Design in Type-I, Run in Type-III: Fast and Scalable Bilinear-Type Conversion using Integer Programming

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

[1] Information Sharing Platform Laboratories
NTT Corporation, Japan
[2] Security Fundamentals Laboratory, CSR
NICT, Japan
[3] Graduate School of Informatics
Kyoto University, Japan
[4] School of Computing, Department of Mathematical and Computing Science
Tokyo Institute of Technology, Japan

Abstract. Bilinear type conversion is to convert cryptographic schemes designed over symmetric groups instantiated with imperilled curves into ones that run over more secure and efficient asymmetric groups. In this paper we introduce a novel type conversion method called *IPConv* using 0-1 Integer Programming. Instantiated with a widely available IP solver, it instantly converts existing intricate schemes, and can process large-scale schemes that involves more than a thousand variables and hundreds of pairings.
Such a quick and scalable method allows a new approach in designing cryptographic schemes over asymmetric bilinear groups. Namely, designers work without taking much care about asymmetry of computation but the converted scheme runs well in the asymmetric setting. We demonstrate the usefulness of conversion-aided design by presenting somewhat counter-intuitive examples where converted DLIN-based Groth-Sahai proofs are more compact than manually built SXDH-based proofs.

Keywords. Conversion, Bilinear Groups, Integer Programming, Groth-Sahai Proofs, Zero-Knowledge

1 Introduction

1.1 Background

Prime-order bilinear groups consist of source groups $\mathbb{G}_0$ and $\mathbb{G}_1$, target group $\mathbb{G}_T$, and a pairing $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_T$. In so called Type-I bilinear groups, $\mathbb{G}_0 = \mathbb{G}_1$, i.e., the pairing is symmetric. It has been a popular choice in early research and development. Recent progress in analyzing symmetric pairing groups instantiated with small characteristic curves [30, 31, 26, 9] motivates crypto designers to move to Type-III groups where $\mathbb{G}_0 \neq \mathbb{G}_1$, i.e., the pairing is asymmetric, and no efficient mapping is known between $\mathbb{G}_0$ and $\mathbb{G}_1$. For Type-III groups, no such weakness

has been observed until now and efficient instantiations have been developed. Yet Type-I setting is useful for presenting and understanding cryptographic schemes for their simplicity. Besides, number of schemes have been designed only for Type-I groups in the literature, e.g. [37, 36, 2, 8, 41, 40, 16].

Bilinear-type conversion is a method to translate schemes designed for Type-I groups into ones that work over Type-III groups. Cryptographic schemes designed in Type-I setting do not necessarily work in Type-III 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.

Besides saving existing schemes over Type-I groups, conversion plays the role in putting *"Design in Type-I and Run in Type-III"* paradigm into practice as suggested in the pioneering work by Akinyele, Green and Hohenberger [7]. That is, let crypto designers focus on their high-level idea of construction without taking much care about asymmetry of computation by designing in Type-I setting, and then convert the results to obtain executable schemes over Type-III groups. For conversion tools to be useful, the processing speed and scalability are of importance on top of the performance of the final executables. Like compilers for high-level programming languages a conversion tool will be executed over and over again throughout the development. Quick response is strongly desired for productivity and stress-free developing environment. Its importance increases when large-scale systems that consist of several building blocks are targeted. Nevertheless, only small-scale monolithic schemes has been targeted so far. Hence the validity of the design paradigm has not been well substantiated yet.

1.2 Our Contribution

We propose a new efficient conversion algorithm, which we call 'IPConv', based on 0-1 Integer Programming (IP). A technical highlight that separates this work from previous ones [7, 6] is how to encode several kinds of constraints into a system of linear relations over binary variables, and how to implement ones metric into an objective function the 0-1 IP minimizes subject to the constraints. The idea of encoding computational constraints into an objective function follows from previous works. 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 [28, 5, 24, 33, 35, 34].

Performance of IPConv is demonstrated by experiments over real cryptographic schemes in the literature. IPConv instantly completes the task even

for complex schemes. To measure the scalability, 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 [5] as an engine. The concrete figures of course become magnitude of better with a powerful commercial IP solver e.g. [28]. Scaling up to thousands of pairings may seem an overkill. However, for instance, schemes that include Groth-Sahai (GS) proof system [27] easily involve dozens or even hundreds of pairings when their security proofs are taken into account. Furthermore, tools such as [12, 10, 11] 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.

Next we, for the first time, prove the usefulness of the conversion-assisted design for middle-scale schemes. It is shown that schemes involving GS proofs based on decision linear assumption (DLIN) can be converted to ones based on XDLIN assumption [1] in Type-III so that they are more efficient than their direct instantiation based on the symmetric external Diffie-Hellman assumption (SXDH). The result may be counter-intuitive since the commitments and proofs of SXDH-based GS-proofs require less group elements than those based on DLIN. Key observations that explain our result are:

- Relations such as $e(X, A) = e(B, Y)$ for variables X and Y are considered as linear pairing product equations (PPEs) in Type-I whose proof consists of 3 elements whereas they are more costly two-sided PPEs in Type-III that costs 8 elements. Proving linear PPEs can be converted without duplicating the proofs and commitments in general.
- Commitments and proofs in the converted proof system are allocated mostly in $\mathbb{G}_0$ whereas they appear in both $\mathbb{G}_0$ and $\mathbb{G}_1$ in direct SXDH-based instantiation. Taking the fact that elements in $\mathbb{G}_1$ is typically twice as long as those in $\mathbb{G}_0$ in bits, the former can be shorter than the latter in some cases.

Our first example in Section 5.2 is a scheme for showing ones possession of a correct structure-preserving signature [3] on a public message in zero-knowledge. The scheme obtained by conversion yields proofs that are up to 50% shorter (asymptotic in the message length) than those generated by direct constructions based on SXDH. It uses a novel fine-tuning for zero-knowledge GS-proofs (GSZK) presented in Section 5.1 that takes the above mentioned advantages.

Our second example in Section 5.3 is to demonstrate that our framework can be applied to schemes that is already designed in Type-III setting to seek for better instantiations. We pick an automorphic blind signature scheme [3] that involves GS-proofs and is secure under SXDH assumption in Type-III setting. We show that the proofs can be replaced with the DLIN-based ones and it can be converted to work in Type-III under XDLIN assumption. Though the GS-proofs are witness indistinguishable for this time, it still can take the above mentioned advantages and saves 28% in the length of the signatures 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 [13]. This unveils an open issue on optimization of schemes involving GS-proofs.

1.3 Related Works

There are some conversion systems in the literature. Early works on type conversion, e.g. [39, 18, 17, 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 [7] is the first automated conversion system that converts schemes from Type-I to Type-III. 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 [21]. It then search for the 'optimal' solution that conforms to other mathematical constraints and ones preferences. When there are number of possible solutions, the performance gets lower. In this pioneering work, the security of the resulting converted scheme was not guaranteed. In [4], Abe et. al., established a theoretical ground for preserving security during conversion. Their framework, reviewed in Section 2, provides useful theorems for security guarantee. But their conversion algorithm is basically a brute-force search over all possible conversions and it requires exponential time in the number of pairings. Recently in [6], Akinyele, Garman, and Hohenberger introduced an upgraded system called AutoGroup+ that integrates the framework of [4] 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.

Regarding Groth-Sahai zero-knowledge proofs, the closest work is the one by Escala and Groth in [22]. 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 G^0 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 in Section 5.1 uses the same property for the commitment of G but use it in a different manner that is most effective in Type-I setting. Another close work is [25] 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 [23, 29], 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 Conversion based on Dependency Graphs

2.1 Overview

In this section we review the framework in [4]. 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 [4] that if the original assumptions are valid in Type-I generic bilinear group model [15], 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.

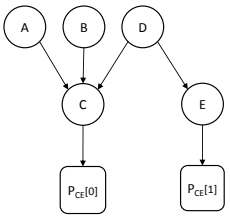
We follow the framework of [4] that consists of the following four steps.

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 [4], we focus on step 3 and propose an efficient 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.

2.2 Dependency Graph

A dependency graph is a directed graph that represents computational dependencies among variables storing source group elements in the target system. In Figure 1, we show an example of a dependency graph for a program that computes some group operations over Type-I bilinear groups. In the right is a sample program that takes source group elements A, B, D as input and computes C and E via group operations (multiplication and exponentiation), and outputs a result of pairing $e(C, E)$. In the left is a dependency graph that corresponds to the algorithm. Nodes represent the source group elements and edges correspond to group operations. Each input to the pairing operation is represented by a connection to node $\mathsf{P}_{\mathsf{CE}}[b]$ called a pairing node. As the graph only describes relations between group elements via group operations, it does not show the



`Sample(a,` A, B, D`):`
$a \in \mathbb{Z}_p,\ A, B, C, D, E \in \mathbb{G}$

`if` $a = 0$ `then`
  $C := A \cdot B, \quad E := D$
`else`
  $C := D^a, \quad E := D^3$
`endif`
`Output` $e(C, E)$

Fig. 1. An example of a dependency graph for a program in Type-I bilinear groups.

structure of the program like "`if-then-else`" directive or involve non source group elements like $a \in \mathbb{Z}_p$. Operations in the target group are irrelevant either.

There are several types of nodes in a dependency graph. Node types can be considered as attributes attached to the nodes or lists of nodes. We use either way according to the context.

- Pairing nodes ($\mathcal{P}$). They 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.
- Control nodes ($\mathcal{CT}$). These are the ones added to the graph to control the assignment to their parent nodes. A control node has one or more incoming edges but no outgoing edges. By specifying which group to assign to a control node, its parent nodes are also assigned to the same group. For instance, when two variables associated to nodes n and n' are to be compared, a control node is added with incoming edges from n and n'. This results in assigning n and n' to the same group the control node is assigned. The control nodes are used also to implement user specified preferences such as grouping as we shall explain later.
- Regular nodes ($\mathcal{R}$). All nodes other than pairing nodes and control nodes are regular nodes. Regular nodes may have other attributes named as follows.
  - Bottom nodes ($\mathcal{B}$). A regular node is a bottom node if it does not have outgoing edges. This includes a 'pseudo' bottom node that virtually works as a bottom node in a closure.
  - Prohibited nodes ($\mathcal{PH}$). These are nodes that must not be duplicated for some reasons. They are assigned to either of the source groups but the assignment is not fixed in advance. Nodes representing variables as an output of "hash-to-group" function that directly maps to group elements must be a prohibited node. Currently known technology does not allow us to hash an input onto two source group elements in a way that their exponents are unknown but remain in a preliminary fixed relation. Another example of the prohibiting nodes are inputs given to

the target scheme from outside like messages in a signature scheme. They are subject to other building blocks and hence demanding duplicated messages loses generality of the signature scheme. Thus it is generally desirable that messages are considered as prohibited nodes.

From the above classification, we have $V = \mathcal{P} \cup \mathcal{CT} \cup \mathcal{R}$. The nodes that will be assigned to either of the source groups exclusively are called constrained nodes. Precisely, we define constrained nodes $\mathcal{C}$ by $\mathcal{C} := \mathcal{P} \cup \mathcal{CT} \cup \mathcal{B} \cup \mathcal{PH}$.

2.3 Valid Split

It has been shown in [4] 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. Let $\mathsf{Anc}(\Gamma, X)$ denote a subgraph of Γ that consists of X and all paths that reach to X. Let NoDup be a list of nodes representing variables as output of hash-to-group function.

Definition 1 (Valid Split). *Let $\Gamma = (V, E)$ be a dependency graph for $\tilde{\Pi}$. Let $P = (p_1[0], \ldots, p_{n_p}[1]) \subset V$ be pairing nodes. A pair of graphs $\Gamma_0 = (V_0, E_0)$ and $\Gamma_1 = (V_1, E_1)$ is a valid split of Γ with respect to $\mathsf{NoDup} \subseteq V$ if:*

1. *merging Γ_0 and Γ_1 recovers Γ,*
2. *for each $i \in \{0, 1\}$ and every $X \in V_i \setminus P$, the subgraph $\mathsf{Anc}(\Gamma, X)$ is in Γ_i,*
3. *for each $i \in \{1, \ldots, n_p\}$, paring nodes $p_i[0]$ and $p_i[1]$ are separately included in V_0 and V_1, and*
4. *$V_0 \cap V_1 \cap \mathsf{NoDup} = \emptyset$.*

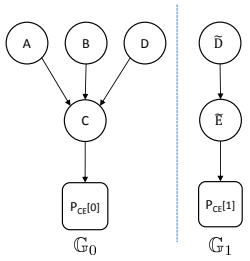
The first condition guarantees that all variables and computations are preserved during conversion. The second condition guarantees that all variables needed to compute a variable belong to the same source group. The third condition guarantees consistency of pairing operations by forcing that every pairing operation takes inputs from $\mathbb{G}_0$ and $\mathbb{G}_1$. The last condition is to conform with the constraint about the hash-to-group functions. In Figure 2, we illustrate a valid split for the dependency graph shown in Figure 1 and the resulting program in Type-III.

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.

3 Finding Optimal Valid Split with IP

3.1 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



`Sample(a,`$A, B, D, \tilde{D}$`):`
$a \in \mathbb{Z}_p$, $A, B, C, D \in \mathbb{G}_0$, $\tilde{D}, \tilde{E} \in \mathbb{G}_1$
`if` $e(G, \tilde{D}) \neq e(D, \tilde{G})$ then err;
`if` $a = 0$ `then`
$C := A \cdot B, \quad \tilde{E} := \tilde{D}$
`else`
$C := D^a, \quad \tilde{E} := \tilde{D}^3$
`endif`
`output` $e(C, \tilde{E})$

Fig. 2. A valid split for the dependency graph in Fig.1, and a converted program.

in the public key elements, or the smallest possible duplication in the instance of assumptions. We show in the following several types of preferences that can be handled in our conversion procedure.

1. **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.
2. **Prohibiting duplication.** By specifying a node as 'prohibited', the node will never be duplicated.
3. **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.)
4. **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.
5. **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.)
6. **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 $m = (m[0], \ldots, m[k])$ with constant k

can be represented by a node referred to by $\mathsf{m}[\mathsf{i}]$. Such a node should have a magnification factor of k. It must be equal or larger than one.

In the next section, we explain how these preferences are incorporated to the objective function and constraints given to Integer Programming.

3.2 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 Section 2.1. It takes as input a dependency graph Γ for source group $\mathbb{G}$ of Type-I scheme, and outputs two dependency graphs Γ_0 and Γ_1 for $\mathbb{G}_0$ and $\mathbb{G}_1$, respectively, of the converted Type-III scheme.

IPConv consists of the following stages. Details are given after the overview.

1. **Preprocessing on the graph.** The input dependency graph is modified to implement some user-specified preferences. The output of this stage is the modified dependency graph and a list of constrained nodes.
2. **Establishing the objective function.** Binary variables that represent (non-)membership in each source group are placed on constraint nodes. They must satisfy relations for consistency and for user's preferences. Sanity checking is done to assure the existence of a solution that conforms to the constraints. Then the objective function over the variables is established.
3. **Running Integer Programming.** Run 0-1 Integer Programming for finding an assignment to the variables that minimizes the objective function subject to the constraints.
4. **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.

Preprocessing on the graph. First of all, user preference in prohibiting duplication is dealt simply by including the specified nodes to the list of prohibited nodes $\mathcal{PH}$. A specific assignment to a specific node, say n, is handled by adding a new control node, c, and edge (n, c) to the graph. 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 of nodes $n_1, \ldots, n_k$ is handled in the same manner by adding a new control node c, and edges $(n_1, c), \ldots, (n_k, c)$ to the graph. This step outputs the updated graph with attributes that identifies the constraint nodes.

Establishing the objective function By $|\mathbb{G}_b|$ we denote number of bits necessary to represent arbitrary element in $\mathbb{G}_b$. Let $\Gamma = (V, E)$ be a dependency graph. By $dec(n)$ for node $n \in V$, we denote all descendant nodes of n in Γ, i.e., all nodes that can be reached from n. For every node $n \in \mathcal{C}$ we associate a binary variable

x_{nb} for $b = 0, 1$ that [5]:

$$x_{nb} = \begin{cases} 1 & (n \in \mathbb{G}_b) \\ 0 & (n \notin \mathbb{G}_b) \end{cases} \tag{1}$$

Let x denote the set of all those variables; $x := \{x_{nb} \mid n \in V, b \in \{0,1\}\}$. Let $\Phi(x)$ be a collection of relations on variables in x needed for consistency of assignments. Since every constrained node should be exclusively assigned to either of the source groups, relation $x_{n0} + x_{n1} = 1$ for all $n \in \mathcal{C}$ are included in $\Phi(x)$. For every pair of pairing nodes, say n and n', they must get exclusive assignment to either of the source groups. Thus it must hold that $x_{n0} + x_{n'1} = 1$ and $x_{n1} + x_{n'0} = 1$. The same relation should hold for every pair of nodes specified to have exclusive assignment. For every pair of nodes n and n' in $\mathcal{C}$, if $n' \in dec(n)$, then $x_{n0} - x_{n'0} = 0$ and $x_{n1} - x_{n'1} = 0$ must be included in $\Phi(x)$ as they have to receive the same assignment. For a control node n for specifying assignment $\mathbb{G}_b$ to a regular node, relation $x_{nb} = 1$ is included in $\Phi(x)$. Control nodes for prohibiting duplication and grouping need no further treatment since they are already treated as a constrained nodes.

We apply a sanity checking that the constraints in $\Phi(x)$ are satisfiable. Observe that relations in $\Phi(x)$ can be seen as a system of equations over $GF(2)$. Then $\Phi(x)$ is satisfiable if and only if the system of equations is not overdetermined. Such a checking can be done in $O(|\mathcal{C}|^3)$ binary operations. Despite the asymptotic growth rate, the sanity check indeed finishes instantly even for large inputs and in fact negligible compared to the main workload shown in the next. By $\Phi(x) = 1$, we denote that constraints in $\Phi(x)$ are satisfiable. We denote $\Phi(x) = 0$, otherwise.

We then establish the objective function, $\mathcal{E}$, and constraints Ψ. Define a function $n \overset{?}{\in} \mathbb{G}_b$ for $n \in V$ and $b = 0, 1$ by

$$n \overset{?}{\in} \mathbb{G}_b = \begin{cases} 1 & (n \in \mathbb{G}_b) \\ 0 & (n \notin \mathbb{G}_b) \end{cases} . \tag{2}$$

For every node $n \in \mathcal{C}$, it is clear, by definition, that

$$(n \overset{?}{\in} \mathbb{G}_b) = x_{nb}. \tag{3}$$

For regular nodes (as defined in Section 2.2) other than those included in $\mathcal{C}$, i.e., $n \in V \setminus \mathcal{C} = \mathcal{R} \setminus (\mathcal{B} \cup \mathcal{PH})$, observe that $n \in \mathbb{G}_b$ holds if there is a constrained node in the descendant of n that is assigned to $\mathbb{G}_b$. Let $\mathcal{C}_n$ denote $\mathcal{C} \cap dec(n)$ that are the constrained nodes reached from node n. Then we have

$$(n \overset{?}{\in} \mathbb{G}_b) = \bigvee_{d \in \mathcal{C}_n} x_{db} = \neg \bigwedge_{d \in \mathcal{C}_n} \neg x_{db} = 1 - \prod_{d \in \mathcal{C}_n} (1 - x_{db}). \tag{4}$$

[5] Instead, we can associate a single variable x_n set to b if the node is in $\mathbb{G}_b$ as done in our proof of concept implementation. It slightly reduces the number of relations, but here we choose x_{nb} for comprehensible explanation.

We now use a well known lemma [20] to remove the higher-order term in the above formula.

Lemma 2. *For binary variables $x_1, \ldots, x_k$ and y, relation*

$$\prod_{i=1}^{k} x_i = y \tag{5}$$

holds if and only if the following relations hold:

$$k - 1 - \sum_{i=1}^{k} x_i + y \geq 0 \quad \textit{and} \quad x_i - y \geq 0 \textit{ for all } i = 1, \ldots, k. \tag{6}$$

With this trick, we write (4) using a new variable, y_{nb}, as

$$(n \stackrel{?}{\in} \mathbb{G}_b) = 1 - \prod_{d \in \mathcal{C}_n} (1 - x_{db}) = y_{nb} \tag{7}$$

and put constraints

$$\sum_{d \in \mathcal{C}_n} x_{db} - y_{nb} \geq 0, \quad \text{and} \quad y_{nb} - x_{db} \geq 0 \quad \text{for all } d \in \mathcal{C}_n. \tag{8}$$

Define function $\mathit{eval}(n)$ for every regular node $n \in \mathcal{R}$ by

$$\mathit{eval}(n) := \sum_{b \in \{0,1\}} w_{nb} \cdot (n \stackrel{?}{\in} \mathbb{G}_b) \tag{9}$$

where w_{nb} is a positive real number associated to node n. Also define

$$\begin{aligned} &\mathit{eval\_max}(n) := w_{n0} + w_{n1}, \\ &\mathit{eval\_2nd}(n) := \begin{cases} w_{n0} + w_{n1} & (\text{if } w_{n0} = w_{n1}), \\ \max(w_{n0}, w_{n1}) & (\text{if } w_{n0} \neq w_{n1}), \end{cases} \\ &\mathit{eval\_min}(n) := \min(w_{n0}, w_{n1}), \end{aligned} \tag{10}$$

which means the maximum, second-minimum, and minimum value $\mathit{eval}(n)$ can take respectively.

Parameter w_{nb} represents the cost of having node n in $\mathbb{G}_b$ and the concrete value for the parameter is defined according to one's metrics. In this work, we set $w_{n0} := 1$ and $w_{n1} := 2$ according to the typical ratio of bit length of elements in $\mathbb{G}_0$ and $\mathbb{G}_1$. When a magnification factor k_n is defined, they are multiplied by k_n. The idea for the setting is that we seek for a conversion requiring minimum space for storing objects specified in the priority.

We then compose an objective function according to the given priority $(I_1, \ldots, I_k)$. Let I_{k+1} be regular nodes that do not appear in the priority, i.e., $I_{k+1} := \mathcal{R} \setminus (\bigcup_{i=1}^{k} I_i)$. For each node n, let

$$\Delta_n := \mathit{eval\_max}(n) - \mathit{eval\_min}(n) \tag{11}$$

which means the relative impact of duplicating n in the priority of n. And for each I_i, let

$$\Xi_i := \min_{n \in I_i} \{ eval\_2nd(n) - eval\_min(n) \}, \tag{12}$$

that is the relative minimum impact in the I_i of the assigning one single node to the larger group. For every I_i, we define priority factor ρ_i as

$$\rho_i \cdot \Xi_i > \sum_{j=i+1}^{k+1} \rho_j \sum_{n \in I_j} \Delta_n. \tag{13}$$

This means that assigning one single node to the larger group in any level of priority has more significant impact than duplicating all nodes in all lower levels of priority. For example, it is enough to let $\rho_{k+1} := 1$ and

$$\rho_i := 1 + \frac{1}{\Xi_i} \sum_{j=i+1}^{k+1} \rho_j \sum_{n \in I_j} \Delta_n \tag{14}$$

for $i = k$ down to 1. Let v denote all variables x_{nb} and y_{nb}. We define the target function $\mathcal{E}(v)$ by

$$\mathcal{E}(v) := \sum_{i=1}^{k+1} \rho_i \sum_{n \in I_i} eval(n) - eval\_min(n), \tag{15}$$

which is linear over variables in v. By $\Psi(v)$ we denote associated constraints that include all relations in $\Phi(x)$ and relations in (8). By $\Psi(v) = 1$ we denote that all constrains in $\Psi(v)$ are fulfilled. Otherwise $\Psi(v) = 0$.

Running 0-1 Integer Programming. Now we run 0-1 IP solver by giving $\mathcal{E}(v)$ and $\Psi(v)$ as input. The output is an assignment to v that minimizes $\mathcal{E}(v)$ subject to $\Psi(v) = 1$. Note that the IP solver, SCIP, used in our implementation recognizes unsolvable inputs by nature as a part of its functionality. It makes the sanity check in the previous stage redundant. Nevertheless, the sanity check in the earlier stage is useful for debugging.

Composing the final split. Given the assignment to v one can compute $(n \stackrel{?}{\in} \mathbb{G}_b)$ for all $n \in V$, and construct two dependency graphs for $\mathbb{G}_0$ and $\mathbb{G}_1$ in such a way that every edge (n, n') in the input dependency graph is included in at least one of the resulting graphs that include the destination n'. Since the assignment conforms to all given constraints, this yields a valid split. The split is optimal in the sense that it minimizes the target value $\mathcal{E}(v)$ that measures one's preferences. This completes the description of our IPConv method.

3.3 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

Target Scheme	Graph Size		Processing Time	Notes
	#vertices	#pairings		
Waters' DSE [41]	95	13	146 ms	(4639 ms)
BBS HIBE [14]	283	56	262 ms	(15667 ms)
BlindAutoSIG [3]	339	116	142 ms	-
AHO[3]+GSZK [27]	597	222	463 ms	-
Trace. Group Enc.[38]	1604	588	6306 ms	-

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.

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.

4 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 [5] (non-commercial) and GUROBI [28] (commercial).

4.1 Processing Time for Real Schemes

Small-scale schemes. In the first two rows of Table 1, we show the processing time of IPConv for converting Boneh-Boyen HIBE [14] with $\ell = 9$ hierarchy, and

Waters' Dual-system encryption [41]. 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 AutoGroup+. According to [6], 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.

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 [3], 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 Section 5.

Blind Automorphic Signature Scheme: The second experiment is for the automorphic blind signature scheme from [3]. 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.

Traceable Group Encryption: Our last experiment is for a traceable group encryption scheme from [38] that is more intricate involving several building blocks such as a tag-based encryption [32], 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.

4.2 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.

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 $i \in \{1, \ldots, N\}$. Pairing nodes $p_{ij}[0]$ and $p_{ij}[1]$ represent a pairing with nodes i and j as input.

[Random Dependency Graph Generation]

1. Generate regular nodes $1, \ldots, N$.
2. For every regular node $i \in \{1, \ldots, N\}$, select $k' \leftarrow \{1, \ldots, k\}$ and repeat the following k' times:
   - Select $j \leftarrow \{1, \ldots, i-1\}$.
   - Generate an edge (j, 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 $p_{ij}[0]$ and $p_{ij}[1]$ and edges $(i, p_{ij}[0])$ and $(j, p_{ij}[1])$.

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.

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. Figure 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 Figure 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 Figure 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.

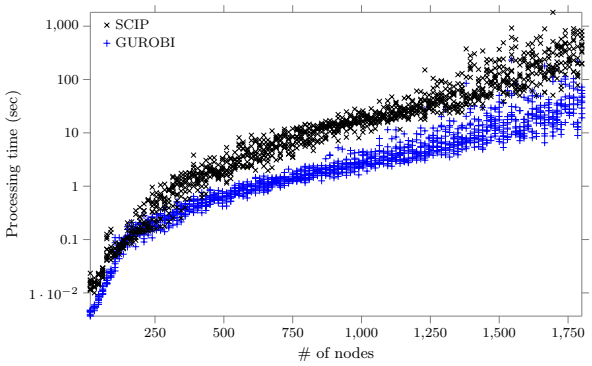


Fig. 3. Processing time in the semi-log scale for random dependency graphs.

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 in the previous experiment. 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 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 Figure 5 with Figure 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

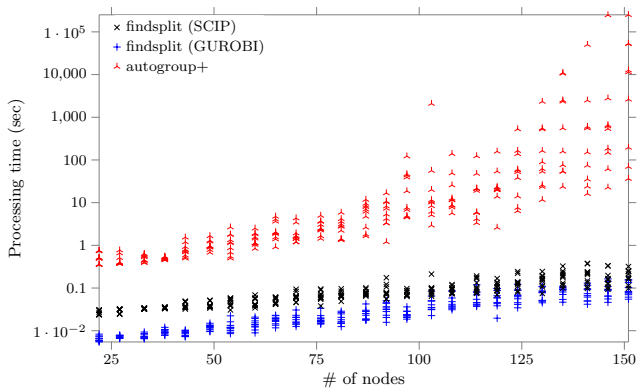


Fig. 4. Comparison between IPConv and AutoGroup+ regarding stability of processing time.

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.

5 Using Conversion in Cryptographic Design

In this section we show how conversion plays the role in designing cryptographic schemes. We begin by introducing a new fine-tuned construction of GS Zero-knowledge proofs in Type-I setting in Section 5.1. It is followed by an example that combines the GS ZK with the AHO signature scheme in Section 5.2. We then show another example in Section 5.3 that demonstrates conversion of an automorphic blind signature scheme designed originally in Type-III.

5.1 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 [27], it is done by proving a relation represented by a general multi-scalar multiplication equation (MSE). We

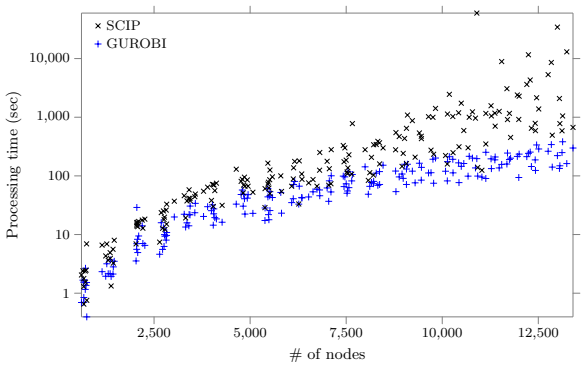


Fig. 5. Processing time in the semi-log scale for cluster dependency graphs.

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

$$[X]^1 \cdot A^{-[1_{\mathbb{Z}_p}]} = 1_{\mathbb{G}}. \tag{16}$$

Zero-knowledge simulation with a hiding CRS is done as follows. The simulator opens the default commitment $[1_{\mathbb{Z}_p}; 0, 0]$ as $[0_{\mathbb{Z}_p}; r'_1, r'_2]$ by using the trapdoor. It then sets $X = 1_{\mathbb{G}}$ and computes $[X]$ which is perfectly indistinguishable from $[A]$. With respect to those commitments relation (16) is read as $[1_{\mathbb{G}}]^1 \cdot A^{-[0_{\mathbb{Z}_p}]} = 1_{\mathbb{G}}$, which is true. Thus the simulator can generate a proof following the legitimate procedure.

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

$$e([X], G)\, e(A^{-1}, [G^1]) = 1_{\mathbb{G}_T}. \tag{17}$$

instead of (16). Since we are considering the DLIN-based instantiation for now, (17) is a *linear* PPE that costs only 3 group elements whereas proof of (16) requires 9 elements.

Zero-knowledge simulation with a hiding CRS is done by first equivocating $[G^1; 0, 0, 0]$ into $[G^0; s_1, s_2, s_3]$ using the trapdoor. Then, by setting $X = 1_{\mathbb{G}}$, relation (17) is $e([1_{\mathbb{G}}], G)\, e(A^{-1}, [G^0]) = 1_{\mathbb{G}_T}$, which is true. Thus the zero-knowledge simulator can prove it using the witness.

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 [1]. While the security is guaranteed by the conversion framework of [4], 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 $\mathbb{G}_0$). Below,we present the resulting proof system in detail. It is particularly important to see that A and $[X]$ in (17) are in the same group without duplicating A. Full details are presented in the following.

To cope with the original description of the Groth-Sahai proof system, we switch to additive notation in the rest of this section. Let us define some notations used in the following. Let $(p, \mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_T, e, G, \tilde{G})$ be an asymmetric bilinear group with $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_T$. For $X, Y \in \mathbb{G}_b^n$, operation $X + Y$ denotes the result of element-wise group operations in $\mathbb{G}_b$. By $\mathrm{Mat}_{n\times m}$, we denote all matrices of size $n \times m$ over $\mathbb{Z}_p$. Let $\tilde{F}$ be a function that

$$\tilde{F}\left(\begin{pmatrix} X_1 \\ X_2 \\ X_3 \end{pmatrix}, \begin{pmatrix} Y_1 \\ Y_2 \\ Y_3 \end{pmatrix}\right) := \begin{pmatrix} \hat{e}(X_1, Y_1) & \hat{e}(X_1, Y_2) & \hat{e}(X_1, Y_3) \\ \hat{e}(X_2, Y_1) & \hat{e}(X_2, Y_2) & \hat{e}(X_2, Y_3) \\ \hat{e}(X_3, Y_1) & \hat{e}(X_3, Y_2) & \hat{e}(X_3, Y_3) \end{pmatrix} \tag{18}$$

where

$$\hat{e}(X, Y) = \begin{cases} e(X, Y) & (X \in \mathbb{G}_0 \wedge Y \in \mathbb{G}_1) \\ e(Y, X) & (Y \in \mathbb{G}_0 \wedge X \in \mathbb{G}_1)\ . \\ \bot & (\text{otherwise}) \end{cases} \tag{19}$$

By $X \mathbin{\tilde{\bullet}} Y$, we denote $\tilde{F}(X, Y)$. For vectors $\boldsymbol{X} = (X^{(1)}, \ldots, X^{(n)})$ and $\boldsymbol{Y} = (Y^{(1)}, \ldots, Y^{(n)})$, we denote $\boldsymbol{X} \mathbin{\tilde{\bullet}} \boldsymbol{Y}$ for shorthand of $\sum_{i=1}^{n} \left(X^{(i)} \mathbin{\tilde{\bullet}} Y^{(i)}\right)$.

It is important to see that computation in $\tilde{F}$ and $\tilde{\bullet}$ can be carried out as long as X and Y are taken exclusively from $\mathbb{G}_0$ and $\mathbb{G}_1$. We use convention that large case letters like A represent elements in $\mathbb{G}_0$, and those with tilde like $\tilde{A}$ represent elements in $\mathbb{G}_1$.

Now we are ready to describe how to prove that $[X]$ is a correct commitment of $A \in \mathbb{G}_0$ with the GS proof system instantiated in Type-III setting based on XDLIN.

[CRS Generation]

Choose $\alpha, \beta, \xi_1, \xi_2 \leftarrow \mathbb{Z}_p$ and compute $G_1 := G^\alpha$, $G_2 := G^\beta$, $\boldsymbol{u}_1 := (G_1, \mathcal{O}, G)$, $\boldsymbol{u}_2 := (\mathcal{O}, G_2, G)$, and

$$\boldsymbol{u}_3 = (G_{31}, G_{32}, G_{33}) := \xi_1 \cdot \boldsymbol{u}_1 + \xi_2 \cdot \boldsymbol{u}_2 + (\mathcal{O}, \mathcal{O}, -\gamma \cdot G) \tag{20}$$

$$= (\xi_1 \cdot G_1, \xi_2 \cdot G_2, (\xi_1 + \xi_2 - \gamma) \cdot G) \tag{21}$$

where $\gamma = 0$ for binding and $\gamma = 1$ for hiding mode. Compute $\tilde{\boldsymbol{u}}_1$, $\tilde{\boldsymbol{u}}_2$, and $\tilde{\boldsymbol{u}}_3$ exactly in the same way using the same randomness $(\alpha, \beta, \xi_1, \xi_2)$ but with generator $\tilde{G}$ instead of G. Then CRS is $(\boldsymbol{u}, \tilde{\boldsymbol{u}})$ where

$$\boldsymbol{u} := \begin{pmatrix} \boldsymbol{u}_1 \\ \boldsymbol{u}_2 \\ \boldsymbol{u}_3 \end{pmatrix}, \quad \text{and} \quad \tilde{\boldsymbol{u}} := \begin{pmatrix} \tilde{\boldsymbol{u}}_1 \\ \tilde{\boldsymbol{u}}_2 \\ \tilde{\boldsymbol{u}}_3 \end{pmatrix}. \tag{22}$$

[Prover Algorithm]

Given $A \in \mathbb{G}_0$ as a witness, first commit to $X := A$ using randomness $\mathcal{S}_X := (s_{1,X}, s_{2,X}, s_{3,X}) \leftarrow \mathrm{Mat}_{1\times 3}$ as

$$[X] := (\mathcal{O}, \mathcal{O}, X) + \mathcal{S}_X \, \boldsymbol{u} = (C_{1,X}, C_{2,X}, C_{3,X}). \tag{23}$$

Set $(s_{1,\tilde{G}}, s_{2,\tilde{G}}, s_{3,\tilde{G}}) = (0,0,0) \in \mathbb{Z}_p^3$. Compute proof $\theta_{(17)}$ as

$$\theta_{(17)} := \begin{pmatrix} s_{1,X} & s_{1,\tilde{G}} \\ s_{2,X} & s_{2,\tilde{G}} \\ s_{3,X} & s_{3,\tilde{G}} \end{pmatrix} \begin{pmatrix} \mathcal{O} & \mathcal{O} & G \\ \mathcal{O} & \mathcal{O} & A^{-1} \end{pmatrix} = \begin{pmatrix} \mathcal{O} & \mathcal{O} & \theta_{1,(17)} \\ \mathcal{O} & \mathcal{O} & \theta_{2,(17)} \\ \mathcal{O} & \mathcal{O} & \theta_{3,(17)} \end{pmatrix}. \tag{24}$$

Output $[X]$ and $\theta_{(17)}$ as a proof. Dropping trivial elements, they consist of 6 group elements in $\mathbb{G}_0$.

[Verifier Algorithm]

Compute the default commitment of $\tilde{G}$ as

$$[\tilde{G}] := (\mathcal{O}, \mathcal{O}, \tilde{G}) + (0,0,0)\, \boldsymbol{u} = (\mathcal{O}, \mathcal{O}, \tilde{G}) = (\tilde{C}_{1,\tilde{G}}, \tilde{C}_{2,\tilde{G}}, \tilde{C}_{3,\tilde{G}}). \tag{25}$$

Then output 1 if the following holds. Output 0, otherwise.

$$\begin{pmatrix} C_{1,X} \\ C_{2,X} \\ C_{3,X} \end{pmatrix} \tilde{\bullet} \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ \tilde{G} \end{pmatrix} + \begin{pmatrix} \tilde{C}_{1,\tilde{G}} \\ \tilde{C}_{2,\tilde{G}} \\ \tilde{C}_{3,\tilde{G}} \end{pmatrix} \tilde{\bullet} \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ A^{-1} \end{pmatrix} = (\tilde{\boldsymbol{u}})^\top \tilde{\bullet} (\theta_{(17)})^\top \tag{26}$$

[Zero-Knowledge Simulation]

Generate CRS with $\gamma = 1$ (hiding mode). Given $A \in \mathbb{G}_0$ and trapdoor $(\alpha, \beta, \xi_1, \xi_2)$, set $(s_{1,\tilde{G}}, s_{2,\tilde{G}}, s_{3,\tilde{G}}) := (\xi_1, \xi_2, -1)$, which equivocate the default commitment

$[\tilde{G}^1; 0, 0, 0]$ to $[\tilde{G}^0; \xi_1, \xi_2, -1]$. Also set $X := G^0$. Then follow the prover algorithm using these witnesses.

Direct Fine-Tuning in Type-III. The above idea can be applied to SXDH-based GS-proofs in Type-III as well. However, it is limited to the case where A is duplicated. The reason is that, relation (17) must be proved as one-side PPE in Type-III where involved commitments appear only in one side of the pairing operations. Namely, (17) has to be rewritten as

$$e([X], \tilde{G})\, e([G^1], \tilde{A}^{-1}) = 1_{\mathbb{G}_T}. \tag{27}$$

Thus we need $A \in \mathbb{G}_0$ to compute $[X]$ and additionally need $\tilde{A} \in \mathbb{G}_1$ to verify the proof.

If duplicating A is not acceptable, we have to get back to the original construction that proves MSE (16) instead. It costs 6 group elements. Note that it is also possible to prove (17) as a two-side PPE but it costs 8 group elements.

5.2 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

$$e(A_0, [A_1])\; e(A_2, [A_3]) = e(G_z, [Z])\, e(G_r, [R])\, e(S', [T']) \prod_{i=1}^{n} e(G_i, [M_i]) \tag{28}$$

$$e(B_0, [B_1])\; e(B_2, [B_3]) = e(H_z, [Z])\, e(H_u, [U])\, e(V', [W']) \prod_{i=1}^{n} e(H_i, [M_i]) \tag{29}$$

hold with respect to committed variables in the brackets. Additionally, relation (17) for every public value $X \in \{A_1, A_3, B_1, B_3, M_1, \ldots, M_n\}$ is proved by using the technique in Section 5.1 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.

Conversion: By converting the above scheme we obtain a scheme in Type-III. Details for the proof part are presented in Appendix A. 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 (17) are assigned to the same group without duplicating X while proving (17) as a linear PPE. This approach is the most efficient in the proof size since most of commitments and proofs can be allocated in $\mathbb{G}_0$.

Construction	Duplicated Object	Proof Size $\mathbb{G}_0$	Proof Size $\mathbb{G}_1$	Proof Size in bits	# of Pairings naive	# of Pairings batched
Conversion	crs	$6n+39$	6	$(6n+51)\lambda$	$18n+90$	$2n+20$
Direct (1)	msg	$2n+18$	$3n+12$	$(8n+42)\lambda$	$12n+60$	$2n+17$
Direct (2)	pk	$4n+26$	$4n+16$	$(12n+58)\lambda$	$20n+84$	$n+23$
Direct (3)	-	$4n+26$	$4n+20$	$(12n+66)\lambda$	$22n+100$	$2n+22$

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 is for GS commitments and proofs. Column "naive" counts the number of pairings literally in the verification equations, and "batched" counts the number of pairings in batch verification.

Direct instantiation 1 (with duplicated messages): Next we consider instantiating the GS-proofs directly over Type-III groups based on the SXDH assumption. As observed in Section 5.1, 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 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 (16) as shown in the beginning of Section 5.1. 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, there is no clear winner. The scheme obtained by conversion yields the most compact proofs for messages of $n > 5$. But for short and duplicable messages, direct construction produces more compact proofs. Regarding the computational workload, when batch verification is taken into account, there is not much difference for small n no matter what approach is taken. But for large n, direct instantiation in Type-III with duplicated public-key is more advantageous.

5.3 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.

In the automorphic blind signature scheme in [3], a blind signature is a GS-proof for one's possession of a correct (plain) automorphic signature on a clear message. A plain automorphic signature consists of five group elements $\sigma := (A, B, \tilde{D}, R, \tilde{S})$ verified by PPEs:

$$e(A, \tilde{Y} \cdot \tilde{D}) = e(K \cdot M, \tilde{G})\, e(T, \tilde{S}), \tag{30}$$

$$e(B, \tilde{G}) = e(F, \tilde{D}), \quad e(R, \tilde{G}) = e(G, \tilde{S}). \tag{31}$$

An automorphic blind signature is a GS-proof of (30) and (31) with $(A, B, \tilde{D}, R, \tilde{S})$ as a witness. The security of the original construction bases on SXDH assumption and Asymmetric Double Hidden Strong DH Assumption (ADHDH) [3].

To incorporate the scheme into the conversion framework, we need to build a dependency graph in such a manner that the original scheme is included in a possible solution of conversion. First, a special treatment is needed to the nodes representing X and $\tilde{Y}$ that are already in the duplicated form since they should not be individually duplicated by conversion. We set dependency $Y \to X$, and prohibit duplication of X. In this way, Y will be duplicated so that X is assigned to $\mathbb{G}_b$ and $\tilde{Y}$ is assigned to $\mathbb{G}_{1-b}$. Such a treatment is applied to (M, N) and (R, S) as well. Second, we need to build a dependency graph for the assumptions. Since ADHDH is known to hold even in the Type-I generic bilinear group model, we simply ignore the distinction of $\mathbb{G}_0$ and $\mathbb{G}_1$. For SXDH, we prohibit duplication of any variable in its instance and use grouping of variables so that they are allocated to the same group. In this way, the assumption remains valid when converted back to Type-III. Finally, the GS-proof part is described by using the DLIN-based instantiation of GS-proofs. They are witness indistinguishable proofs and we do not rely on the fine-tuning as in the previous case.

After conversion, the resulting scheme in Type-III is secure based on SXDH, ADHDH with duplicated $\tilde{D}$, and XDLIN assumptions. We present details of the converted scheme for the part of generating and verifying a blind signature in Appendix B. Table 3 summarizes the performance in comparison with the original construction. The converted scheme saves 28% of blind signature in bits and equal or slightly better in verification workload.

6 Conclusion

We have proposed an efficient type conversion method based on 0-1 Integer Programming. It is shown how to represent several constraints into a system of linear binary equations so that a 0-1 IP solver can find an optimal solution that

Construction	Duplicated Objects	Size of Blind Sig. $\mathbb{G}_0$	$\mathbb{G}_1$	in bits	# of Pairings naive	batched
Conversion	crs, $\tilde{D}$	24	6	$36\,\lambda$	64	13
Original[3]	-	18	16	$50\,\lambda$	68	13

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 [3]. 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.

meets the constraints. The performance and scalability are demonstrated over real schemes and randomly generated samples.

Usefulness of the conversion-aided design approach is demonstrated by examples that outputs more compact GS-proofs than those manufactured directly in Type-III setting. A fine-tuning technique that improves the performance of converted GS-proofs is introduced.

Nevertheless, results in this paper can be seen as a step toward realizing automated modular design of cryptographic protocols. Depending on the target schemes, direct instantiation in Type-III based on SXDH can yield better results. It is in fact another optimization issue that machines can help to find a globally optimal solution. We include it as an interesting research and engineering target in our future plan.

Finally, a proof-of-concept implementation with source codes and data files for experiments in Section 5 are available from the authors for review. Open source development is certainly in our future plan.

7 Acknowledgements

The authors thank Susan Hohenberger Waters and co-authors of [7, 6] for their help to understand AutoGroup. We also thank to Takeya Tango for an alternative sanity checking method. Special thanks to the developers of SCIP [5] for their quality software.

References

1. M. Abe, M. Chase, B. David, M. Kohlweiss, R. Nishimaki, and M. Ohkubo. Constant-size structure-preserving signatures: Generic constructions and simple assumptions. In *ASIACRYPT 2012*, vol.7658 of *LNCS*, pp.4–24, 2012.
2. M. Abe, B. David, M. Kohlweiss, R. Nishimaki, and M. Ohkubo. Tagged one-time signatures: Tight security and optimal tag size. In *PKC 2013*, vol.7778 of *LNCS*, pp.312–331, 2013.

3. M. Abe, G. Fuchsbauer, J. Groth, K. Haralambiev, and M. Ohkubo. Structure-preserving signatures and commitments to group elements. *J. Cryptology*, 29(2):363–421, 2016.
4. M. Abe, J. Groth, M. Ohkubo, and T. Tango. Converting cryptographic schemes from symmetric to asymmetric bilinear groups. In *CRYPTO 2014*, vol.8616 of *LNCS*, pp.241–260, 2014.
5. T. Achterberg. CIP: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009. `http://mpc.zib.de/index.php/MPC/article/view/4`.
6. J. A. Akinyele, C. Garman, and S. Hohenberger. Automating fast and secure translations from type-i to type-iii pairing schemes. In *ACM CCS 2015*, pp.1370–1381, ACM, 2015.
7. J. A. Akinyele, M. Green, and S. Hohenberger. Using SMT solvers to automate design tasks for encryption and signature schemes. In *ACM CCS 2013*, pp.399–410, ACM, 2013.
8. M. Backes, D. Fiore, and R. M. Reischuk. Verifiable delegation of computation on outsourced data. In *ACM CCS 2013*, pp.863–874, ACM, 2013.
9. R. Barbulescu, P. Gaudry, A. Joux, and E. Thomé. A quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic. IACR ePrint Archive, Report 2013/400, 2013. `http://eprint.iacr.org`.
10. G. Barthe, E. Fagerholm, D. Fiore, J. C. Mitchell, A. Scedrov, and B. Schmidt. Automated analysis of cryptographic assumptions in generic group models. In *CRYPTO 2014*, vol 8616. of *LNCS*, pp.95–112, 2014.
11. 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 *PKC 2015*, vol 9020. of *LNCS*, pp.355–376, 2015.
12. B. Blanchet. Cryptoverif: A computationally sound mechanized prover for cryptographic protocols. In Dagstuhl seminar Formal Protocol Verification Applied, 10 2007.
13. O. Blazy, G. Fuchsbauer, M. Izabachène, A. Jambert, H. Sibert, and D. Vergnaud. Batch Groth-Sahai. In *ACNS 2010*, vol.6123 of *LNCS*, pp.218–235, 2010.
14. D. Boneh and X. Boyen. Efficient selective-ID secure identity based encryption. In *EUROCRYPT 2004*, vol.3027 of *LNCS*, pp.223–238, 2004.
15. D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In *CRYPTO 2004*, vol.3152 of *LNCS*, pp.41–55, 2004.
16. D. Boneh and H. Shacham. Group signatures with verifier-local revocation. In *ACM CCS 2004*, pp.168–177. ACM, 2004.
17. S. Chatterjee, D. Hankerson, E. Knapp, and A. Menezes. Comparing two pairing-based aggregate signature schemes. *Des. Codes Cryptography*, 55(2-3):141–167, 2010.
18. S. Chatterjee and A. Menezes. On cryptographic protocols employing asymmetric pairings - the role of psi revisited. IACR ePrint Archive, Report 2009/480, 2009. `http://eprint.iacr.org`.
19. S. Chatterjee and A. Menezes. On cryptographic protocols employing asymmetric pairings - the role of revisited. *Discrete Applied Mathematics*, 159(13):1311–1322, 2011.
20. D.-S. Chen, R. G. Batson, and Y. Dang. *Applied Integer Programming: Modeling and Solution*. WILEY, 2009.
21. L. De Moura and N. Bjørner. Z3: An efficient smt solver. In *TACAS 2008, ETAPS 2008*, vol.4963 of *LNCS*, pp.337–340, 2008.

22. A. Escala and J. Groth. Fine-tuning Groth-Sahai proofs. In *PKC 2014*, vol.8383 of *LNCS*, pp.630–649, 2014.
23. A. Escala, G. Herold, E. Kiltz, C. Ràfols, and J. L. Villar. An algebraic framework for Diffie-Hellman assumptions. In *CRYPTO 2013*, vol.8043 of *LNCS*, pp.129–147, 2013.
24. G. Gamrath and M. E. Lübbecke. Experiments with a generic Dantzig-Wolfe decomposition for integer programs. In *SEA 2010*, vol.6049 of *LNCS*, pp.239–252, 2010.
25. E. Ghadafi, N. P. Smart, and B. Warinschi. Groth-Sahai proofs revisited. In *PKC 2010*, vol.6056 of *LNCS*, pp.177–192, 2010.
26. 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 $\mathrm{f}_{2^{1971}}$. IACR ePrint Archive, Report 2013/074, 2013. `http://eprint.iacr.org`.
27. J. Groth and A. Sahai. Efficient noninteractive proof systems for bilinear groups. *SIAM J. Comput.*, 41(5):1193–1232, 2012.
28. Gurobi Optimization, Inc. Gurobi optimizer reference manual. `https://www.gurobi.com/documentation/6.5/refman.pdf`. `http://www.gurobi.com/`.
29. G. Herold, J. Hesse, D. Hofheinz, C. Ràfols, and A. Rupp. Polynomial spaces: A new framework for composite-to-prime-order transformations. In *CRYPTO 2014*, vol.8616 of *LNCS*, pp.261–279, 2014.
30. A. Joux. Faster index calculus for the medium prime case application to 1175-bit and 1425-bit finite fields. In *EUROCRYPT 2013*, vol.7881 of *LNCS*, pp.177–193, 2013.
31. A. Joux. A new index calculus algorithm with complexity l(1/4+o(1)) in very small characteristic. IACR ePrint Archive, Report 2013/095, 2013. `http://eprint.iacr.org`.
32. E. Kiltz. Chosen-ciphertext security from tag-based encryption. In *TCC 2006*, vol.3876 of *LNCS*, pp.581–600, 2006.
33. T. Koch. *Rapid Mathematical Prototyping*. PhD thesis, Technische Universität Berlin, 2004.
34. LINDO Systems. LINDO. `http://www.lindo.com/`.
35. M. Melnick. LiPS. `http://lipside.sourceforge.net/`.
36. B. Libert and M. Joye. Group signatures with message-dependent opening in the standard model. In *CT-RSA 2014*, vol.8366 of *LNCS*, pp.286–306, 2014.
37. B. Libert, M. Joye, M. Yung, and T. Peters. Secure efficient history-hiding append-only signatures in the standard model. In *PKC 2015*, vol.9020 of *LNCS*, pp.450–473, 2015.
38. B. Libert, M. Yung, M. Joye, and T. Peters. Traceable group encryption. In *PKC 2014*, vol.8383 of *LNCS*, pp.592–610, 2014.
39. N. P. Smart and F. Vercauteren. On computable isomorphisms in efficient asymmetric pairing-based systems. *Discrete Applied Mathematics*, 155(4):538–547, 2007.
40. B. Waters. Efficient identity-based encryption without random oracles. In *EUROCRYPT 2005*, vol.3494 of *LNCS*, pp.114–127, 2005.
41. B. Waters. Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In *CRYPTO 2009*, vol.5677 of *LNCS*, pp.619–636, 2009.

A 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 $\boldsymbol{u} \in \mathbb{G}_0^3$ and $\tilde{\boldsymbol{u}} \in \tilde{\mathbb{G}}_1^3$ are generated in exactly the same manner as described in Section 5.1. The relations to prove are PPEs (28), (29), and (17) re-numbered as follows.

$$\hat{e}(\tilde{A}_0, [A_1])\, \hat{e}(\tilde{A}_2, [A_3]) = \hat{e}(\tilde{G}_z, [Z])\, \hat{e}(\tilde{G}_r, [R])\, \hat{e}(\tilde{S}', [T']) \prod_{i=1}^{n} \hat{e}(\tilde{G}_i, [M_i]) \tag{32}$$

$$\hat{e}(\tilde{B}_0, [B_1])\, \hat{e}(\tilde{B}_2, [B_3]) = \hat{e}(\tilde{H}_z, [Z])\, \hat{e}(\tilde{H}_u, [U])\, \hat{e}(\tilde{V}', [W']) \prod_{i=1}^{n} \hat{e}(\tilde{H}_i, [M_i]) \tag{33}$$

$$\hat{e}(\tilde{G}, [X])\, \hat{e}([\tilde{G}], X^{-1}) = 1_{\mathbb{G}_T} \text{ for each } X \in \{A_1, A_3, B_1, B_3, M_i\}. \tag{34}$$

With pairing $\hat{e}$ defined as (19), 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]

Commit to $Y \in (Z, R, U, T', W', A_1, A_3, B_1, B_3, M_i)$ by computing

$$[Y] := (\mathcal{O}, \mathcal{O}, Y) + \mathcal{S}_Y\, \boldsymbol{u} = (C_{1,Y}, C_{2,Y}, C_{3,Y}) \in \mathbb{G}_0^3. \tag{35}$$

with independently uniform $\mathcal{S}_Y \leftarrow \mathrm{Mat}_{1\times 3}$. Let $\mathcal{S}_{\tilde{G}} := (0,0,0) \in \mathbb{Z}_p^3$, and let

$$\mathcal{S}_{(32)} := \begin{pmatrix} \mathcal{S}_{A_1} \\ \mathcal{S}_{A_3} \\ \mathcal{S}_Z \\ \mathcal{S}_R \\ \mathcal{S}_{T'} \\ \mathcal{S}_{M_i} \end{pmatrix}, \quad \mathcal{S}_{(33)} := \begin{pmatrix} \mathcal{S}_{B_1} \\ \mathcal{S}_{B_3} \\ \mathcal{S}_Z \\ \mathcal{S}_U \\ \mathcal{S}_{W'} \\ \mathcal{S}_{M_i} \end{pmatrix}, \quad \text{and} \quad \mathcal{S}_{(34),X} := \begin{pmatrix} \mathcal{S}_{\tilde{G}} \\ \mathcal{S}_X \end{pmatrix}. \tag{36}$$

Compute $\tilde{\theta}_{(32)}$, $\tilde{\theta}_{(33)}$ and $\theta_{(34),X}$ for $X \in \{A_1, A_3, B_1, B_3, M_1, \ldots, M_i\}$ where:

$$\tilde{\theta}_{(32)} := \mathcal{S}_{(32)}^{\top} \begin{pmatrix} \mathcal{O}\ \mathcal{O}\ \tilde{A}_0 \\ \mathcal{O}\ \mathcal{O}\ \tilde{A}_2 \\ \mathcal{O}\ \mathcal{O}\ \tilde{G}_z^{-1} \\ \mathcal{O}\ \mathcal{O}\ \tilde{G}_r^{-1} \\ \mathcal{O}\ \mathcal{O}\ \tilde{G}_t^{-1} \\ \mathcal{O}\ \mathcal{O}\ \tilde{G}_i^{-1} \end{pmatrix} = \begin{pmatrix} \mathcal{O}\ \mathcal{O}\ \tilde{\theta}_{1,(32)} \\ \mathcal{O}\ \mathcal{O}\ \tilde{\theta}_{2,(32)} \\ \mathcal{O}\ \mathcal{O}\ \tilde{\theta}_{3,(32)} \end{pmatrix} \quad \in \tilde{\mathbb{G}}_1^{3\times 3}, \tag{37}$$

$$\tilde{\theta}_{(33)} := \mathcal{S}_{(33)}^{\top} \begin{pmatrix} \mathcal{O}\ \mathcal{O}\ \tilde{B}_0 \\ \mathcal{O}\ \mathcal{O}\ \tilde{B}_2 \\ \mathcal{O}\ \mathcal{O}\ \tilde{H}_z^{-1} \\ \mathcal{O}\ \mathcal{O}\ \tilde{H}_u^{-1} \\ \mathcal{O}\ \mathcal{O}\ \tilde{H}_w^{-1} \\ \mathcal{O}\ \mathcal{O}\ \tilde{H}_i^{-1} \end{pmatrix} = \begin{pmatrix} \mathcal{O}\ \mathcal{O}\ \tilde{\theta}_{1,(33)} \\ \mathcal{O}\ \mathcal{O}\ \tilde{\theta}_{2,(33)} \\ \mathcal{O}\ \mathcal{O}\ \tilde{\theta}_{3,(33)} \end{pmatrix} \quad \in \tilde{\mathbb{G}}_1^{3\times 3}, \tag{38}$$

$$\theta_{(34),X} := \mathcal{S}_{(34),X}^{\top} \begin{pmatrix} \mathcal{O}\ \mathcal{O}\ G \\ \mathcal{O}\ \mathcal{O}\ X^{-1} \end{pmatrix} = \begin{pmatrix} \mathcal{O}\ \mathcal{O}\ \theta_{1,(34),X} \\ \mathcal{O}\ \mathcal{O}\ \theta_{2,(34),X} \\ \mathcal{O}\ \mathcal{O}\ \theta_{3,(34),X} \end{pmatrix} \quad \in \mathbb{G}_0^{3\times 3}. \tag{39}$$

Output all $[Y]$, $\tilde{\theta}_{(32)}$, $\tilde{\theta}_{(33)}$, and $\theta_{(34),X}$ dropping redundant $\mathcal{O}$.

[Verifier Algorithm]

Given the above proof and CRS as input, output 1 (as accept) if all the following equations hold. Output 0, otherwise.

$$\begin{aligned} &\begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ \tilde{A}_0 \end{pmatrix} \tilde{\bullet} \begin{pmatrix} C_{1,A_1} \\ C_{2,A_1} \\ C_{3,A_1} \end{pmatrix} + \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ \tilde{A}_2 \end{pmatrix} \tilde{\bullet} \begin{pmatrix} C_{1,A_3} \\ C_{2,A_3} \\ C_{3,A_3} \end{pmatrix} + \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ \tilde{G}_z^{-1} \end{pmatrix} \tilde{\bullet} \begin{pmatrix} C_{1,Z} \\ C_{2,Z} \\ C_{3,Z} \end{pmatrix} \\ &+ \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ \tilde{G}_r^{-1} \end{pmatrix} \tilde{\bullet} \begin{pmatrix} C_{1,R} \\ C_{2,R} \\ C_{3,R} \end{pmatrix} + \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ \tilde{S'}^{-1} \end{pmatrix} \tilde{\bullet} \begin{pmatrix} C_{1,T'} \\ C_{2,T'} \\ C_{3,T'} \end{pmatrix} + \sum_{i=1}^{n} \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ \tilde{G}_i^{-1} \end{pmatrix} \tilde{\bullet} \begin{pmatrix} C_{1,M_i} \\ C_{2,M_i} \\ C_{3,M_i} \end{pmatrix} \\ &= \left(\tilde{\theta}_{(32)}\right)^{\top} \tilde{\bullet} (\boldsymbol{u})^{\top} \end{aligned} \tag{40}$$

$$\begin{aligned} &\begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ \tilde{B}_0 \end{pmatrix} \tilde{\bullet} \begin{pmatrix} C_{1,B_1} \\ C_{2,B_1} \\ C_{3,B_1} \end{pmatrix} + \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ \tilde{B}_2 \end{pmatrix} \tilde{\bullet} \begin{pmatrix} C_{1,B_3} \\ C_{2,B_3} \\ C_{3,B_3} \end{pmatrix} + \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ \tilde{H}_z^{-1} \end{pmatrix} \tilde{\bullet} \begin{pmatrix} C_{1,Z} \\ C_{2,Z} \\ C_{3,Z} \end{pmatrix} \\ &+ \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ \tilde{H}_u^{-1} \end{pmatrix} \tilde{\bullet} \begin{pmatrix} C_{1,U} \\ C_{2,U} \\ C_{3,U} \end{pmatrix} + \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ \tilde{V'}^{-1} \end{pmatrix} \tilde{\bullet} \begin{pmatrix} C_{1,W'} \\ C_{2,W'} \\ C_{3,W'} \end{pmatrix} + \sum_{i=1}^{n} \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ \tilde{H}_i^{-1} \end{pmatrix} \tilde{\bullet} \begin{pmatrix} C_{1,M_i} \\ C_{2,M_i} \\ C_{3,M_i} \end{pmatrix} \\ &= \left(\tilde{\theta}_{(33)}\right)^{\top} \tilde{\bullet} (\boldsymbol{u})^{\top} \end{aligned} \tag{41}$$

$$\begin{pmatrix} C_{1,X} \\ C_{2,X} \\ C_{3,X} \end{pmatrix} \tilde{\bullet} \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ \tilde{G} \end{pmatrix} + \begin{pmatrix} \tilde{C}_{1,\tilde{G}} \\ \tilde{C}_{2,\tilde{G}} \\ \tilde{C}_{3,\tilde{G}} \end{pmatrix} \tilde{\bullet} \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ X^{-1} \end{pmatrix} = (\tilde{\boldsymbol{u}})^{\top} \tilde{\bullet} \left(\theta_{(34),X}\right)^{\top} \tag{42}$$

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

B 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 $\boldsymbol{u} \in \mathbb{G}_0^3$ and $\tilde{\boldsymbol{u}} \in \tilde{\mathbb{G}}_1^3$ are generated as described in Section 5.1. 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 (30) and (31) as re-numbered as follows.

$$\hat{e}([A], \tilde{Y})\, \hat{e}([A], [\tilde{D}]) = \hat{e}(K, \tilde{G})\, \hat{e}(M, \tilde{G})\, \hat{e}(T, [S]) \tag{43}$$

$$\hat{e}([B], \tilde{G}) = \hat{e}(F, [\tilde{D}]) \tag{44}$$

$$\hat{e}([R], \tilde{G}) = \hat{e}(G, [S]) \tag{45}$$

With pairing $\hat{e}$ defined as (19), 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

$$[\delta] := (\mathcal{O}, \mathcal{O}, \delta) + \mathcal{S}_\delta\, \boldsymbol{u} = (C_{1,\delta}, C_{2,\delta}, C_{3,\delta}) \in \mathbb{G}_0^3, \text{ and} \tag{46}$$

$$[\tilde{\rho}] := (\mathcal{O}, \mathcal{O}, \tilde{\rho}) + \mathcal{S}_{\tilde{\rho}}\, \tilde{\boldsymbol{u}} = (C_{1,\tilde{\rho}}, C_{2,\tilde{\rho}}, C_{3,\tilde{\rho}}) \in \mathbb{G}_1^3 \tag{47}$$

where $\mathcal{S}_\delta \leftarrow \mathrm{Mat}_{1\times 3}$ and $\mathcal{S}_{\tilde{\rho}} \leftarrow \mathrm{Mat}_{1\times 3}$. Let T_p be a random 3×3 matrix over $\mathbb{Z}_p$. Compute $\theta_{(43)}$, $\theta_{(44)}$, and $\theta_{(45)}$ as:

$$\begin{aligned} \theta_{(43)} = {} & \mathcal{S}_A^{\top}\, (\mathcal{O}, \mathcal{O}, X) + \mathcal{S}_A^{\top}\, (\mathcal{O}, \mathcal{O}, D) + \mathcal{S}_{\tilde{D}}^{\top}\, (\mathcal{O}, \mathcal{O}, A) + \mathcal{S}_A^{\top}\, \mathcal{S}_{\tilde{D}}\, \boldsymbol{u} \\ & - \mathcal{S}_{\tilde{S}}^{\top}\, (\mathcal{O}, \mathcal{O}, T) + (T_p - T_p^{\top})\, \boldsymbol{u} \end{aligned} \tag{48}$$

$$\theta_{(44)} = \mathcal{S}_B^{\top}\, (\mathcal{O}, \mathcal{O}, G) - \mathcal{S}_{\tilde{D}}^{\top}\, (\mathcal{O}, \mathcal{O}, F) \tag{49}$$

$$\theta_{(45)} = \mathcal{S}_R^{\top}\, (\mathcal{O}, \mathcal{O}, G) - \mathcal{S}_{\tilde{S}}^{\top}\, (\mathcal{O}, \mathcal{O}, G) \tag{50}$$

Output all $[\delta]$, $[\tilde{\rho}]$, $\theta_{(43)}$, $\theta_{(44)}$, and $\theta_{(45)}$ 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.

$$\begin{pmatrix} C_{1,A} \\ C_{2,A} \\ C_{3,A} \end{pmatrix} \tilde{\bullet} \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ \tilde{Y} \end{pmatrix} + \begin{pmatrix} C_{1,A} \\ C_{2,A} \\ C_{3,A} \end{pmatrix} \tilde{\bullet} \begin{pmatrix} C_{1,\tilde{D}} \\ C_{2,\tilde{D}} \\ C_{3,\tilde{D}} \end{pmatrix} \tag{51}$$

$$= \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ K \end{pmatrix} \tilde{\bullet} \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ \tilde{G} \end{pmatrix} + \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ M \end{pmatrix} \tilde{\bullet} \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ \tilde{G} \end{pmatrix} + \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ T \end{pmatrix} \tilde{\bullet} \begin{pmatrix} C_{1,\tilde{S}} \\ C_{2,\tilde{S}} \\ C_{3,\tilde{S}} \end{pmatrix} + \left(\theta_{(43)}\right)^{\top} \tilde{\bullet} \left(\tilde{\boldsymbol{u}}\right)^{\top}$$

$$\begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ \tilde{G} \end{pmatrix} \tilde{\bullet} \begin{pmatrix} C_{1,B} \\ C_{2,B} \\ C_{3,B} \end{pmatrix} = \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ F \end{pmatrix} \tilde{\bullet} \begin{pmatrix} C_{1,\tilde{D}} \\ C_{2,\tilde{D}} \\ C_{3,\tilde{D}} \end{pmatrix} + \left(\theta_{(44)}\right)^{\top} \tilde{\bullet} \left(\tilde{\boldsymbol{u}}\right)^{\top} \tag{52}$$

$$\begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ \tilde{G} \end{pmatrix} \tilde{\bullet} \begin{pmatrix} C_{1,R} \\ C_{2,R} \\ C_{3,R} \end{pmatrix} = \begin{pmatrix} \mathcal{O} \\ \mathcal{O} \\ G \end{pmatrix} \tilde{\bullet} \begin{pmatrix} C_{1,\tilde{S}} \\ C_{2,\tilde{S}} \\ C_{3,\tilde{S}} \end{pmatrix} + \left(\theta_{(45)}\right)^{\top} \tilde{\bullet} \left(\tilde{\boldsymbol{u}}\right)^{\top} \tag{53}$$