Redesigning Group Key Exchange Protocol based on Bilinear Pairing Suitable for Various Environments

Yvo Desmedt[1] * and Atsuko Miyaji[2] **

[1] University College London
y.desmedt@cs.ucl.ac.uk
[2] Japan Advanced Institute of Science and Technology
miyaji@jaist.ac.jp

Abstract. Group key exchange (GKE) allows a group of n parties to share a common secret key over insecure channels. Since key management is important, NIST is now looking for a standard. The goal of this paper is to redesign GKE using bilinear pairings, proposed by Desmedt and Lange, from the point of view of arrangement of parties. The arrangement of parties is called a party tree in this paper. Actually, we are able to *redesign* the party tree, to reduce the computational and communicational complexity compared with the previous scheme, when GKE is executed among a small group of parties. We also redesign the general party tree for a large number of parties, in which each party is in a different environment such as having large or limited computational resources, electrical power, etc.

Key words: group key exchange, pairing, party tree, graph theory

1 Introduction

A group key exchange protocol (GKE) allows a group of n parties to share a common secret key over an insecure channel and, thus, parties in the group can encrypt and decrypt messages among group members. Secure communication among many parties has become an integral part of many applications. For example, ad hoc wireless networks are deployed in many areas such as homes, schools, disaster areas, etc., where a network is susceptible to attacks ranging from passive eavesdropping to active interference. Besides ad hoc networks, another environment where ad hoc groups are popular is in the context of new emerging social networks. The most well-known examples are Facebook and the professional network LinkedIn. Note that, as pointed out by Katz-Yung [15] some dynamic GKE protocols are slower than restarting from scratch with an efficient GKE. So, we do not consider making our protocols dynamic.

* A part of this work was done while funded by PSRC EP/C538285/1, by BT, (as BT Chair of Information Security), by RCIS (AIST), and by JSPS/S-08153.
** This study is partly supported by Grant-in-Aid for Scientific Research (A), 21240001.

Some previous GKEs are based on the DH-key exchange protocol [7, 8]. These GKEs were defined over a finite field, however, it can also be naturally defined over an elliptic curve for efficiency, denoted by GKE-ECDH. Some GKEs [10, 12], based on Joux's tripartite key exchange protocols [14] using a bilinear pairing, follow constructions by [7, 8], which are denoted by GKE-BP. Other GKEs [1–3] are not based on [7, 8], which combine DH-key over an elliptic curve and Joux's tripartite key exchange. In this paper, we focus on GKE-BP proposed by Desmedt and Lange [12].

Let us discuss the differences between GKE-ECDH and GKE-BP from the point of view of arrangement of parties. A party-arrangement tree is called a *party tree* in this paper. GKE-ECDH is based on a two-party GKE and, thus, the generalization to an n-party GKE uses a binary tree. On the other hand, GKE-BP is based on Joux's [14] three-party GKE. In order to generalize the three-party GKE-BP to an n-party GKE-BP, n parties are simply arranged in a triplet tree. As a result, GKE-BP has the merit of reducing the height of the tree which arranges parties. In addition to this fact, GKE-BP is fit for combination with the short signature [6], since both GKE-BP and the short signature are based on a bilinear pairing over elliptic curve, using similar technology. However, there might be room for improvement in the arrangement of multiple triangles from the point of view of communicational or computational complexity; and the most efficient party tree might be different according to the number of parties. Previous protocols [12] based on GKE-BP, denoted by BDI-BP and BDII-BP[3] in this paper, do not focus on the party tree, and use a triplet tree to arrange parties by connecting the triangles at the nodes. In fact, few generalizations were developed to achieve an n-party version, although Joux's 3-party GKE-BP is a heavily cited paper.

In this paper, we explore the improvement of n-party versions of GKE-BP by redesigning the party tree, and investigate what type of party tree is suitable for given each condition: for example, the number of parties is decided according to application, or in the case of a large number of parties, each party may be under a different environment, such as having limited computational resources, electrical power, etc. As a result, we succeed in constructing a new GKE based on bilinear pairings which uses a new party tree and arranges parties by connecting the triangles at the edges, which we call an *edge-based GKE*. Compared with our edge-based GKE, BDII-BP is called a node-based GKE. We also analyze the performance of our edge-based GKE, and, the node-based GKE carefully, and show that the most efficient party arrangement is different, according to the number of parties, n. In addition, each tree has various strengths and weaknesses. Edge-based scheme has an advantage over node-based GKE in sent message complexity for parties with low computational resources. On the other hand, node-based GKE has an advantage over Edge-based scheme in received message complexity. Edge-based scheme is suitable in the case of a small number of parties. From the point of view of computational

[3] We focus on BDII-BP in this paper since BDII-BP is more efficient than BDI-BP.

complexity, our edge-based GKE can work more efficiently than node-based GKE for $4 \leq n \leq 9$ and $16 \leq n \leq 21$.

We also investigate GKE in the case of a large number of parties. In such a case, each party may be in a different environment. For example, some parties may have large computational resources, but others may have few resources; and some parties may have almost unlimited electrical power, but others may run on small batteries. In [12], a GKE among a group with two types of parties is discussed. n_1 parties have large computational resources and n_2 parties have few resources. In this paper, we give the general and systematic construction of a GKE among a group by redesigning the party tree, in which n_1, n_2, $\cdots$, n_m parties have computational resources in descending order, which we call an $(n_1, n_2, \cdots, n_m)$-GKE. From a practical point of view, the necessary features for GKE depend on the application. By using our results, we can choose the optimal party tree according to each application.

This paper is organized as follows. Section 2 summarizes computational assumptions, security assumptions, and security definitions of GKE, together with notations. Section 3 reviews the previous GKEs based on bilinear pairings. Section 4 presents our new edge-based GKE using bilinear pairings, after making clear the differences between edge-based and node-based GKE. Section 5 shows how to construct $(n_1, n_2, \cdots, n_m)$-GKE, in which n_1, n_2, $\cdots$, n_m parties have computational resources in descending order.

2 Preliminary

This section summarizes notations, assumptions, and the basic security notions used in this paper.

2.1 The Bilinear Map, its Related Assumptions, and Security Model of GKE

Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two cyclic groups of prime order q. $\mathbb{G}_1$ (resp. $\mathbb{G}_2$) is represented additively (resp. multiplicatively), where O (resp. 1) represents the zero element (identity element) for addition (multiplication) in $\mathbb{G}_1$ (resp. $\mathbb{G}_2$). The following bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ is defined over $\mathbb{G}_1$.

1. Bilinearity: $\hat{e}(G_0 + G_1, G_2) = \hat{e}(G_0, G_2) * \hat{e}(G_1, G_2)$, $\hat{e}(G_0, G_1 + G_2) = \hat{e}(G_0, G_1) * \hat{e}(G_0, G_2)$ $\forall G_0, G_1, G_2 \in \mathbb{G}_1$.
2. Non-degeneracy: $\hat{e}(G, G) \neq 1$ for any $G \in \mathbb{G}_1 \setminus \{O\}$.
3. Computability: There is an efficient algorithm to compute $\hat{e}(G_0, G_1)$ for any $G_0, G_1 \in \mathbb{G}_1$.

Let k be a security parameter. A DBDH (Decision Bilinear Diffie Hellman) parameter generator $\mathcal{IG}$ is a probabilistic polynomial time (PPT) algorithm that on input 1^k, outputs a description of the above $(\mathbb{G}_1, \mathbb{G}_2, \hat{e})$. The *DBDH problem* with respect to $\mathcal{IG}$ is: given random $G, Y_1, Y_2, Y_3 \in \mathbb{G}_1$ and $z \in \mathbb{G}_2$ where $Y_i = \alpha_i G$ $(i = 1, \cdots, 3)$ to decide whether $z = \hat{e}(G, G)^{\alpha_1\alpha_2\alpha_3}$ or not. More precisely, we say that $\mathcal{IG}$ satisfies the *DBDH assumption* if $|p_1 - p_2|$ is negligible (in k) for

all PPT algorithms A, where $p_1 = \Pr[(\mathbb{G}_1, \mathbb{G}_2, \hat{e}) \leftarrow \mathcal{IG}(1^k); G, Y_1 = \alpha_1 G, Y_2 = \alpha_2 G, Y_3 = \alpha_3 G \leftarrow \mathbb{G}_1 : A(\mathbb{G}_1, \mathbb{G}_2, \hat{e}, G, Y_1, Y_2, Y_3, \hat{e}(G,G)^{\alpha_1\alpha_2\alpha_3}) = 0]$ and $p_2 = \Pr[(\mathbb{G}_1, \mathbb{G}_2, \hat{e}) \leftarrow \mathcal{IG}(1^k); G, Y_1 = \alpha_1 G, Y_2 = \alpha_2 G, Y_3 = \alpha_3 G \leftarrow \mathbb{G}_1, z \leftarrow \mathbb{G}_2 : A(\mathbb{G}_1, \mathbb{G}_2, \hat{e}, G, Y_1, Y_2, Y_3, z) = 0]$. This assumption is believed to hold if $\hat{e}$ is a Weil/Tate pairing on either a supersingular elliptic curve or an ordinary elliptic curve [18].

Let Π be a GKE protocol with n parties, let k be a security parameter, and let $\mathcal{P} = \{P_1, \cdots, P_n\}$ be a set of n parties, where n is bounded above by a polynomial in k. We follow the security model described in [15]. Here we review their definitions while focusing on models used in this paper.

We assume that parties do not deviate from the protocol and an adversary $\mathcal{A}$ is an outsider, that is, never participates as a party in the protocol. The interaction between $\mathcal{A}$ and parties occurs only via the following oracle queries, where Π_P^i denotes the i-th instance of party P; $\mathsf{sk}_\mathsf{P}^\mathsf{i}$ denotes the session key after execution of the protocol by Π_P^i; sid_P^i denotes the session identity for instance Π_P^i; and pid_P^i denotes the partner identity for instance Π_P^i.

$\mathsf{Send}(P, i, m)$: This sends message m to instance Π_P^i, and outputs the reply generated by this instance. This query models an active attack.

$\mathsf{Execute}(P_1, i_1, \cdots, P_n, i_n)$: This executes the protocol between the unused instances $\{\Pi_{P_j}^{i_j}\}_{1 \le j \le n}$ and outputs the transcript of the execution;

$\mathsf{Reveal}(P, i)$: This outputs a session key $\mathsf{sk}_\mathsf{P}^\mathsf{i}$ for a terminated instance Π_P^i.

$\mathsf{Corrupt}(P)$: This outputs the long-term secret key of a party P.

$\mathsf{Test}(P, i)$: This query is asked only once, at any time during the adversary's execution. A bit $b \in \{0, 1\}$ is chosen uniformly at random. The adversary is given $\mathsf{sk}_\mathsf{P}^\mathsf{i}$ if $b = 1$, and a random session key if $b = 0$.

A passive adversary is given access to the Execute, Reveal, Corrupt, and Test oracles, while an active adversary is additionally given access to the Send oracle. The adversary can query Send, Execute, Reveal, and Corrupt oracles several times, but Test oracle is asked only once and on a fresh instance[4].

Finally, the adversary outputs a guess bit b'. Then, Succ, the event in which $\mathcal{A}$ wins the game for a protocol Π, occurs if $b = b'$ where b is the hidden bit used by the Test oracle. The advantage of $\mathcal{A}$ is defined as $\mathtt{Adv}_\Pi(k) = |\mathrm{Prob}[\mathsf{Succ}] - 1/2|$. We say Π is a secure group key exchange protocol if, for any PPT passive adversary $\mathcal{A}$, $\mathtt{Adv}_\Pi^{\mathtt{KE}}$ is negligible (in k). We say Π is a secure group authenticated key exchange protocol if, for any PPT active adversary $\mathcal{A}$, $\mathtt{Adv}_\Pi^{\mathtt{AKE}}$ is negligible (in k). The requirement of forward secrecy is already included in the above definition, since $\mathcal{A}$ is allowed to access the Corrupt oracle in each case.

The Katz-Yung compiler [15], or a variant of [13], transforms any GKE which is secure against a passive adversary with or without forward secrecy into one that is secure against active adversaries with or without forward secrecy, respectively. Let us briefly describe how the compiler transforms any passive-

[4] Π_P^i is a fresh instance unless one of the following is true: (1) $\mathcal{A}$, at some point, queried $\mathsf{Reveal}(P, i)$ or $\mathsf{Reveal}(P', j)$ with $P' \in \mathsf{pid}_P^i$; or (2) $\mathcal{A}$ queried $\mathsf{Corrupt}(P'')$ with $(P'' \in \mathsf{pid}_P^i)$ before a query of $\mathsf{Send}(P, i, *)$ or $\mathsf{Send}(P', j, *)$ with $(P' \in \mathsf{pid}_P^i)$.

adversary-secure GKE into active-adversary-secure GKE: to avoid replay attacks, the compiler introduces a fresh random nonce for each party for each execution of the protocol, adds a message number for each party, and makes a signature on message, the message number, and the nonce by using a strongly unforgeable signature under adaptive chosen message attack. As for the detailed construction, refer to [15, 13]. This paper focuses on redesigning BDII-BP from the graph-theory point of view, and investigates the optimal party tree for a given condition, where BDII-BP is secure against a passive attack and does not have a long-term secret key. Therefore, we focus solely on the passive case without long-term secret key. Note that, our scheme will be shown to be secure against passive adversaries, and, thus, both our scheme and BDII-BP are transformed to become secure against active adversaries.

2.2 Assumptions regarding Computational Complexity

We make some assumptions necessary to compute the computational complexity. The GKE we will build consists of scalar multiplications on $\mathbb{G}_i$ ($i = 1, 2$), multiplications on $\mathbb{G}_i$, and pairings $\hat{e}$. We denote the computational complexity of a single scalar multiplication on $\mathbb{G}_i$, a single multiplication on $\mathbb{G}_i$, and a single pairing, by SM_i, M_i, and e, respectively, where $i = 1$ or 2. Based on the current security parameters, the size of $\mathbb{G}_2$ is 6 or more times larger than $\mathbb{G}_1$. Using the conventional algorithm [16], the ratio of computational complexity of M_2 versus M_1 can be set to $M_1 = \left(\frac{|\mathbb{G}_1|}{|\mathbb{G}_2|}\right)^2 M_2$. Similarly, the ratio of the computational complexity of SM_i versus M_i can be set to $SM_i = \frac{3|\mathbb{G}_i|}{2} M_i$ for each $\mathbb{G}_i$. The computational complexity of these operations, in descending order, is $e > SM_2 > SM_1 > M_2 > M_1$. In our evaluation, we focus primarily on the computational complexity of e, SM_1, and M_2. Note that we do not use a scalar multiplication on $\mathbb{G}_2$ in any GKE presented in this paper.

2.3 Notation and Assumptions regarding GKE

This paper deals with each computational resources or electrical power slightly precisely. For this purpose, we introduce notation, (n_1, n_2)-GKE, which means GKE among n_1 parties that have large computational resources and enough electrical power, and n_2 parties that have low computational resources or are running on batteries. More generally, we explore $(n_1, n_2, \cdots, n_m)$-GKE among n_1, $n_2, \cdots, n_m$ parties, each with computational resources or electrical power levels in descending order.

Let us first make some observations regarding GKE. In this paper, when we evaluate the communicational complexity per party, it is from the point of view of the party with the maximum *sent* and *received* data. We distinguish between point-to-point and broadcast communication, while we do not distinguish between multicast and broadcast communication. We use $\mathbf{p}_i$ (resp. $\mathbf{b}_i$) to denote a message in $\mathbb{G}_i$ ($i = 1, 2$) sent/received through point-to-point (resp. broadcast) communication.

Another measure for comparing protocols is the computational complexity per party. Keeping the discussion in Section 2.2 in mind, we focus on the computational complexity of e, SM_1, and M_2.

We also introduce a concept, "*auxiliary elements*". In some GKEs, some parties can compute a shared key by themselves, that is, they can compute a shared key using their own secret key and public data. Some parties, however, cannot compute a shared key by themselves, that is, they need some additional data computed and sent by others. Moreover, this data has to be received and stored by the recipient. The maximum number of auxiliary elements a party receives is denoted by MAE. MAE is also a good characteristic for evaluating each GKE.

To express these evaluations in detail we use the notation (n_1-[#e, #SM_1, #M_2], n_2-[#e, #SM_1, #M_2], $\cdots$, n_m-[#e, #SM_1, #M_2]; #MAE)-GKE. For example, (n_1 -[3, 4, 5], n_2 -[1, 1, 0]; 2)-GKE indicates that n_1 parties compute $3e + 4SM_1 + 5M_2$; n_2 parties compute $1e + 1SM_1$; some parties need to receive 2 auxiliary elements for the shared key.

For the sake of completeness, we define a triplet tree. A triplet tree is a hypergraph [4] (V, E) in which each hyperedge is a 3-set, and the intersection of two edges is a single vertex or empty. Note that some of our constructions are not triplet trees (see Section 4). Our constructions can also be regarded as a hypergraph (V, E) in which each hyperedge is a 3-set, and which is vertex-connected. When such a hyperedge involves the vertices a, b, c we often denote it as Δ_{abc}. We also denote an edge which involves the vertices a, b as E_{ab}. Although we use a normal graph to represent this hypergraph, the hyperedge will be obvious.

3 Background

This section summarizes previous GKEs based on bilinear maps. The original GKE, called BDI [7] and BDII [8], are constructed over a finite field or an elliptic curve. They were adapted to work using pairings in [10, 12]. BDI using pairings was proposed by [10], but it is neither more efficient than [12] nor adapted to the situation of parties with different computational resources since it is fully contributory. We are interested in dealing with parties having different computational resources each other and, thus, redesigning the asymmetric party tree. This is why we focus on BDII based on pairings [12], which we call BDII-BP.

In BDII-BP, parties are arranged in a tree based on a triangle, in which each node is connected to two other triangles (See Fig 3.1). We denote the parent, the two left children, or the two right children of i by par(i), l.child.1(i) and l.child.2(i), or r.child.1(i) and r.child.2(i), respectively, and the sibling of i, who is in the same triangle, by sib(i) (See Fig 3.2). The concepts of child and parent of i are defined by using the distance from a shaded triangle Δ_{123}. In these figures, one party is set to each node; black nodes correspond to parties with large computational resources; white nodes correspond to parties with low computational resources; and a shaded triangle corresponds to the exact triangle of a shared key.

Protocol 1 (BDII-BP[12])

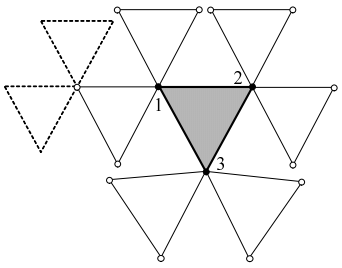

Fig. 3.1. BDII-BP using a bilinear map

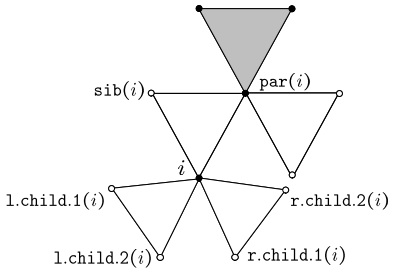

Fig. 3.2. Neighbors of P_i

1. *Each P_i computes $Z_i = r_i G$ for a secretly chosen $r_i \in \mathbb{Z}_q^*$ and sends it to 6 parties such as its 2 left children, 2 right children, the sibling, and the parent.*
2. *Each P_i computes both $X_{\mathtt{left},i}$ and $X_{\mathtt{right},i}$ and multicasts these respectively to its left and right descendants, where*

$$\begin{aligned} X_{\mathtt{left},i} &= \frac{\hat{e}(Z_{\mathtt{par}(i)}, Z_{\mathtt{sib}(i)})^{r_i}}{\hat{e}(Z_{\mathtt{l.child.1}(i)}, Z_{\mathtt{l.child.2}(i)})^{r_i}} = \frac{\hat{e}(Z_{\mathtt{par}(i)}, r_i Z_{\mathtt{sib}(i)})}{\hat{e}(Z_{\mathtt{l.child.1}(i)}, r_i Z_{\mathtt{l.child.2}(i)})} \\ &= \hat{e}(r_i Z_{\mathtt{par}(i)}, Z_{\mathtt{sib}(i)}) \cdot \hat{e}(Z_{\mathtt{l.child.1}(i)}, -r_i Z_{\mathtt{l.child.2}(i)}), \\ X_{\mathtt{right},i} &= \frac{\hat{e}(Z_{\mathtt{par}(i)}, Z_{\mathtt{sib}(i)})^{r_i}}{\hat{e}(Z_{\mathtt{r.child.1}(i)}, Z_{\mathtt{r.child.2}(i)})^{r_i}} = \frac{\hat{e}(Z_{\mathtt{par}(i)}, r_i Z_{\mathtt{sib}(i)})}{\hat{e}(Z_{\mathtt{r.child.1}(i)}, r_i Z_{\mathtt{r.child.2}(i)})} \\ &= \hat{e}(r_i Z_{\mathtt{par}(i)}, Z_{\mathtt{sib}(i)}) \hat{e}(Z_{\mathtt{r.child.1}(i)}, -r_i Z_{\mathtt{r.child.2}(i)}). \end{aligned}$$

3. *Each P_i computes a shared key $K = \hat{e}(r_i Z_{\mathtt{par}(i)}, Z_{\mathtt{sib}(i)}) \prod_{j \in ancestor(i)} X_j = \hat{e}(G, G)^{r_1 r_2 r_3}$.*

BDII-BP works[5] in $(\frac{n}{4} - \frac{3}{4}, \frac{3n}{4} + \frac{3}{4})$-GKE, where $\frac{n}{4} - \frac{3}{4}$ parties execute $3e + 4SM_1 + (\log_4 (n+1))M_2$ at most by keeping and reusing $\hat{e}(r_i Z_{\mathtt{par}(i)}, Z_{\mathtt{sib}(i)})$, and $\frac{3n}{4} + \frac{3}{4}$ parties execute $e + 2SM_1 + (\log_4 (n+1) - 1)M_2$ at most.

4 Redesigning GKE based on Bilinear Pairing

In this section, we present our basic new GKEs, which use a new arrangement of parties by using triangles that overlap at edges. When viewing the triangles as hyperedges, this formally corresponds with hyperedges at which an intersection corresponds to two vertices. Any previous GKE based on bilinear pairings uses a triplet tree, and so triangles were connected to each other at single nodes. Before showing our new edge-based GKE, let us investigate the differences between edge-based and node-based GKEs when using bilinear pairings.

4.1 Differences between edge-based and node-based GKE

We present three GKEs (Protocols 2, 3, and 4) between 7 parties, and investigate the differences among them, where Figures 4.1, 4.2, and 4.3 show each

[5] From now on we will drop the notations (both $\lfloor\ \rfloor$ and $\lceil\ \rceil$) when there is no confusion.

arrangement of parties, respectively. In these figures, arrows correspond to the flow to compute the shared key and other descriptions such as nodes and a shaded triangles are the same as in Section 3. Protocol 2 is a node-based GKE using bilinear pairings; this protocol is easily derived from the previous GKE [12]. Protocols 3 and 4 are edge-based GKEs, which are new arrangements introduced in this paper. Let us compare Protocols 3 and 4 and, then, Protocols 2 and 4 after we describe these protocols.

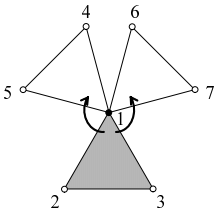

Fig. 4.1. Protocol 2

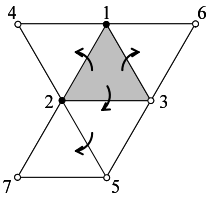

Fig. 4.2. Protocol 3

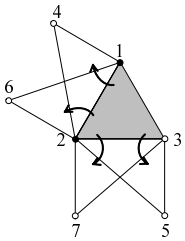

Fig. 4.3. Protocol 4

Protocol 2 ((1 -[3, 4, 2], 4 -[1, 2, 1], 2 -[1, 2, 0]; 1)-GKE)

1. *Each party P_i computes $Z_i = r_i G$ for a (private) randomly chosen $r_i \in \mathbb{Z}_q^*$ and sends it to its neighbors.*
2. *P_1 computes two auxiliary elements $X_{4,5}$ and $X_{6,7}$ and sends them to (P_4, P_5) and (P_6, P_7), respectively, where*

$$X_{4,5} = \frac{\hat{e}(Z_2, Z_3)^{r_1}}{\hat{e}(Z_4, Z_5)^{r_1}} = \hat{e}(r_1 Z_2, Z_3)\hat{e}(Z_4, -r_1 Z_5) \text{ (reusing } \hat{e}(r_1 Z_2, Z_3));$$
$$X_{6,7} = \frac{\hat{e}(Z_2, Z_3)^{r_1}}{\hat{e}(Z_6, Z_7)^{r_1}} = \hat{e}(r_1 Z_2, Z_3)\hat{e}(Z_6, -r_1 Z_7)$$

3. *A shared key is given as $K = \hat{e}(G, G)^{r_1 r_2 r_3}$. P_1 computes this as $K = \hat{e}(r_1 Z_2, Z_3)$; P_2 as $K = \hat{e}(r_2 Z_1, Z_3)$; P_3 as $K = \hat{e}(r_3 Z_1, Z_2)$; P_4 as $K = X_{4,5}\hat{e}(r_4 Z_1, Z_5)$; P_5 as $K = X_{4,5}\hat{e}(r_5 Z_1, Z_4)$; P_6 as $K = X_{6,7}\hat{e}(r_6 Z_1, Z_7)$; and P_7 as $K = X_{6,7}\hat{e}(r_7 Z_1, Z_6)$ and this shared key is depicted in Figure 4.1 as computed from the shaded area.*

Protocol 3 ((2 -[3, 3, 0], 1 -[1, 2, 2], 3 -[1, 2, 1], 1 -[1, 2, 0]; 2)-GKE)

1. *Each party P_i computes $Z_i = r_i G$ for a (private) randomly chosen $r_i \in \mathbb{Z}_q^*$ and sends it to its neighbors.*

2. *P_1 and P_2 compute 2 auxiliary elements (X_4, X_6) and (X_5, X_7) and send them to (P_4, P_6) and (P_5, P_7), respectively, where*

$$P_1 : X_4 = \frac{\hat{e}(Z_2, Z_3)^{r_1}}{\hat{e}(Z_2, Z_4)^{r_1}} = \hat{e}(r_1 Z_2, (Z_3 - Z_4)) \text{ (reusing } r_1 Z_2);$$
$$X_6 = \frac{\hat{e}(Z_3, Z_2)^{r_1}}{\hat{e}(Z_3, Z_6)^{r_1}} = \hat{e}(Z_3, r_1(Z_2 - Z_6));$$
$$P_2 : X_5 = \frac{\hat{e}(Z_3, Z_1)^{r_2}}{\hat{e}(Z_3, Z_5)^{r_2}} = \hat{e}(r_2 Z_3, (Z_1 - Z_5)) \text{ (reusing } r_2 Z_3);$$
$$X_7 = \frac{\hat{e}(Z_5, Z_3)^{r_2}}{\hat{e}(Z_5, Z_7)^{r_2}} = \hat{e}(Z_5, r_2(Z_3 - Z_7)).$$

3. *A shared key is given as $K = \hat{e}(G, G)^{r_1 r_2 r_3}$. P_1 computes this as $K = \hat{e}(r_1 Z_2, Z_3)$; P_2 as $K = \hat{e}(Z_1, r_2 Z_3)$; P_3 as $K = \hat{e}(r_3 Z_1, Z_2)$; P_4 as $K = X_4 \hat{e}(r_4 Z_1, Z_2)$; P_5 as $K = X_5 \hat{e}(r_5 Z_2, Z_3)$; P_6 as $K = X_6 \hat{e}(r_6 Z_1, Z_3)$; and P_7 as $K = X_5 X_7 \hat{e}(r_7 Z_5, Z_2)$.*

Protocol 4 ((2 -[3, 2, 0], 4 -[1, 2, 1], 1 -[1, 2, 0]; 1)-GKE)

1. *Each party P_i computes $Z_i = r_i G$ for a (private) randomly chosen $r_i \in \mathbb{Z}_q^*$ and sends it to its neighbors.*
2. *P_1 and P_2 compute two auxiliary elements (X_4, X_6) and (X_5, X_7) and send them to (P_4, P_6) and (P_5, P_7), respectively, where*

$$P_1 : X_4 = \frac{\hat{e}(Z_2, Z_3)^{r_1}}{\hat{e}(Z_2, Z_4)^{r_1}} = \hat{e}(r_1 Z_2, Z_3 - Z_4) \text{ (reusing } r_1 Z_2);$$
$$X_6 = \frac{\hat{e}(Z_3, Z_2)^{r_1}}{\hat{e}(Z_3, Z_6)^{r_1}} = \hat{e}(r_1 Z_3, Z_2 - Z_6);$$
$$P_2 : X_5 = \frac{\hat{e}(Z_3, Z_1)^{r_2}}{\hat{e}(Z_3, Z_5)^{r_2}} = \hat{e}(r_2 Z_3, Z_1 - Z_5) \text{ (reusing } r_2 Z_3);$$
$$X_7 = \frac{\hat{e}(Z_3, Z_1)^{r_2}}{\hat{e}(Z_3, Z_7)^{r_2}} = \hat{e}(r_2 Z_3, Z_1 - Z_7).$$

3. *A shared key is given as $K = \hat{e}(G, G)^{r_1 r_2 r_3}$. P_1 computes this as $K = \hat{e}(r_1 Z_2, Z_3)$; P_2 as $K = \hat{e}(Z_1, r_2 Z_3)$; P_3 as $K = \hat{e}(r_3 Z_1, Z_2)$; P_4 as $K = X_4 \hat{e}(r_4 Z_1, Z_2)$; P_5 as $K = X_5 \hat{e}(r_5 Z_2, Z_3)$; P_6 as $K = X_6 \hat{e}(r_6 Z_1, Z_3)$; and P_7 as $K = X_7 \hat{e}(r_7 Z_2, Z_3)$.*

We now compare these protocols by focusing on how to compute auxiliary elements. The computational complexity of each of these 3 GKEs are summarized in Table 1. Here we assume that parties with large computational resources compute at least three pairings, and these with low computational resources compute at most one.

In Protocol 2, Δ_{123}, Δ_{154} and Δ_{167} share a node (not edge) with party P_1 and, thus, the computational complexity of one auxiliary element is $2e + 2SM_1 + M_2$. To compute another auxiliary element, $e + SM_1$ can be *reused*, and, thus the additional computational complexity is $e + SM_1 + M_2$. One auxiliary element enables 2 parties to compute the shared key. P_1 also can compute the shared

Table 1. Computational complexity of GKE among 7 parties

Party Type	large computational resources			low computational resources					
	$\#e$	$\#SM_1$	$\#M_2$	$\#e$	$\#SM_1$	$\#M_2$	$\#e$	$\#SM_1$	$\#M_2$
(node-based GKE) Protocol 2	3	4	2	1	2	1	1	2	0
(edge-based GKE) Protocol 3	3	3	0	1	2	2(1)	1	2	0
(edge-based GKE) Protocol 4	3	2	0	1	2	1	1	2	0

key itself during computation of the auxiliary element. Remark that MAE, the maximum number of auxiliary elements, is 1. In Protocol 3, Δ_{123} shares one edge E_{12} with Δ_{124} and another edge E_{13} with Δ_{136}. Thus, the computational complexity of one auxiliary element such as X_4 or X_6 is $e + SM_1$. In this protocol, computation of each auxiliary element is independent, and, thus, that of another auxiliary element is $e + SM_1$. However, SM_1 can be *reused* to compute the shared key, and, thus only additional computation of e is required for the computation of the shared key. One auxiliary element enables 1 party to compute the shared key. Remark that MAE is 2, and, thus, there exists a party which needs two auxiliary elements. In Protocol 4, Δ_{123} shares one edge E_{12} with Δ_{124} and the same edge E_{12} with Δ_{126}. Thus, the computational complexity of one auxiliary element is $e + SM_1$. To compute another auxiliary element, SM_1 can be *reused*, and, thus the additional computational complexity is e. Furthermore, SM_1 can be *reused* to compute the shared key, and, thus only additional computation of e is required for the computation of the shared key. One auxiliary element enables 1 party to compute the shared key. Remark that MAE is 1.

Let us compare the two edge-based GKEs, Protocols 3 and 4. The differences are:

1. In Protocol 4, no party needs to use two auxiliary elements to compute the shared key. This is due to the "parallel" locations of P_5 and P_7, while in Protocol 3 these two parties are arranged "serially". So, in Protocol 3, P_7 needs 2 auxiliary elements to compute the shared key. (When using a graph (E, E') as used to explain the triplet tree in Section 2.3, then Δ_{257} is at distance 2 from the shaded Δ_{123}, while no triangle is at such a distance in Figure 4.3.)

2. In Protocol 4, computations of auxiliary elements are not independent. In Protocol 3, computation of auxiliary elements is independent. This is due to the "edge-sharing" of Δ_{124} and Δ_{126} in Protocol 4, while in Protocol 3 these two triangles Δ_{124} and Δ_{136} do not share any edge. So, the computations of auxiliary elements X_4 and X_6 in Protocol 3 are done independently.

Comparing the node-based GKE (Protocol 2) with the better edge-based GKE (Protocol 4), the edge-based GKE can reduce the computational complexity of parties with large computational resources by $2SM_2$. As for parties with low computational resources, both the node-based GKE and the edge-based GKE need $e + 2SM_1$. The concepts developed in Protocol 4 are applied to construct

the edge-based GKE among any group in order to reduce MAE and reuse computations.

4.2 New edge-based GKE

We show the generalization of Protocol 4 as Protocol 5. All parties are arranged in a graph which consists of triangles, seen in Figure 4.4. Figure 4.5 shows the party tree which describes the relation between a parent and two children, where all nodes except leaves have two children, and a parent node generates two auxiliary elements for its two children to compute a shared key. The parties P_1, P_2, and P_3 are parents to each other, that is, P_1(resp. P_2, resp. P_3) is the parent of P_3(rep. P_1, resp. P_2). In detail, P_i is arranged in a tree that starts with P_1, P_2, or P_3, where the tree is decided by the residue class of i in $\mathbb{Z}_3$, and nodes in leaves correspond to parties with low computational resources or small batteries. Figure 4.6 shows neighbors of a party P_i, which is a close-up of the structure of Figure 4.4. Let a party P_i be an inner node in Figure 4.5. Then, neighbors of P_i are described in Figure 4.6: $P_{\mathtt{par}(i)}$ (resp. $P_{\mathtt{par}(\mathtt{par}(i))}$) corresponds to the parent of P_i (resp. $P_{\mathtt{par}(i)}$) and $P_{\mathtt{l.child}(i)}$ and $P_{\mathtt{r.child}(i)}$ correspond to left and right children of P_i in Figure 4.5. P_i computes auxiliary elements for parties $P_{\mathtt{l.child}(i)}$ and $P_{\mathtt{r.child}(i)}$. By using the residue class of i in $\mathbb{Z}_3$, i can be represented by $i = 3 \cdot j_i + a_i$ for $a_i = 1, 2, 3$ and $j_i \geq 0$. So, two children $P_{\mathtt{l.child}(i)}$ and $P_{\mathtt{r.child}(i)}$ of P_i ($i \geq 1$) are denoted by $P_{3(2j_i+1)+a_i}$ and $P_{3(2j_i+2)+a_i}$, respectively.

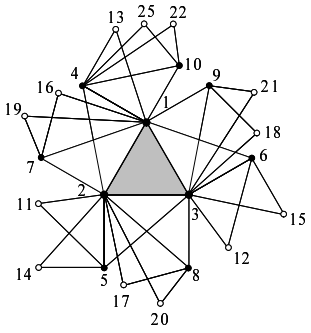

Fig. 4.4. Protocol 5

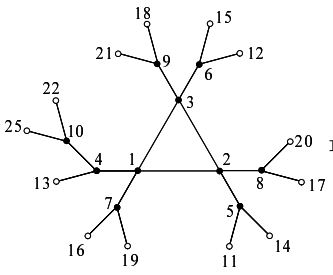

Fig. 4.5. Party tree of Protocol 5

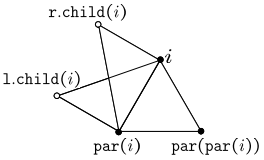

Fig. 4.6. Neighbors of P_i

Protocol 5 ($(\frac{n-3}{2}$-$[3, 2, \log_2(\frac{n}{3}+1)-1], \frac{n+3}{2}$-$[1, 2, \log_2(\frac{n}{3}+1)]; \log_2(\frac{n}{3}+1))$-GKE)

1. *Each party P_i computes $Z_i = r_i G$ using a (private) uniformly random chosen $r_i \in \mathbb{Z}_q^*$ and sends Z_i to its parent, children, and grandchildren.*
2. *Let P_i be an inner-node party, where $i = 3j_i + a_i$ ($a_i = 1, 2, 3$). Then, P_i computes two auxiliary elements, $X_{3(2j_i+b)+a_i}$ ($b \in \{1, 2\}$), and multicasts each to its left and*

right descendants, where

$$X_{3(2j_i+b)+a_i} = \frac{\hat{e}(Z_{\mathtt{par}(i)}, Z_{\mathtt{par}(\mathtt{par}(i))})^{r_i}}{\hat{e}(Z_{\mathtt{par}(i)}, Z_{3(2j_i+b)+a_i})^{r_i}} = \hat{e}(r_i Z_{\mathtt{par}(i)}, Z_{\mathtt{par}(\mathtt{par}(i))} - Z_{3(2j_i+b)+a_i}) \text{ (reusing } r_i Z_{\mathtt{par}(i)}).$$

3. *Let P_i ($i = 1, \cdots, n$) be a party, represented by $i = 3j_i + a_i$ ($a_i = 1, 2, 3$); and the sequence of ancestors of i be $\mathcal{A}_i = v_{i,1} \cdots v_{i,\ell}$, where $v_{i,1} = a_i$ and $v_{i,\ell} = i$. Then, P_i computes a shared key*

$$K = \hat{e}(r_i Z_{\mathtt{par}(i)}, Z_{\mathtt{par}(\mathtt{par}(i))}) \cdot \prod_{t \in \mathcal{A}_i, t = v_{i,\ell}}^{v_{i,1}} X_t.$$

Remarks 1
1. Note that the numbering of the nodes may seem strange. A quick check of Figure 4.5, however, will show that the computational (and communication) load is balanced among the inner nodes. For this reason we call our protocol balanced.
2. Let us call n_l the number of leaves in the tree of Figure 4.5 and n_c the number of non-leaves. A quick check shows that $n_l \leq n_c + 3$. For this reason it is an $(\frac{n-3}{2}, \frac{n+3}{2})$-GKE.
3. Computing one auxiliary element costs $e + SM_1$ and another auxiliary element costs e by reusing SM_1.

4.3 Comparison and discussion

Table 2 summarizes the communicational complexity of Protocols 5 and BDII-BP [12] for $n \geq 4$ and Table 3 summarizes their computational complexity[6]. We see that the sent message complexity of Protocol 5 is less than that of BDII-BP for parties with low computational resources. On the other hand, that of BDII-BP is less than that of Protocol 5 for parties with large computational resources. The received message complexity of BDII-BP is less than that of Protocol 5.

Let us compare both Protocol 5 and BDII-BP from the point of view of computational complexity. Both protocols execute the same number of times of $\hat{e}$ for parties with large and low computational complexity. To simplify the comparison, we focus on M_2 and SM_1 and remove e for formulae, where $\mathrm{Comp}_{\mathtt{BDII-Large}}$ and $\mathrm{Comp}_{\mathtt{Our-Large}}$ are the computational complexity of BDII-BP and Protocol 5 for parties with large computational complexity; and $\mathrm{Comp}_{\mathtt{BDII-Low}}$ and $\mathrm{Comp}_{\mathtt{Our-Low}}$ are for parties with low computational complexity,

$$\mathrm{Comp}_{\mathtt{BDII-Large}} = 2SM_1 + \lceil \log_4 (n+1) \rceil M_2, \; \mathrm{Comp}_{\mathtt{Our-Large}} = (\lceil \log_2 (\frac{n}{3} + 1) \rceil - 2) M_2,$$

$$\mathrm{Comp}_{\mathtt{BDII-Low}} = (\lceil \log_4 (n+1) \rceil - 1) M_2, \quad \mathrm{Comp}_{\mathtt{Our-Low}} = (\lceil \log_2 (\frac{n}{3} + 1) \rceil - 1) M_2.$$

The differences between $\mathrm{Comp}_{\mathtt{BDII-Large}}$ and $\mathrm{Comp}_{\mathtt{Our-Large}}$ ($\mathrm{Comp}_{\mathtt{BDII-Low}}$ and $\mathrm{Comp}_{\mathtt{Our-Low}}$) depend on the number of parties, and choice of $\mathbb{G}_1$ and $\mathbb{G}_2$. We

[6] Protocols 5 and BDII-BP are coincident with Joux's three-party GKE for $n = 3$.

investigate computational complexity when $|\mathbb{G}_1| = 160$ and $|\mathbb{G}_2|/|\mathbb{G}_1| = 6$. Figure 4.7 shows each computational complexity of $\mathrm{Comp}_{\texttt{BDII-Large}}$ and $\mathrm{Comp}_{\texttt{Our-Large}}$ with the number of parties $n \leq 10^5$, and Figure 4.8 shows each complexity of $\mathrm{Comp}_{\texttt{BDII-Low}}$ and $\mathrm{Comp}_{\texttt{Our-Low}}$ with $n \leq 10^5$. Formulae are measured by M_1 and conversion of both SM_1 and M_2 to M_1 were shown in Section 2.2. We see that Protocol 5 can always reduce computational complexity for parties with large computational resources in BDII-BP under the above conditions. However, the computational complexity for parties with low computational resources in Protocol 5 is equal to that for those in BDII-BP for $4 \leq n \leq 9$ and $16 \leq n \leq 21$, as we have seen in the case of $n = 7$ in Section 4.1. Protocol 5 is slightly worse than BDII-BP for other n.

In summary, we have seen that the optimal party tree is different according to the number of parties. In addition, each tree has various strengths and weaknesses. Protocol 5 has an advantage over BDII-BP in sent message complexity for parties with low computational resources. On the other hand, BDII-BP has an advantage over Protocol 5 in received message complexity. Protocol 5 has an advantage over BDII-BP in computational complexity among a small number of parties. On the other hand, BDII-BP has an advantage over Protocol 5 in computational complexity among a large number of parties. From a practical point of view, the necessary features for GKE depend on the application. By using our results, we can choose the optimal party tree according to each application.

Table 2. Sent/received message complexity of several GKEs among n parties

	sent message complexity		received message complexity	
Party Type	large comp.	low comp.	large comp.	low comp.
BDII-BP				
$(\frac{n}{4} - \frac{3}{4}, \frac{3n}{4} + \frac{3}{4})$-GKE	$2\mathbf{b}_2 + 6\mathbf{p}_1$	$2\mathbf{p}_1$	$\log_4 (n+1)\mathbf{b}_2 + 6\mathbf{p}_1$	$\log_4 (n+1)\mathbf{b}_2 + 2\mathbf{p}_1$
Protocol 5				
$(\frac{n-3}{2}, \frac{n+3}{2})$-GKE	$2\mathbf{b}_2 + 7\mathbf{p}_1$	$\mathbf{p}_1$	$\log_2 (\frac{n}{3} + 1)\mathbf{b}_2 + 4\mathbf{p}_1$	$\log_2 (\frac{n}{3} + 1)\mathbf{b}_2 + 2\mathbf{p}_1$

Table 3. Computational complexity of several GKEs among n parties

Party Type	large computational resources			low computational resources		
	$\#e$	$\#SM_1$	$\#M_2$	$\#e$	$\#SM_1$	$\#M_2$
BDII-BP						
$(\frac{n}{4} - \frac{3}{4}, \frac{3n}{4} + \frac{3}{4})$-GKE	3	4	$\lceil \log_4 (n+1) \rceil$	1	2	$\lceil \log_4 (n+1) \rceil - 1$
Protocol 5						
$(\frac{n-3}{2}, \frac{n+3}{2})$-GKE	3	2	$\lceil \log_2 (\frac{n}{3} + 1) \rceil - 2$	1	2	$\lceil \log_2 (\frac{n}{3} + 1) \rceil - 1$

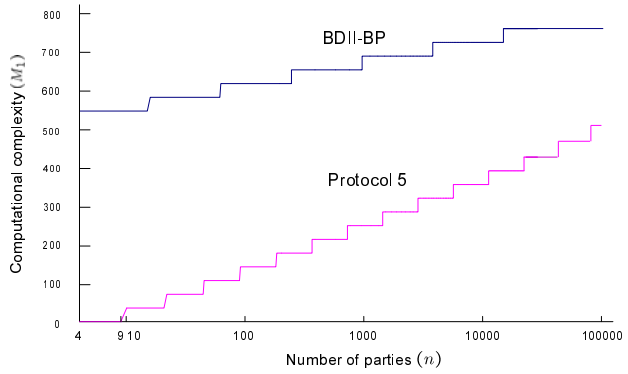

Fig. 4.7. Comparison of $\mathrm{Comp}_{\mathtt{BDII-Large}}$ and $\mathrm{Comp}_{\mathtt{Our-Large}}$ (Estimated by M_1. $|G_1| = 160$, $|G_2|/|G_1| = 6$.)

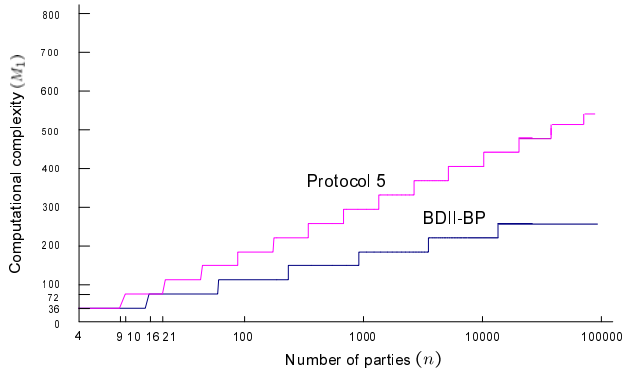

Fig. 4.8. Comparison of $\mathrm{Comp}_{\mathtt{BDII-Low}}$ and $\mathrm{Comp}_{\mathtt{Our-Low}}$ (Estimated by M_1. $|G_1| = 160$, $|G_2|/|G_1| = 6$.)

4.4 Security of Protocol 5

We show that a passive adversary that can break Protocol 5 can be used to solve the DBDH Problem. The detailed proof will be shown in the final paper due to the lack of space.

Theorem 1. *Assuming the DBDH problem over* $\mathbb{G}$ *is hard, Protocol 5, denoted simply by* Π*, is a secure group GKE protocol. Namely,*

$$\mathtt{Adv}^{\mathtt{KE}}_{\Pi}(t, q_{ex}) \leq \mathtt{Adv}^{\mathtt{DBDH}}_{\mathbb{G}}(t'),$$

where $\mathtt{Adv}^{\mathtt{KE}}_{\Pi}(t, q_{ex})$ *is an adversary to* Π *with* q_{ex} Execute *queries and in* t *time, and* $\mathtt{Adv}^{\mathtt{DBDH}}_{\mathbb{G}}(t')$ *is an adversary to DBDH in* $t' = t + (n-3)q_{ex}(e + 2SM_1)$ *time for the number of parties* n*.*

The Katz-Yung compiler [15] and a variant [13] turn Protocol 5 into an authenticated GKE protocol which is secure against active attack, as we have reviewed in Section 2.

Corollary 1. *The authenticated GKE Π' obtained from Protocol 5, denoted simply by Π, by applying the compiler is secure against active adversary. Namely, for the number of* Send *queries, q_s, and the number of* Execute *queries, q_{ex}, we obtain*

$$\mathbf{Adv}^{\mathtt{AKE-fs}}_{\Pi'}(t, q_{ex}, q_s) \leq \mathbf{Adv}^{\mathtt{KE}}_{\Pi}(t', q_{ex}) + \frac{q_s}{2}\mathbf{Adv}^{\mathtt{KE}}_{\Pi}(t', 1) + n\mathsf{Succ}_{\Sigma}(t') + \frac{q_s^2 + q_{ex}q_s}{2^k},$$

where $\mathbf{Adv}^{\mathtt{AKE-fs}}_{\Pi'}(t, q_{ex}, q_s)$ is an adversary to Π', $t' = t + (nq_{ex} + q_s)t_{\Pi'}$, $t_{\Pi'}$ is the time required for an execution of Π' by any party, and Succ_{σ} is the success probability against the signature scheme used, Σ.

5 Construction of GKE among a Large Group

In this section, we will deal with a large group. In such a group, some parties may have large computational resources and electrical power, while others do not, that is, the resources may be different for each party. As we have seen in Section 4, node-based GKEs are suitable for a large group. We will further redesign a party tree of node-based GKEs from the point of view of different *weight*, to construct GKEs suitable for a large group among parties with different resources.

5.1 Variants of node-based GKEs

In order to apply node-based GKEs to parties with different resources, we redesign variants of Protocol 2, in such a way that one party has large computational resources and electrical power to compute all auxiliary elements.

Protocol 6 ($(1\text{-}[\frac{n-3}{2}+1, \frac{n-3}{2}+2, \frac{n-3}{2}], (n-3)\text{-}[1,2,1], 2\text{-}[1,2,0]; 1)$**-GKE)** *Figure 5.1 shows an arrangement of parties, which is a variant of* $(1,6)$*-GKE (Protocol 2) for the case in which 1 party constructs all auxiliary elements.*

1. *$n-3$ parties are set under U_1, which are denoted by $U_{1,i}$ $(i = 1, \cdots, n-3)$.*
2. *Each party U_i $(i = 1, 2, 3)$ computes $Z_i = r_i G$ for a (private) randomly chosen $r_i \in \mathbb{Z}_q^*$ and sends it to its neighbors. Only U_1 broadcasts Z_1 to all parties. Each party $U_{1,i}$ $(i = 1, \cdots, n-3)$ computes $Z_{1,i} = r_{1,i}G$ and sends it to its neighbors (including U_1).*
3. *U_1 computes $\lceil \frac{n-3}{2} \rceil$ auxiliary elements and sends them to the corresponding parties $\{U_{1,2j-1}, U_{1,2j}\}$ $(j = 1, \cdots, \lceil \frac{n-3}{2} \rceil)$,*

$$X_j = \frac{\hat{e}(Z_2, Z_3)^{r_1}}{\hat{e}(Z_{1,2j-1}, Z_{1,2j})^{r_1}} = \hat{e}(r_1 Z_2, Z_3)\hat{e}(-r_1 Z_{1,2j-1}, Z_{1,2j}) (\textit{reusing } \hat{e}(r_1 Z_2, Z_3)). \quad (1)$$

4. *A shared key is given as $K = \hat{e}(G, G)^{r_1 r_2 r_3}$, where U_1 computes $K = \hat{e}(r_1 Z_2, Z_3)$; U_2 computes $K = \hat{e}(r_2 Z_1, Z_3)$; U_3 computes $K = \hat{e}(r_3 Z_1, Z_2)$; and $U_{1,2j-1}$ and $U_{1,2j}$ $(j = 1, \cdots, \lceil \frac{n-3}{2} \rceil)$ compute $K = X_j \hat{e}(r_{1,2j-1} Z_1, Z_{1,2j})$ and $K = X_j \hat{e}(r_{1,2j} Z_1, Z_{1,2j-1})$, respectively.*

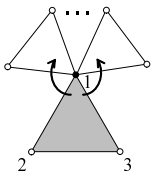

Fig. 5.1. Protocol 6

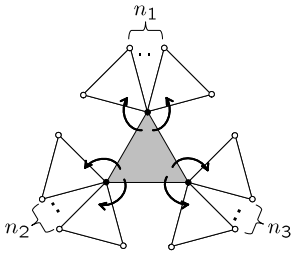

Fig. 5.2. Variants of Protocol 6

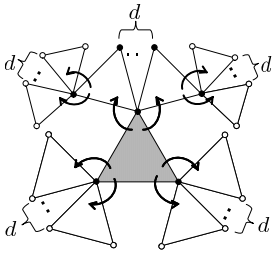

Fig. 5.3. Protocol 7

Remarks 2
1. As we have discussed in Protocol 2, computing $(n-3)$-party auxiliary elements costs $(\frac{n-3}{2}+1)e + (\frac{n-3}{2}+2)SM_1 + \frac{n-3}{2}M_2$ by reusing $e + SM_1$ for every auxiliary element. Note that the shared key itself is computed during this computation.
2. Protocol 6 can be executed as a $(2, n-2)$ or a $(3, n-3)$-GKE by using the computational resources and electrical power of 2 or 3 parties instead of 1 party. It can also be generalized to $(1, 1, 1, 2n_1 + 2n_2 + 2n_3)$-GKE for $n_1 \geq n_2 \geq n_3$ as seen in Figure 5.2.
3. Protocol 6 can be modified in such a way that n parties $\{U_i\}_{i=1}^{n}$ have already shared a key K; $2t$ parties $\{U_{i,j}\}_{j=1}^{2t}$ are set under each U_i; each U_i computes t auxiliary elements and sends each auxiliary element to the appropriate party, $\{U_{i,j}\}_{j=1}^{2t}$, and, thus, $n + 2tn$ parties share the key K by changing Equation (1) to $X_{i,j} = K\hat{e}(-r_i Z_{i,2j}, Z_{i,2j+1})$. This achieves $(n\text{-}[1,2,1], 2tn\text{-}[1,2,1]; 1)$-GKE.

We show that a passive adversary that can break Protocol 6 can be used to solve the DBDH Problem. The detailed proof will be shown in the final paper due to the lack of space.

Theorem 2. *Assuming the DBDH problem over* $\mathbb{G}$ *is hard, Protocol 6, denoted simply by* Π*, is a secure group GKE protocol. Namely,*

$$\mathbf{Adv}^{\mathrm{KE}}_{\Pi}(t, q_{ex}) \leq \mathbf{Adv}^{\mathrm{DBDH}}_{\mathbb{G}}(t'),$$

where $\mathbf{Adv}^{\mathrm{KE}}_{\Pi}(t, q_{ex})$ *is an adversary to* Π *with* q_{ex} Execute *queries and in* t *time, and* $\mathbf{Adv}^{\mathrm{DBDH}}_{\mathbb{G}}(t')$ *is an adversary to DBDH in* $t' = t + \frac{n-3}{2}q_{ex}(e + 5SM_1)$ *time for the number of parties* n.

In the same way as Protocol 5, Protocol 6 can be turned into an authenticated GKE protocol, as we have seen in Section 4.4. We avoid the repetition of the same corollary here.

5.2 Construction of GKE in Different Environments

We investigate $(n_1, \cdots, n_k, m)$-GKE from the following point of view: the computational resources or electrical power required by $n_1, \cdots, n_k$ parties are ar-

ranged according to the allowable number of computations of pairings[7] with $d_1 > \cdots > d_k \geq 3$; and m parties with the lowest computational resources or electrical power can share the key by computing 1 pairing.

For the sake of simplicity, we will show the case of $k = 1$, i.e. (n, m)-GKE, where n parties with large computational resources and electrical power compute d pairings with $d \geq 3$; and m parties with low computational resources or which are using small batteries compute 1 pairing.

Protocol 7 (**(n, m)-GKE**)

Step 1. Check the conditions

If $m > (d-1)^h$ for the height $h = \lceil \log_{2(d-1)} (\frac{n(2d-3)}{3} + 1) \rceil - 1$, then stop and output "Computational resources are not sufficient".

Step 2. Arrange parties according to the given environment of (n, m)

Arrange a party tree in such a way that each node has $(d-1)$ triangles, that is, $2(d-1)$ children; then set n parties to inner nodes on the party tree; finally set m parties to leaves on the party tree.

Step 3. Key exchange among $n + m$ parties based on a variant of Protocol 6

See Figures 5.2 and 5.3. First, Step 1 in Protocol 6 is executed; then, n parties compute $(d-1)$ auxiliary elements and send them to their descendants; and finally, $K = \hat{e}(G, G)^{r_1 r_2 r_3}$ is shared.

Protocol 7 realizes $(n\text{-}[d, d+1, h-1], m\text{-}[1, 2, h]; h)$-GKE for $h = \lceil \log_{2(d-1)} (\frac{n(2d-3)}{3} + 1) \rceil - 1$.

6 Concluding Remarks

Earlier schemes [10, 12] developed to achieve an n-party GKE based on Joux's tripartite scheme, were based on combining several 3-party Joux based GKEs, in which the 3 parties involved were represented by a triangle. Earlier schemes did not focus on arrangement of parties, and, thus, simply, these triangles only had at most one node in common.

We discovered that by just redesigning the arrangement in such a way that these triangles overlap in two parties, we can reduce the communicational or computational complexity according to the number of parties. To obtain this advantage, we used an ingenuous trick by exploiting the mathematics of bilinear pairings (i.e., a new method to compute auxiliary elements) and proposed Protocol 5. By redesigning this arrangement, we can point out that the most efficient party arrangement is different, according to the number of parties. In fact, Protocol 5 can work more efficiently than BDII-BP for $4 \leq n \leq 9$ and $16 \leq n \leq 21$, from the point of view of computational complexity.

Although earlier schemes already discussed asymmetric computational resources, they characterized all machines into two classes based on their resources. Moreover, roughly half the nodes had large computational resources,

[7] n_i parties with large computational resources should be able to compute $d_i \geq 3$ pairings. If $d_i \leq 2$, those parties are assumed to be those with the least computational resources.

while the other half had few resources. We also simply generalized this by redesigning the arrangement of parties, so that we can use more than two classes, and for some of our schemes we do not require a 50-50 division into two classes.

This paper enables us to give the optimal party tree for given either communication or computation complexity. The following open problem still remains: For given each upper bound on P_i's available computation and communication complexity, what is the optimal hyper-graph by using a bilinear-based group key exchange?

References

1. R. Barua, and R. Dutta, "Dynamic Group Key Agreement in Tree-Based Setting", *ACISP 2005*, LNCS **3574**(1994), 101-112, Springer-Verlag.
2. R. Barua, R. Dutta, P. Sarkar, "Extending Joux's Protocol to Multi Party Key Agreement (Extended Abstract)", *INDOCRYPT 2003*, LNCS **2904**(1994), 205-217, Springer-Verlag.
3. R. Barua, R. Dutta, P. Sarkar, "Provably Secure Authenticated Tree Based Group Key Agreement", *ICICS 2004*, LNCS **3269**(1994), 92-104, Springer-Verlag.
4. C. Berge, "Graphs and Hypergraphs", *Elsevier Science*, 1985.
5. A. Abdel-Hafez, A. Miri, L. Orozco-Barbosa, "Authenticated group key agreement protocols for ad hoc wireless networks", Journal of Network Security, 2007.
6. D. Boneh and X. Boyen, "Short signatures without random oracles and the SDH assumption in bilinear groups", *J. Cryptology*, Vol. **21**, No. **2** (2008), 149-177.
7. M. Burmester and Y. Desmedt, "A secure and efficient conference key distribution system", *In Proceedings of Eurocrypt'94*, LNCS **950**(1994), 275-286, Springer-Verlag.
8. M. Burmester and Y. Desmedt, "Efficient and secure conference key distribution", *In Security Protocols*, LNCS **1189**(1997), 119-130, Springer-Verlag.
9. D. Boneh and M. Franklin, "Identity based encryption from the Weil pairing", *In Proceedings of Crypto'01*, LNCS **2139**(2001), 213-229, Springer-Verlag.
10. K. Y. Choi, J. Y. Hwang. and, D. H. Lee, "Efficient ID-based group key agreement with bilinear map", *In Proceedings of PKC'04*, LNCS **2947**(2004), 130-144, Springer-Verlag.
11. M. Ciet, M. Joye, K. Lauter, and P. L. Montgomey, "Trading inversions for multiplications in elliptic curve cryptography", *Designs, Codes and Cryptography*, Vol. **39**, No. **2**(2006), Springer Netherlands, 189–206.
12. Y. Desmedt and T. Lange, "Revisiting pairing based group key exchange", *In Proceedings of FC'08*, LNCS **5143**(2008), 53-68, Springer-Verlag.
13. Y. Desmedt, T. Lange and M. Burmester, "Scalable authenticated tree based group key exchange for ad-hoc groups", *In Proceedings of FC'07*, LNCS **4886**(2007), 104-118, Springer-Verlag.
14. A. Joux, "A One Round Protocol for Tripartite Diffie-Hellman", *J. Cryptology*, Vol. **17**, No. **4** (2004), 263-276.
15. J. Katz and M. Yung, "Scalable Protocols for Authenticated Group Key Exchange", *In CRYPTO 2003*, LNCS **2729**(2003), 110-125, Springer-Verlag.
16. D. E. Knuth, *The Art of Computer Programming, vol. 2, Seminumerical Algorithms*, 2nd ed., Addison-Wesley, Reading, Mass. 1981.
17. E. Konstantinou, "Cluster-based group key agreement for wireless ad hoc networks", ARES 2008, 2008.
18. A. Miyaji, M. Nakabayashi and S. Takano, "New explicit conditions of Elliptic Curve Traces under FR-reduction", *IEICE Trans.*, Fundamentals. vol. E84-A, No.5(2001), 1234-1243.