and}$$
$$\mathcal{S}^{\top}_{(\text{A}\cdot 3),X} := \left(\mathcal{S}_{\tilde{G}}, \mathcal{S}_X\right).$$

Compute $\tilde{\theta}_{(\text{A}\cdot 1)}$, $\tilde{\theta}_{(\text{A}\cdot 2)}$ and $\theta_{(\text{A}\cdot 3),X}$ for $X \in \{A_1, A_3, B_1, B_3, M_1, \ldots, M_i\}$ where:

$$\tilde{\theta}_{(\text{A}\cdot 1)} := \mathcal{S}^{\top}_{(\text{A}\cdot 1)} \begin{pmatrix} \mathcal{O} & \mathcal{O} & \tilde{A}_0 \\ \mathcal{O} & \mathcal{O} & \tilde{A}_2 \\ \mathcal{O} & \mathcal{O} & \tilde{G}_z^{-1} \\ \mathcal{O} & \mathcal{O} & \tilde{G}_r^{-1} \\ \mathcal{O} & \mathcal{O} & \tilde{G}_t^{-1} \\ \mathcal{O} & \mathcal{O} & \tilde{G}_i^{-1} \end{pmatrix} = \begin{pmatrix} \mathcal{O} & \mathcal{O} & \tilde{\theta}_{1,(\text{A}\cdot 1)} \\ \mathcal{O} & \mathcal{O} & \tilde{\theta}_{2,(\text{A}\cdot 1)} \\ \mathcal{O} & \mathcal{O} & \tilde{\theta}_{3,(\text{A}\cdot 1)} \end{pmatrix} \in \tilde{\mathbb{G}}_1^{3\times 3},$$

$$\tilde{\theta}_{(\text{A}\cdot 2)} := \mathcal{S}^{\top}_{(\text{A}\cdot 2)} \begin{pmatrix} \mathcal{O} & \mathcal{O} & \tilde{B}_0 \\ \mathcal{O} & \mathcal{O} & \tilde{B}_2 \\ \mathcal{O} & \mathcal{O} & \tilde{H}_z^{-1} \\ \mathcal{O} & \mathcal{O} & \tilde{H}_u^{-1} \\ \mathcal{O} & \mathcal{O} & \tilde{H}_w^{-1} \\ \mathcal{O} & \mathcal{O} & \tilde{H}_i^{-1} \end{pmatrix} = \begin{pmatrix} \mathcal{O} & \mathcal{O} & \tilde{\theta}_{1,(\text{A}\cdot 2)} \\ \mathcal{O} & \mathcal{O} & \tilde{\theta}_{2,(\text{A}\cdot 2)} \\ \mathcal{O} & \mathcal{O} & \tilde{\theta}_{3,(\text{A}\cdot 2)} \end{pmatrix} \in \tilde{\mathbb{G}}_1^{3\times 3},$$

$$\theta_{(\text{A}\cdot 3),X} := \mathcal{S}^{\top}_{(\text{A}\cdot 3),X} \begin{pmatrix} \mathcal{O} & \mathcal{O} & G \\ \mathcal{O} & \mathcal{O} & X^{-1} \end{pmatrix} = \begin{pmatrix} \mathcal{O} & \mathcal{O} & \theta_{1,(\text{A}\cdot 3),X} \\ \mathcal{O} & \mathcal{O} & \theta_{2,(\text{A}\cdot 3),X} \\ \mathcal{O} & \mathcal{O} & \theta_{3,(\text{A}\cdot 3),X} \end{pmatrix} \in \mathbb{G}_0^{3\times 3}.$$

Output all $[Y]$, $\tilde{\theta}_{(\text{A}\cdot 1)}$, $\tilde{\theta}_{(\text{A}\cdot 2)}$, and $\theta_{(\text{A}\cdot 3),X}$ dropping redundant $\mathcal{O}$.

[Verifier Algorithm]

Let $\tilde{\mathcal{X}} \tilde{\bullet} \mathcal{Y}$ denote a binary operation for $\tilde{\mathcal{X}} \in \mathbb{G}_1^3$ and $\mathcal{Y} \in \mathbb{G}_0^3$ that results in 3×3 matrix consisting of elements of $\mathbb{G}_T$ obtained by computing pairings for every combination of elements in $\tilde{\mathcal{X}}$ and $\mathcal{Y}$. Given the above proof and CRS as input, output 1 (as accept) if all the following equations hold. Output 0, otherwise.

$$\begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ \tilde{A}_0 \end{pmatrix} \tilde{\bullet} \begin{pmatrix} C_{1,A_1} \\ C_{2,A_1} \\ C_{3,A_1} \end{pmatrix} + \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ \tilde{A}_2 \end{pmatrix} \tilde{\bullet} \begin{pmatrix} C_{1,A_3} \\ C_{2,A_3} \\ C_{3,A_3} \end{pmatrix} + \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ \tilde{G}_z^{-1} \end{pmatrix} \tilde{\bullet} \begin{pmatrix} C_{1,Z} \\ C_{2,Z} \\ C_{3,Z} \end{pmatrix} +$$
$$\begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ \tilde{G}_r^{-1} \end{pmatrix} \tilde{\bullet} \begin{pmatrix} C_{1,R} \\ C_{2,R} \\ C_{3,R} \end{pmatrix} + \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ \tilde{S}'^{-1} \end{pmatrix} \tilde{\bullet} \begin{pmatrix} C_{1,T'} \\ C_{2,T'} \\ C_{3,T'} \end{pmatrix} + \sum_{i=1}^{n} \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ \tilde{G}_i^{-1} \end{pmatrix} \tilde{\bullet} \begin{pmatrix} C_{1,M_i} \\ C_{2,M_i} \\ C_{3,M_i} \end{pmatrix}$$
$$= \left(\tilde{\theta}_{(\text{A}\cdot 1)}\right)^{\top} \tilde{\bullet} (\vec{u})^{\top},$$

$$\begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ \tilde{B}_0 \end{pmatrix} \tilde{\bullet} \begin{pmatrix} C_{1,B_1} \\ C_{2,B_1} \\ C_{3,B_1} \end{pmatrix} + \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ \tilde{B}_2 \end{pmatrix} \tilde{\bullet} \begin{pmatrix} C_{1,B_3} \\ C_{2,B_3} \\ C_{3,B_3} \end{pmatrix} + \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ \tilde{H}_z^{-1} \end{pmatrix} \tilde{\bullet} \begin{pmatrix} C_{1,Z} \\ C_{2,Z} \\ C_{3,Z} \end{pmatrix} +$$
$$\begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ \tilde{H}_u^{-1} \end{pmatrix} \tilde{\bullet} \begin{pmatrix} C_{1,U} \\ C_{2,U} \\ C_{3,U} \end{pmatrix} + \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ \tilde{V}'^{-1} \end{pmatrix} \tilde{\bullet} \begin{pmatrix} C_{1,W'} \\ C_{2,W'} \\ C_{3,W'} \end{pmatrix} + \sum_{i=1}^{n} \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ \tilde{H}_i^{-1} \end{pmatrix} \tilde{\bullet} \begin{pmatrix} C_{1,M_i} \\ C_{2,M_i} \\ C_{3,M_i} \end{pmatrix}$$
$$= \left(\tilde{\theta}_{(\text{A}\cdot 2)}\right)^{\top} \tilde{\bullet} (\vec{u})^{\top},$$

$$\begin{pmatrix} C_{1,X} \\ C_{2,X} \\ C_{3,X} \end{pmatrix} \tilde{\bullet} \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ \tilde{G} \end{pmatrix} + \begin{pmatrix} \tilde{C}_{1,\tilde{G}} \\ \tilde{C}_{2,\tilde{G}} \\ \tilde{C}_{3,\tilde{G}} \end{pmatrix} \tilde{\bullet} \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ X^{-1} \end{pmatrix} = \left(\vec{\tilde{u}}\right)^{\top} \tilde{\bullet} (\theta_{(\text{A}\cdot 3),X})^{\top},$$

for $X \in \{A_1, A_3, B_1, B_3, M_i\}$ where $(\tilde{C}_{1,\tilde{G}}, \tilde{C}_{2,\tilde{G}}, \tilde{C}_{3,\tilde{G}}) := (\mathcal{O},\mathcal{O},\tilde{G})$.

A.2 Converted Automorphic Blind Signature Scheme

This section presents details of automorphic blind signature scheme obtained by conversion. A full description includes key generation, blinding, signing, unblinding, verification algorithms, and also security proofs. Here, we focus on presenting user's and verifier's algorithms in transferring a blind signature. They actually consist of prover and verifier algorithms like the previous case. CRS $\vec{u} \in \mathbb{G}_0^3$ and $\vec{\tilde{u}} \in \tilde{\mathbb{G}}_1^3$ are generated by using our fine-tuning technique Let parameters be asymmetric bilinear groups $gk := (p, \mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_T, e, G, \tilde{G})$, verification-key $pk := (gk, F, K, T, X(= G^x), \tilde{Y}(= \tilde{G}^x))$, message $(M(= G^m), \tilde{N}(= \tilde{G}^m))$. An automorphic blind signature is a witness indistinguishable GS-proof for relations as re-numbered as follows.

$$\hat{e}([A],\tilde{Y})\,\hat{e}([A],[\tilde{D}]) = \hat{e}(K,\tilde{G})\,\hat{e}(M,\tilde{G})\,\hat{e}(T,[S]), \qquad (\text{A}\cdot 5)$$

$$\hat{e}([B],\tilde{G}) = \hat{e}(F,[\tilde{D}]),\ \text{and} \qquad (\text{A}\cdot 6)$$

$$\hat{e}([R],\tilde{G}) = \hat{e}(G,[S]). \qquad (\text{A}\cdot 7)$$

With pairing $\hat{e}$ defined as (A·4), the second and third relations are regarded as linear PPEs. Again, we switch to additive notation while describing GS-proofs in the following.

[Blind Signature Issuing Algorithm]

Commit to $\delta \in (A, B, R)$ and $\tilde{\rho} \in (\tilde{D}, \tilde{S})$ by

$$[\delta] := (\mathcal{O},\mathcal{O},\delta) + \mathcal{S}_\delta\,\vec{u} = (C_{1,\delta}, C_{2,\delta}, C_{3,\delta}) \in \mathbb{G}_0^3, \quad \text{and}$$
$$[\tilde{\rho}] := (\mathcal{O},\mathcal{O},\tilde{\rho}) + \mathcal{S}_{\tilde{\rho}}\,\vec{\tilde{u}} = (C_{1,\tilde{\rho}}, C_{2,\tilde{\rho}}, C_{3,\tilde{\rho}}) \in \mathbb{G}_1^3.$$

where $\mathcal{S}_\delta \overset{\$}{\leftarrow} \mathbb{Z}_p^{1\times 3}$ and $\mathcal{S}_{\tilde{\rho}} \overset{\$}{\leftarrow} \mathbb{Z}_p^{1\times 3}$. Let T_p be a random 3×3 matrix over $\mathbb{Z}_p$. Compute $\theta_{(\text{A}\cdot 5)}$, $\theta_{(\text{A}\cdot 6)}$, and $\theta_{(\text{A}\cdot 7)}$ as:

$$\theta_{(\text{A}\cdot 5)} = \mathcal{S}_A^{\top}(\mathcal{O},\mathcal{O},X) + \mathcal{S}_A^{\top}(\mathcal{O},\mathcal{O},D) + \mathcal{S}_{\tilde{D}}^{\top}(\mathcal{O},\mathcal{O},A) + \mathcal{S}_A^{\top}\mathcal{S}_{\tilde{D}}\,\vec{u} - \mathcal{S}_{\tilde{S}}^{\top}(\mathcal{O},\mathcal{O},T) + (T_p - T_p^{\top})\,\vec{u},$$

$$\theta_{(\text{A}\cdot 6)} = \mathcal{S}_B^{\top}(\mathcal{O},\mathcal{O},G) - \mathcal{S}_{\tilde{D}}^{\top}(\mathcal{O},\mathcal{O},F),\ \text{and}$$

$$\theta_{(\mathrm{A}\cdot 7)} = \mathcal{S}_R^\top(\mathcal{O}, \mathcal{O}, G) - \mathcal{S}_{\tilde{S}}^\top(\mathcal{O}, \mathcal{O}, G).$$

Output all $[\delta]$, $[\tilde{\rho}]$, $\theta_{(\mathrm{A}\cdot 5)}$, $\theta_{(\mathrm{A}\cdot 6)}$, and $\theta_{(\mathrm{A}\cdot 7)}$ without redundant $\mathcal{O}$ as a blind signature.

[Verifier Algorithm]

Given the above blind signature and message $msg := (M, \tilde{N})$, output 1 if all the following equations hold. Output 0, otherwise.

$$\begin{pmatrix} C_{1,A} \\ C_{2,A} \\ C_{3,A} \end{pmatrix} \tilde{\bullet} \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ \tilde{Y} \end{pmatrix} + \begin{pmatrix} C_{1,A} \\ C_{2,A} \\ C_{3,A} \end{pmatrix} \tilde{\bullet} \begin{pmatrix} C_{1,\tilde{D}} \\ C_{2,\tilde{D}} \\ C_{3,\tilde{D}} \end{pmatrix} = \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ K \end{pmatrix} \tilde{\bullet} \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ \tilde{G} \end{pmatrix}$$

$$+ \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ M \end{pmatrix} \tilde{\bullet} \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ \tilde{G} \end{pmatrix} + \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ T \end{pmatrix} \tilde{\bullet} \begin{pmatrix} C_{1,\tilde{S}} \\ C_{2,\tilde{S}} \\ C_{3,\tilde{S}} \end{pmatrix} + (\theta_{(\mathrm{A}\cdot 5)})^\top \tilde{\bullet} \left(\vec{\tilde{u}}\right)^\top,$$

$$\begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ \tilde{G} \end{pmatrix} \tilde{\bullet} \begin{pmatrix} C_{1,B} \\ C_{2,B} \\ C_{3,B} \end{pmatrix} = \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ F \end{pmatrix} \tilde{\bullet} \begin{pmatrix} C_{1,\tilde{D}} \\ C_{2,\tilde{D}} \\ C_{3,\tilde{D}} \end{pmatrix} + (\theta_{(\mathrm{A}\cdot 6)})^\top \tilde{\bullet} \left(\vec{\tilde{u}}\right)^\top,$$

$$\begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ \tilde{G} \end{pmatrix} \tilde{\bullet} \begin{pmatrix} C_{1,R} \\ C_{2,R} \\ C_{3,R} \end{pmatrix} = \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ G \end{pmatrix} \tilde{\bullet} \begin{pmatrix} C_{1,\tilde{S}} \\ C_{2,\tilde{S}} \\ C_{3,\tilde{S}} \end{pmatrix} + (\theta_{(\mathrm{A}\cdot 7)})^\top \tilde{\bullet} \left(\vec{\tilde{u}}\right)^\top.$$

Appendix B: Sample Dependency Graphs in Sect. 6.2

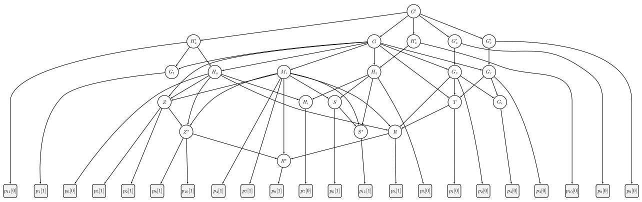

Fig. A·1 A dependency graph for Tagged One-time Signature Scheme in [11].

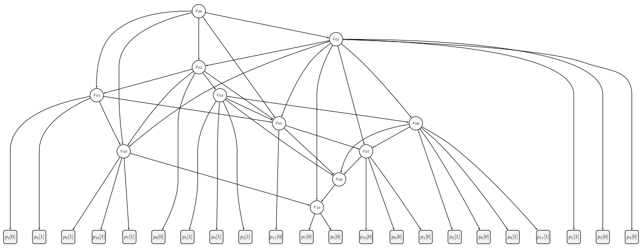

Fig. A·2 A random dependency graph with the same number of pairing nodes as above.

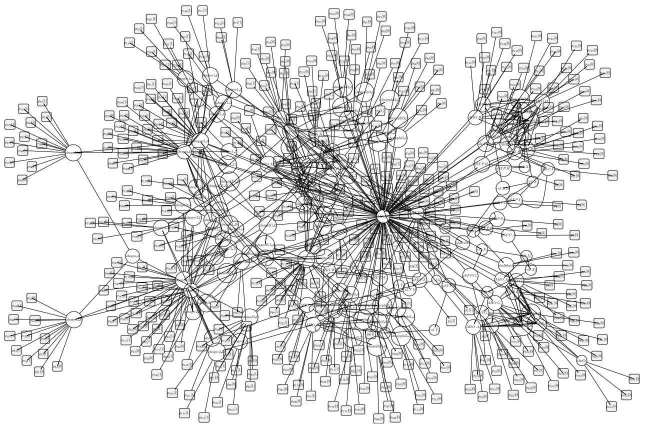

Fig. A·3 A dependency graph of AHO signature scheme with GS Zero-knowledge proof.

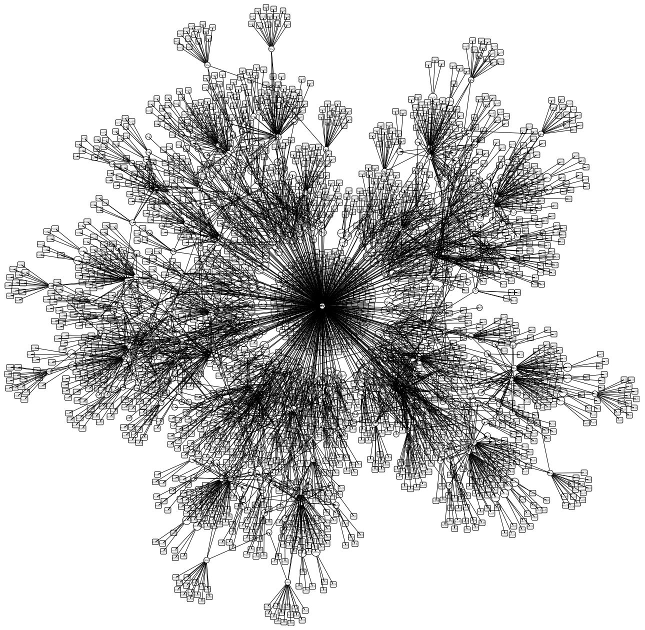

Fig. A·4 An example of a random cluster dependency graph at $n = 5$.

Masayuki Abe has been working for NTT (Nippon Telegraph and Telephone Corporation, Japan) since 1992. He received Ph.D. from University of Tokyo in 2002. Currently, he is a senior distinguished research scientist in NTT Secure Platform Laboratories. He served as a program chair for CT-RSA'07, ACM ASIACCS'08, and Asiacrypt'10. His research interest includes digital signatures, public-key encryption, and efficient instantiation of cryptographic protocols.

Fumitaka Hoshino received the B.Eng. and M.Eng. degrees from University of Tokyo, Japan, in 1996 and 1998, respectively. He is a senior scientist in NTT Secure Platform Laboratories, and a doctoral student at Tokyo Institute of Technology. His current research interests cover a wide range of topics between applied mathematics and computer science e.g. algorithmic number theory, combinatorial optimization, and cryptology.

Miyako Ohkubo received the B.E., and M.E. degrees from Shinshu University in 1995 and 1997, respectively, and Ph.D. degree from the Chuo University in 2004. She had worked at NTT from 1997 to 2010. She is currently a senior researcher of security architecture laboratory in National Institute of Information and Communications Technology. She received the SCIS Paper Award in 2000. She is a member of the International Association for Cryptologic Research (IACR).