# $T$-Robust Scalable Group Key Exchange Protocol with $O(\log n)$ complexity

Tetsuya Hatano, Atsuko Miyaji *, and Takashi Sato

Japan Advanced Institute of Science and Technology
miyaji@jaist.ac.jp

**Abstract.** Group key exchange (GKE) allows a large group of $n$ parties to share a common secret key over insecure channels. The goal of this paper is to present $T$-robust scalable GKE with communicational and computational complexity $O(\log n)$ for the size of $n$ parties. As a result, our GKE not only has a resistance to party failures resulting from party crashes, run-down batteries, and network failures, but also satisfies scalability: each party does not need to have the same environment such as computational resources, batteries, etc. The previous schemes in this area focus on Burmester-Desmedt GKE with complexity $O(n)$ (BDI) and without scalability. As a result, the previous robust GKEs, proposed by Jarecki, Kim and Tsudik (JKT), need computational complexity $O(n)$ without scalability although it allows any $T$-party fault in any position.
We, by focusing the well-known Burmester-Desmedt GKE with complexity $O(\log n)$ (BDII), propose a new robust GKE with scalability, called CH-GKE. CH-GKE can *reduce the communicational and computational complexity and allow parties be in different environments*. Then, we extend CH-GKE to increase the number of faults and present $T$-robust scalable efficient GKE by a novel combination of CH-GKE and JKT. Our $T$-robust scalable GKE can work in flexible settings between fault tolerance and efficiency, such as communicational and computational complexity.

Key words: group key exchange, robustness, scalability

## 1 Introduction

A group key exchange protocol (GKE) allows a large 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 within a large group 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 such as Facebook and LinkedIn.

* This study is partly supported by Grant-in-Aid for Scientific Research (A), 21240001

Widely-known GKEs based on the DH-key exchange protocol, such as BDI [4] and BDII [5], can work with constant rounds. The important difference in efficiency between BDI and BDII is that BDI needs communicational complexity $O(n)$ while BDII works with only communicational complexity $O(\log n)$. Another important difference in practicality between BDI and BDII is scalability. BDI assumes that all parties work in the same environment. On the other hand, parties in BDII can work in different environments. For example, some parties may have large computational resources, but others may have low resources; while some parties may have almost unlimited electrical power, others may run on small batteries. On the other hand, both schemes are not robust: if some parties fail during protocol execution, then some other parties cannot share a common secret key. Therefore, the protocol must be re-started from scratch whenever a player fails, which increases the computational, communicational, and round complexity, since total complexity of protocols is multiplied by the number of faults. This is why we need a constant-round GKE that is robust to some parties' failures.

The first robust GKE was proposed by [1], which needs round complexity $O(n)$. Subsequently, the constant-round robust GKE was proposed by [6], which needs communicational complexity $O(n^2)$. The efficient robust GKE, called JKT in this paper, was proposed by [9]. JKT works with constant-round complexity and both communicational[1] and computational complexity $O(n+T)$, and tolerates up to $T$-party failures. JKT has a useful feature of *flexible trade-off* between complexity and fault tolerance. The feature is practical because complexity of GKE can be arranged according to the reliability of network. R-TDH1 [3] achieves the full robustness for a tree-based GKE [10], however, it does not have a feature of flexible trade-off between complexity and fault tolerance.

In this paper, we focus on JKT, which is $T$-robust GKE and satisfies a feature of flexible trade-off between complexity and fault tolerance. JKT is constructed by adding robustness to BDI, and thus, it inherits all features described above from BDI: it can achieve neither $O(\log n + T)$ complexity nor scalability. We present $T$-robust GKE among $n$ parties, which can achieve efficient communicational and computational complexity as well as scalability up to $T$-party failures. From the feature of scalability, our $T$-robust GKE can work in different environments of parties with large resources and/or low resources. Our $T$-robust GKE can work with both computational and communicational complexity $O(\log n + T)$. This is the first result that can work with $O(\log n)$ complexity according to the size of parties $n$. Let us explain how we construct the efficient $T$-robust GKE. In order to achieve such efficiency, we investigate adding robustness to BDII and construct a new scalable constant-round robust GKE, which is secure in the standard model under the Square Decision Diffie-Hellman assumption. The proposed GKE, called *CH-GKE* in this paper, inherits efficiency and scalability described above from BDII. For example, CH-GKE works with 2 round complexity with

[1] Communicational complexity can be measured from the point of view of maximum number of sent or received messages. In JKT, maximum number of sent messages is $O(T)$ but that of received messages is $O(n+T)$.

communicational and computational complexity $O(\log n)$, which tolerates up to $\frac{n}{2}$ party failures. Then, we generalize the construction of CH-GKE to increase the number of party failures. Finally, we combine both CH-GKE and JKT, and propose $T$-robust GKE with $O(\log n)$ complexity, where it tolerates up to any $T$-party failures in any positions.

This paper is organized as follows. Section 2 summarizes computational assumptions, security assumptions and definitions of GKE, together with notations. Section 3 reviews the previous GKEs related with our scheme BDI, JKT, and BDII. Section 4 presents CH-GKE and its generalization. Then, $T$-robust GKE is presented in Section 5. Section 6 compares our $T$-robust GKE with previous GKEs.

## 2 Preliminary

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

### 2.1 The Security Assumptions, and Model of GKE

Let $\mathbb{G}$ be a cyclic group of prime order $p$ and let $k$ be a security parameter.

**Definition 1.** A DDH (Decision Diffie Hellman) parameter generator $\mathcal{IG}_{DDH}$ is a probabilistic polynomial time (PPT) algorithm that, on input $1^k$, outputs a cyclic group $\mathbb{G}$ of prime order $p$. The *DDH problem* with respect to $\mathcal{IG}_{DDH}$ is: given $g$, $g^a$, $g^b$, $y \in \mathbb{G}$ to decide whether $y = g^{ab}$ or not.

**Definition 2.** A Square-DDH (Square-Decision Diffie Hellman) parameter generator $\mathcal{IG}_{Square-DDH}$ is a probabilistic polynomial time (PPT) algorithm that, on input $1^k$, outputs a cyclic group $\mathbb{G}$ of prime order $p$. The *Square-DDH problem* with respect to $\mathcal{IG}_{Square-DDH}$ is: given $g$, $g^x$, $y \in \mathbb{G}$ to decide whether $y = g^{x^2}$ or not.

**Definition 3.** Let $P_1, P_2, \ldots, P_n$ be interactive polynomial-time Turing Machines with history tapes that take part in a Protocol $\Pi$. Protocol $\Pi$ is a *Group Key Exchange* (GKE) if each member $P_i$ computes the same key $K = K_i$ when all group members follow the protocol as specified. We call $P_i$ a party and the $n$ parties a group.

**Definition 4 ([11]).** Let $\Pi$ 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 assume that parties do not deviate from the protocol. An active adversary is given access to the following Send, Execute, Reveal, Corrupt and Test oracles, where all except Test oracle are queried several times, and Test oracle is asked only once at any time during the adversary's execution and on a fresh instance[2]: $\mathsf{Send}(P, i, m)$ sends message $m$ to instance

[2] $\Pi_p^i$ is a fresh instance unless the following is true: $\mathcal{A}$, at some point, queried Reveal $(P, i)$ or Reveal $(P', j)$ with $P' \in \mathsf{pid}_p^i$, where $\mathsf{pid}_p^i$ denotes the party identity for $\Pi_p^i$.

$\Pi_p^i$, and outputs the reply generated by this instance; $\mathsf{Execute}(P_1, i_1, \cdots, P_n, i_n)$ executes the protocol between unused $i_j$-th instances of party $P_j$, $\{\Pi_{P_j}^{i_j}\}_{1 \le j \le n}$, and outputs the transcript of the execution; $\mathsf{Reveal}(P, i)$ outputs a session key $\mathsf{sk}_p^i$ for a terminated instance $\Pi_p^i$; $\mathsf{Corrupt}(P)$ outputs the long-term secret key of a party $P$; $\mathsf{Test}(P, i)$ chooses a bit $b \in \{0, 1\}$ uniformly at random and outputs $\mathsf{sk}_p^i$ or a random session key if $b = 1$ or $b = 0$, respectively.

Finally, the adversary outputs a guess bit $b'$. Then, $\mathsf{Succ}$, the event in which $\mathcal{A}$ wins the game for a protocol $\Pi$, occurs if $b = b'$ where $b$ is the hidden bit used by the $\mathsf{Test}$ oracle. The advantage of $\mathcal{A}$ is defined as $\mathtt{Adv}_{\Pi}(k) = |\mathrm{Prob}[\mathsf{Succ}] - 1/2|$. We say $\Pi$ is a secure group key exchange protocol against an active adversary, if, for any PPT active adversary $\mathcal{A}$, $\mathtt{Adv}_{\Pi}^{\mathtt{KE}}$ is negligible (in $k$).

In the passive adversary model, $\mathsf{Send}$ oracle is ignored. We focus solely on the passive case since the Katz-Yung compiler [11] or a variant of [7] transforms any GKE secure against a passive adversary into one secure against outside active adversaries.

### 2.2 Notation and Assumptions on GKE

We make some assumptions necessary to compute the computational complexity. The GKE we will build consists of multiplications on $\mathbb{G}$, scalar multiplications on $\mathbb{G}$, and inversions on $\mathbb{G}$, whose computational complexity are denoted by $M$, $EM$, and $I$, respectively.

This paper focuses on GKE which can be robust against some number of node faults, while keeping both communicational and computational complexity per party down. Let us first make some observations on 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 $\mathbf{p}$ (resp. $\mathbf{b}$) to denote messages in $\mathbb{G}$ through point-to-point (resp. broadcast) communication, both of which are investigated in two cases of sent and received messages. The computational complexity is measured by the number of $M$, $EM$, and $I$.

We also use the phrase of "*auxiliary elements*", introduced in [8]. In some GKEs, some parties compute data, which help other parties compute a shared key. That is, those parties cannot compute a shared key without auxiliary elements". Actually, failures happen for those parties who need auxiliary elements to compute a shared key but those are not sent to them. In order to achieve a GKE robust against party faults, we will discuss how we provide auxiliary elements in spite of fault parties.

## 3 Background

This section summarizes previous GKEs: BDI [4], its fault-tolerant version [9], and BDII [5]. In BDI, parties are arranged in a ring (See Figure 3.1). When $n$

parties $P_1, P_2, \cdots, P_n$ wish to generate a session key, they proceed as follows (the indices are taken modulo $n$ so that party $P_0$ is $P_n$ and party $P_{n+1}$ is $P_1$).

**Protocol 1 (BDI[4])**

1. *Each $P_i$ computes $z_i = g^{r_i}$ for a secretly chosen $r_i \in \mathbb{Z}_p^*$ and sends it to $P_{i-1}$ and $P_{i+1}$.*
2. *Each $P_i$ computes $x_{[i-1,\,i+1]} = \left(\frac{z_{i+1}}{z_{i-1}}\right)^{r_i} = g^{r_i r_{i+1} - r_{i-1} r_i}$ and broadcasts.*
3. *Each $P_i$ computes a shared key $K = (z_{i-1})^{nr_i} \cdot x_{[i-1,\,i+1]}^{n-1} \cdot x_{[i,\,i+2]}^{n-2} \cdots x_{[i-3,\,i-1]} = g^{r_1 r_2 + r_2 r_3 + \cdots + r_n r_1}$.*

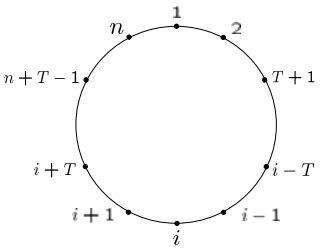


**Fig. 3.1.** BDI

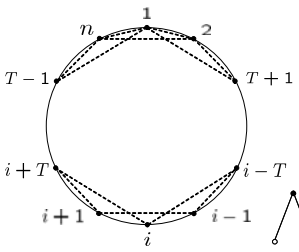


**Fig. 3.2.** JKT

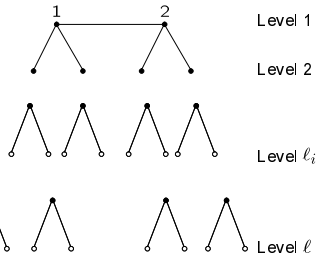


**Fig. 3.3.** BDII

The original BDI is not robust because if any message in the 2nd round is not delivered, then all parties abort, since all parties need auxiliary elements broadcasted in the 2nd round to compute a shared key. BDI is modified to achieve robustness in [9], which is called JKT in this paper. JKT uses Hamilton cycle or Hamilton path to compute a shared key. The maximum failures as well as the key computation in the Hamilton-cycle JKT are different from ones in the Hamilton-path JKT when both compute the same amount of auxiliary elements in the 2nd round. Here we present both Hamilton-cycle and Hamilton-path JKT in the case of sending the same amount of auxiliary elements and discuss each differences (see Figure 3.2).

**Protocol 2 (JKT[9])**

1. *Each $P_i$ computes $z_i = g^{r_i}$ for a secretly chosen $r_i \in \mathbb{Z}_p^*$ and broadcasts.*
2. *Let $\texttt{ActiveList}_1$ be the list of indices of all parties who complete the 1st round. Each $P_i$ computes $x_{[k,\,i]} = \left(\frac{z_i}{z_k}\right)^{r_i} = g^{r_i^2 - r_k r_i}$ for $T$ nearest neighbors to the right and $T$ nearest neighbors to the left among parties $k \in \texttt{ActiveList}_1$ and broadcasts.*
3. *Let $\texttt{ActiveList}_2$ be the list of indices of all parties who complete the 2nd round. Each $P_i$ sorts the parties in $\texttt{ActiveList}_2$ in the same order. We assume that the alive parties constructs a Hamilton cycle or Hamilton path taken twice: $\{P_{a_1}, P_{a_2}, \cdots, P_{a_m}\}$ or $\{P_{a_1}, \cdots, P_{a_{m-1}}, P_{a_m}, P_{a_{m-1}}, \cdots, P_{a_2}\}$, respectively. In the case of Hamilton cycle or path, each $P_{a_i}$ computes a shared key $K = z_{a_{i-1}}^{m \cdot r_{a_i}} \cdot X_{a_i}^{m-1} \cdot X_{a_{i+1}}^{m-2} \cdots X_{a_{i-2}} = g^{r_{a_1} r_{a_2} + r_{a_2} r_{a_3} + \cdots + r_{a_m} r_{a_1}}$ or $K = z_{a_{i-1}}^{(2m-2) \cdot r_{a_i}} \cdot X_{a_i}^{2m-3} \cdot X_{a_{i+1}}^{2m-4} \cdots X_{a_{i-2}} = g^{2(r_{a_1} r_{a_2} + r_{a_2} r_{a_3} + \cdots + r_{a_{m-1}} r_{a_m})}$, respectively. Here, $X_{a_i} = x_{[a_{i-1},\,a_i]} \cdot (x_{[a_{i+1},\,a_i]})^{-1} = g^{r_{a_i} r_{a_{i+1}} - r_{a_{i-1}} r_{a_i}}$.*

Let us discuss how many auxiliary elements in the 2nd round is necessary to achieve $T$-robust GKE. $T$-robust GKE means that GKE toterates all patterns of party faults up to $T$. In the case of Hamilton cycle, $2(T+1)$ auxiliary elements are necessary for $T$-robust GKE. In the case of Hamilton path, $2T$ auxiliary elements are enough to achieve $T$-robust GKE. The detailed comparisons among two types of JKT and our scheme will be shown in Section 6. The security of JKT is given in the theorem below.

**Theorem 1 ([9]).** Assuming the Square-DDH over $\mathbb{G}$ is hard, JKT is a secure group GKE protocol.

Another typical GKE is BDII, proposed by the same authors as BDI. Both BDI and BDII have different features. BDI is fully contributory, but requires $O(n)$ computational and message complexity for any party. In fact, all parties are arranged symmetrically, and thus all parties need to have the same computational resources. On the other hand, BDII is not contributory, but can work with $O(\log n)$ computational and message complexity for any party. Furthermore, parties are not arranged symmetrically, and thus BDII can adapt to the situation of parties with different computational resources.

Up to now, no fault-tolerant version of BDII has been proposed. In order to achieve robust GKE with $O(\log n)$ message size, we focus on BDII in this paper. In BDII, parties are arranged in a binary tree (See Figure 3.3). Therefore, all but the leaves of the tree, each has one parent and two children. We denote the parent, the left child, and the right child by $\mathtt{parent}(i)$, $\mathtt{l.child}(i)$, and $\mathtt{r.child}(i)$, respectively, and denote the set of ancestors of a party $P_i$ by $\mathtt{ancestor}(i)$. Parties $P_1$ and $P_2$ are parents to each other, that is, $P_1$ (resp. $P_2$) is the parent of $P_2$ (resp. $P_1$). Such a relation is used to compute $x_{\mathtt{l.child}(i)}$ and $x_{\mathtt{r.child}(i)}$ for $i = 1$ or 2. However, for a party $P_i$, either $P_1$ or $P_2$ is included in $\mathtt{ancestor}(i)$.

When $n$ parties $P_1, P_2, \cdots, P_n$ wish to generate a session key, they proceed as follows.

**Protocol 3 (BDII[5])**

1. *Each $P_i$ computes $z_i = g^{r_i}$ for a secretly chosen $r_i \in \mathbb{Z}_p^*$ and sends it to its neighbors.*
2. *Each $P_i$ computes both $x_{\mathtt{l.child}(i)} = \left(\frac{z_{\mathtt{parent}(i)}}{z_{\mathtt{l.child}(i)}}\right)^{r_i} = g^{r_{par(i)}r_i - r_i r_{l.child(i)}}$ and $x_{\mathtt{r.child}(i)} = \left(\frac{z_{\mathtt{parent}(i)}}{z_{\mathtt{r.child}(i)}}\right)^{r_i}$ $= g^{r_{par(i)}r_i - r_i r_{r.child(i)}}$ and multicasts these to its left and right descendants, respectively.*
3. *Each $P_i$ computes a shared key $K = (z_{\mathtt{parent}(i)})^{r_i} \cdot \Pi_{j \in \mathtt{ancestor}(i)} x_j = g^{r_1 r_2}$.*

**Theorem 2 ([7]).** Assuming the DDH over $\mathbb{G}$ is hard, BDII is a secure group GKE protocol.

## 4 Robust GKE with $O(\log n)$ complexity

This section presents our GKE with robustness with $O(\log n)$ complexity, called *Cross-Help GKE (CH-GKE)*. We start with intuition on how to make BDII robust, then presents CH-GKE and its generalization.

### 4.1 Intuition

Our robust GKE is constructed over BDII in Section 3. Let us discuss why BDII is not robust in detail. In BDII, a party $P_i$ in level $\ell_i$ makes two auxiliary elements for two children in level $\ell_i + 1$. In the key-sharing phase, a party $P_i$ computes a shared key by using all auxiliary elements sent by ancestors in the path from the parent of $P_i$ to the root $P_1$ or $P_2$ (see Figure 3.3). Note that the path has been determined uniquely in BDII. This is why if a party $P_i$ in level $\ell_i$ fails after the 1st round and cannot send any auxiliary element to descendants, no descendant can compute a shared key. However, unlike BDI, any ancestor of $P_i$ as well as any party who is not in the same path from the failed $P_i$ to the root can compute a shared key.

Before showing our strategy, let us start with a primitive construction of robust GKE. In BDII, an auxiliary element computed by the parent of $P_i$ is necessary for $P_i$ to compute a shared key. Suppose that we add a few extra edges to the graph of BDII and two or more parties compute auxiliary elements for $P_i$, then one of the alive auxiliary elements enables $P_i$ to compute a shared key. For example, suppose that we add 1 more edge to $P_i$ from parties in level $\ell_i - 1$ and add 2 more edges to parties in level $\ell_i + 1$ from $P_i$. Then, if either path from level $\ell_i - 1$ to $P_i$ is alive, 4 parties in level $\ell_i + 1$ as well as $P_i$ can compute a shared key. However, in order to let the $2 \times 4$ path available, $P_i$ computes and multicasts $2 \times 4$ auxiliary elements for 4 parties in level $\ell_i + 1$. Thus, if both $I_i$ edges from parties in level $\ell_i - 1$ to $P_i$ and $O_i$ edges from $P_i$ to parties in level $\ell_i + 1$ exist[3], then the number of auxiliary elements is $I_i \times O_i$. That is, computational and communicational complexity increases multiplicatively according to the number of edges coming in and coming out. Our scheme can reduce the multiplicative cost to the additive cost.

Our scheme, CH-GKE, achieves efficient robust GKE, by realizing a *cross-help* idea with additive computational and communicational complexity. In order to realize such efficient cross-help, we introduce the following ideas.

1. *The division and restoration of auxiliary elements*
   In order to achieve robustness, paths to enable key-sharing need to be increased, which also increases the number of auxiliary elements. For example, for $I_i \times O_i$ paths, $I_i \times O_i$ auxiliary elements are required. In order to reduce computational and communicational complexity, we generalize a technique used in [9]: divide $I_i \times O_i$ auxiliary elements into $I_i + O_i$ parts; divided parts are computed and multicasted; then, in the key-sharing phase, a necessary auxiliary element is restored from two parts of $I_i$ and $O_i$.
2. *A new relation between $P_1$ (resp. $P_2$) and descendants of $P_2$ (resp. $P_1$)*
   Parties $P_1$ and $P_2$ are sisters of each other for those who are descendants of $P_1$ and $P_2$. That is, $P_1$ (resp. $P_2$) is the aunt for parties $P_i$ who are $P_2$'s (resp. $P_1$'s) children. See Figure 4.1.

Let us show an overview of CH-GKE briefly. We denote a child, a parent, and ancestors in the same notation as in Section 3. In addition to these, the sister

[3] $I_i$ edges are input edges to $P_i$. $O_i$ edges are output edges from $P_i$

of $i$ who has the same parent as $i$, the left child and the right child of the sister, and the sister of parent are denoted by $\texttt{sister}(i)$, $\texttt{l.niece}(i)$ and $\texttt{r.niece}(i)$, and $\texttt{aunt}(i)$, respectively.

The relations among neighbors of party $P_i$ are shown in Figure 4.2. In the basic construction of CH-GKE, the cross-help is done by sisters: parent and aunt of $P_i$ in level $\ell_i (i \geq 2)$ make $P_i$'s auxiliary elements, and $P_i$ makes auxiliary elements for 4 parties such as children and nieces in level $\ell_i+1$. Then, $P_i$ computes and multicasts $2+4 = 6$ auxiliary elements. In the general construction, the cross-help is generalized in such a way that $I_i$ parties in level $\ell_i - 1$ make $P_i$'s auxiliary elements, and $P_i$ in level $\ell_i$ makes auxiliary elements for $O_i$ parties in level $\ell_i + 1$. Then, $P_i$ computes $I_i + O_i$ auxiliary elements and multicasts them.

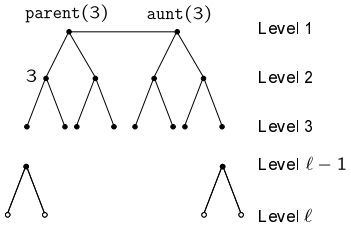


**Fig. 4.1.** Party Tree of CH-GKE

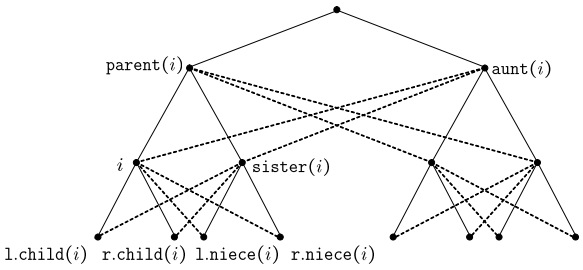


**Fig. 4.2.** CH-GKE

## 4.2 Cross-Help GKE (CH-GKE)

This section presents the basic version of CH-GKE, which tolerates all patterns of party fault if either sister is alive in a binary tree. Parties $P_1$ and $P_2$ in level 1 can cross-help each other, and thus, CH-GKE tolerates even if either party fails after the 1st round.

Figure 4.2 shows the relations of parties in CH-GKE, where a dotted line represents a flow of additional auxiliary elements to BDII. For example, additional auxiliary elements constructed by $\texttt{aunt}(i)$ are sent to $i$ and $\texttt{sister}(i)$.

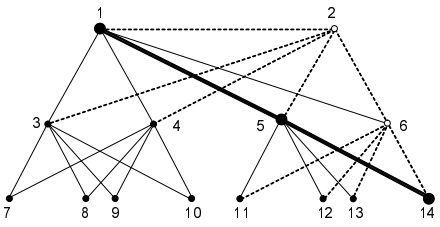


**Fig. 4.3.** Example of CH-GKE

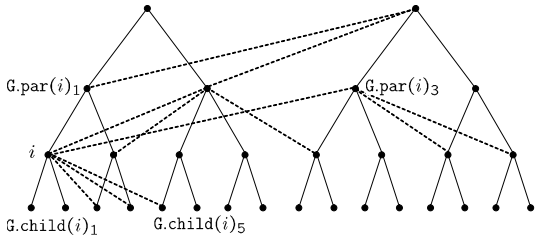


**Fig. 4.4.** Generalized CH-GKE

**Protocol 4 (CH-GKE)**

1. *Each party $P_i$ computes $z_i = g^{r_i}$ for a (private) uniformly and randomly chosen $r_i \in \mathbb{Z}_q^*$ and sends it to its neighbors.*
2. *Let* $\texttt{ActiveList}_1$ *be the list of indices of all parties who complete the 1st round. Then, $P_i$ ($i \in \{1,2\}$) in level 1 computes 5 auxiliary elements $y_{i[\texttt{aunt}(i),\, i]}$, $y_{i[i,\, \texttt{l.child}(i)]}$, $y_{i[i,\, \texttt{r.child}(i)]}$, $y_{i[i,\, \texttt{l.niece}(i)]}$, and $y_{i[i,\, \texttt{r.niece}(i)]}$, and multicasts them to parties in levels $\geq 2$, where*

$$y_{i[\texttt{aunt}(i),\, i]} = \left(\frac{z_{\texttt{aunt}(i)}}{z_i}\right)^{r_i} = g^{r_{\texttt{aunt}(i)} \cdot r_i - r_i^2};$$

$$y_{i[i,\, \texttt{l.child}(i)]} = \left(\frac{z_i}{z_{\texttt{l.child}(i)}}\right)^{r_i} = g^{r_i^2 - r_{\texttt{l.child}(i)} \cdot r_i};\ y_{i[i,\, \texttt{r.child}(i)]} = \left(\frac{z_i}{z_{\texttt{r.child}(i)}}\right)^{r_i} = g^{r_i^2 - r_{\texttt{r.child}(i)} \cdot r_i}$$

$$y_{i[i,\, \texttt{l.niece}(i)]} = \left(\frac{z_i}{z_{\texttt{l.niece}(i)}}\right)^{r_i} = g^{r_i^2 - r_{\texttt{l.niece}(i)} \cdot r_i};\ y_{i[i,\, \texttt{r.niece}(i)]} = \left(\frac{z_i}{z_{\texttt{r.niece}(i)}}\right)^{r_i} = g^{r_i^2 - r_{\texttt{r.niece}(i)} \cdot r_i}.$$

   *Let $P_i$ be an inner-node party in level $\ell_i \geq 2$. Then, $P_i$ computes 6 auxiliary elements $y_{i[\texttt{parent}(i),\, i]}$, $y_{i[\texttt{aunt}(i),\, i]}$, $y_{i[i,\, \texttt{l.child}(i)]}$, $y_{i[i,\, \texttt{r.child}(i)]}$, $y_{i[i,\, \texttt{l.niece}(i)]}$, and $y_{i[i,\, \texttt{r.niece}(i)]}$, and multicasts them to parties in level $\geq \ell_i + 1$, where*

$$y_{i[\texttt{parent}(i),\, i]} = \left(\frac{z_{\texttt{parent}(i)}}{z_i}\right)^{r_i} = g^{r_{\texttt{parent}(i)} \cdot r_i - r_i^2};\ y_{i[\texttt{aunt}(i),\, i]} = \left(\frac{z_{\texttt{aunt}(i)}}{z_i}\right)^{r_i} = g^{r_{\texttt{aunt}(i)} \cdot r_i - r_i^2};$$

$$y_{i[i,\, \texttt{l.child}(i)]} = \left(\frac{z_i}{z_{\texttt{l.child}(i)}}\right)^{r_i} = g^{r_i^2 - r_{\texttt{l.child}(i)} \cdot r_i};\ y_{i[i,\, \texttt{r.child}(i)]} = \left(\frac{z_i}{z_{\texttt{r.child}(i)}}\right)^{r_i} = g^{r_i^2 - r_{\texttt{r.child}(i)} \cdot r_i}$$

$$y_{i[i,\, \texttt{l.niece}(i)]} = \left(\frac{z_i}{z_{\texttt{l.niece}(i)}}\right)^{r_i} = g^{r_i^2 - r_{\texttt{l.niece}(i)} \cdot r_i};\ y_{i[i,\, \texttt{r.niece}(i)]} = \left(\frac{z_i}{z_{\texttt{r.niece}(i)}}\right)^{r_i} = g^{r_i^2 - r_{\texttt{r.niece}(i)} \cdot r_i}.$$

3. *Let* $\texttt{ActiveList}_2$ *be the list of indices of all parties who complete the 2nd round. Then, $P_i$ ($i \in \{1,2\}$) in level 1 computes $K = z_{\texttt{aunt}(i)}^{r_i}$. Note that $P_i$ can compute $K$ even if $P_{\texttt{aunt}}(i) \notin \texttt{ActiveList}_2$. On the other hand, $P_i$ in level $\ell_i \geq 2$ picks up a set of indices from $\texttt{ActiveList}_2$ whose parties form a path from the (reset) parent of $P_i$ to $P_1$ or $P_2$, where the set is denoted by $\underline{\texttt{ancestor}}(i)$: $\underline{\texttt{ancestor}}(i) = \{\underline{\texttt{parent}}(i), \cdots, 1 \text{ or } 2\}$. Actually, the set consists of $\ell_i - 1$ indices of each party from the level $\ell_i - 1$ to 1. If either sister is alive in a binary tree, then $\underline{\texttt{ancestor}}(i)$ exists. A shared key is given as*

$$K = z_{\underline{\texttt{parent}}(i)}^{r_i} \cdot \Pi_{j \in \underline{\texttt{ancestor}}(i)} Y_j = g^{r_1 r_2},$$

   *where $Y_j = y_{j[\underline{\texttt{parent}}(j),\, j]} y_{j[j,\, \underline{\texttt{child}}(j)]} = g^{r_{\underline{\texttt{parent}}(j)} r_j - r_j^2} g^{r_j^2 - r_{\underline{\texttt{child}}(j)} r_j} = g^{r_{\underline{\texttt{parent}}(j)} r_j - r_{\underline{\texttt{child}}(j)} r_j}$ (for $j \neq 1, 2$), $Y_j = y_{j[\texttt{aunt}(j),\, j]} y_{j[j,\, \underline{\texttt{child}}(j)]} = g^{r_{\texttt{aunt}(j)} r_j - r_j^2} g^{r_j^2 - r_{\underline{\texttt{child}}(j)} r_j} = g^{r_{\texttt{aunt}(j)} r_j - r_{\underline{\texttt{child}}(j)} r_j}$ (for $j = 1$ or 2), and $\underline{\texttt{parent}}(j)$ (resp. $\underline{\texttt{child}}(j)$) is the (reset) parent (resp. child) of $j$ in $\underline{\texttt{ancestor}}(i)$.*

Protocol 4 satisfies correctness. Example 1 shows how Party 14 computes a shared key in CH-GKE among 14 parties. See Figure 4.3, where black or white nodes correspond to parties alive or dead in the 2nd round, respectively; big nodes correspond to parties in the path $\underline{\texttt{ancestor}}(14)$; bold edges correspond to the path $\underline{\texttt{ancestor}}(14)$ and dotted lines represent a flow where auxiliary elements have not been sent in the 2nd round.

**Example 1** *Let* $n = 14$; *and* $\texttt{ActiveList}_2 = \{1,3,4,5,7,8,9,10,11,12,13,14\}$. *Then, Party 14 computes a shared key as follows. In this case, the reset parent of* $P_{14}$ *is* $P_5$ *and the reset parent of* $P_5$ *is* $P_1$, *which becomes the end of the path, and thus,* $\underline{\texttt{ancestor}}(14) = \{5,1\}$. *So,* $P_{14}$ *computes* $Y_5 = g^{r_1 r_5 - r_5^2} g^{r_5^2 - r_{14} r_5} = g^{r_1 r_5 - r_5 r_{14}}$; $Y_1 = g^{r_2 r_1 - r_1^2} g^{r_1^2 - r_1 r_5} = g^{r_2 r_1 - r_1 r_5}$; *and thus, results in* $K = z_5^{r_{14}} Y_1 Y_5 = g^{r_1 r_2}$.

**Remarks 1** *1. Parties with low computational resources are arranged to nodes in leaves. Then, they can skip the 2nd round. Those parties executes 1 exponentiation in the round 1 and the computation of the shared key.*
*2. Parties with large computational resources are arranged to inner nodes. They need to execute both the 1st and the 2nd rounds.*

### 4.3 Generalized CH-GKE

We show the generalization of Protocol 4 as Protocol 5. Figure 4.4 shows the relations of parties in the generalized CH-GKE. In Protocol 5, a concept of children and parent of a party in level $\ell_i$ are generalized to $\{\texttt{G.child}(i)_j\}$ and $\{\texttt{G.parent}(i)_j\}$: a party in level $\ell_i$ makes auxiliary elements for $O_i$ parties in level $\ell_i + 1$, denoted by $\texttt{G.child}(i)_1, \cdots, \texttt{G.child}(i)_{O_i}$; and $I_i$ parties in level $\ell_i - 1$, denoted by $\texttt{G.parent}(i)_1, \cdots, \texttt{G.parent}(i)_{I_i}$, make $P_i$'s auxiliary elements, respectively. Remark that the number of $I_i$ is less than that of parties in level $\ell_i - 1$. The generalized CH-GKE tolerates all patterns of party faults if one party in $\{\texttt{G.parent}(i)_j\}_i$ for each party $P_i$ is alive in a binary tree. The detailed protocol is given in the below, which is the same as Protocol 4 except using $\{\texttt{G.child}(i)_j\}$ and $\{\texttt{G.parent}(i)_j\}$.

**Protocol 5 (Generalized CH-GKE)**

1. *Each party* $P_i$ *computes* $z_i = g^{r_i}$ *for a (private) uniformly and randomly chosen* $r_i \in \mathbb{Z}_q^*$ *and sends it to its neighbors.*
2. *Let* $\texttt{ActiveList}_1$ *be the list of indices of all parties who complete the 1st round. Then,* $P_i$ *(*$i \in \{1,2\}$*) computes 5 auxiliary elements* $y_{i[\texttt{parent}(i),\, i]}$, $y_{i[i,\, \texttt{l.child}(i)]}$, $y_{i[i,\, \texttt{r.child}(i)]}$, $y_{i[i,\, \texttt{l.niece}(i)]}$, *and* $y_{i[i,\, \texttt{r.niece}(i)]}$, *and multi-casts them to parties in levels* $\geq 2$. *Let* $P_i$ *be an inner-node party in level* $\ell_i \geq 2$. *Then,* $P_i$ *computes* $I_i + O_i$ *auxiliary elements,* $\{y_{i[\texttt{G.parent}(i)_j,\, i]}\}_{j=1,\cdots,I_i}$ *and* $\{y_{i[i,\, \texttt{G.child}(i)_j]}\}_{j=1,\cdots,O_i}$, *and multicasts them to parties in levels* $\geq \ell_i + 1$, *where*

$$y_{i[\texttt{G.parent}(i)_j,\, i]} = \left(\frac{z_{\texttt{G.parent}(i)_j}}{z_i}\right)^{r_i} = g^{r_{\texttt{G.parent}(i)_j} r_i - r_i^2};\ y_{i[i,\, \texttt{G.child}(i)_j]} = \left(\frac{z_i}{z_{\texttt{G.child}(i)_j}}\right)^{r_i} = g^{r_i^2 - r_{\texttt{G.child}(i)_j} r_i}.$$

3. *Let* $\texttt{ActiveList}_2$ *be the list of indices of all parties who complete the 2nd round. Then,* $P_i$ *(*$i \in \{1,2\}$*) in level 1 computes* $K = z_{\texttt{aunt}(i)}^{r_i}$. $P_i$ *in level* $\ell_i \geq 2$ *picks up a set of indices from* $\texttt{ActiveList}_2$ *whose parties form a path from the (reset) parent of* $P_i$ *to* $P_0$ *or* $P_1$, *where the set is denoted by* $\underline{\texttt{ancestor}}(i)$. *If one of* $\{\texttt{G.parent}(i)_j\}_i$ *is alive in a binary tree, then* $\underline{\texttt{ancestor}}(i)$ *exists. A shared key is given as*

$$K = z_{\underline{\texttt{parent}}(i)}^{r_i} \cdot \Pi_{j \in \underline{\texttt{ancestor}}(i)} Y_j = g^{r_1 r_2},$$

*where* $Y_j = y_{j[\underline{\texttt{parent}}(j),\, j]} y_{j[j,\, \underline{\texttt{child}}(j)]}$ *and* $\underline{\texttt{parent}}(j)$ *and* $\underline{\texttt{child}}(j)$ *are the (reset) parent and child of* $j$ *in* $\underline{\texttt{ancestor}}(i)$, *respectively.*

**Remarks 2** *1. For $P_i$ in level $\ell_i \geq 2$, the number of generalized parents $I_i$ (resp. children $O_i$) needs to be $I_i \leq 2^{\ell_i - 1}$ (resp. $O_i \leq 2^{\ell_i + 1}$) to satisfy correctness.*
*2. The more auxiliary elements $I_i + O_i$ are generated, the more party faults CH-GKE tolerates. However, it needs to locate positions where parties have failed.*

### 4.4 Security of CH-GKE

Theorem 3 that a passive adversary breaks CH-GKE (Protocol 4) is used to solve the Square-DDH Problem.

**Theorem 3.** *Assuming the Square-DDH over $\mathbb{G}$ is hard, CH-GKE (basic) (Protocol 4), denoted simply by $\Pi$, is a secure group GKE protocol. Namely,*

$$\mathtt{Adv}_{\Pi}^{\mathtt{GKE}}(t, q_{ex}) \leq \mathtt{Adv}_{\mathbb{G}}^{\mathtt{Square-DDH}}(t'),$$

*where $\mathtt{Adv}_{\Pi}^{\mathtt{GKE}}(t, q_{ex})$ is an adversary to $\Pi$ with $q_{ex}$*Execute *queries and in $t$ time, and $\mathtt{Adv}_{\mathbb{G}}^{\mathtt{Square-DDH}}(t')$ is an adversary to Square-DDH in $t' = t + q_{ex}(13n - 15)EM$ time for the number of parties $n$.*

**Proof:** Given an algorithm $\mathcal{A}$ against $\Pi$ running in time $t$, we show how to construct an adversary $\mathcal{B}$ against the Square-DDH.

A tuple $(g, y, h) \in \mathbb{G} \times \mathbb{G} \times \mathbb{G}$ is given to $\mathcal{B}$, where $y = g^x$ with unknown $x$ to $\mathcal{B}$ and $h = g^{x^2}$ or a random number in $\mathbb{G}$. Then, $\mathcal{B}$ runs $\mathcal{A}$ to decide whether $h = g^{x^2}$ or not. $\mathcal{B}$ sets $z_1 = y$. Next, choose $c_2, \cdots, c_n \in \mathbb{Z}_p$ randomly, and set $z_i = z_{i-1} \cdot g^{-c_i} = g^{x - \sum_{j=2}^{i} c_j}$ for $i \geq 2$. $z_i$ can be computed since $z_{i-1}$ was computed before. Note that $\mathcal{B}$ knows $z_i$ but does not know the logarithm $r_i$ of $z_i = g^{r_i}$, where $r_i = x - \sum_{j=2}^{i} c_j$. From this, $\mathcal{B}$ can compute $y_{i[\mathtt{parent}(i),\, i]}$, $y_{i[\mathtt{aunt}(i),\, i]}$, $y_{i[i,\, \mathtt{l.child}(i)]}$, $y_{i[i,\, \mathtt{r.child}(i)]}$, $y_{i[i,\, \mathtt{l.niece}(i)]}$, and $y_{i[i,\, \mathtt{r.niece}(i)]}$ as follows.

In the case of $P_1$, set

$$y_{i[\mathtt{aunt}(1),\, 1]} = \left(\frac{z_{\mathtt{aunt}(1)}}{z_1}\right)^{r_1} = g^{r_2 r_1 - r_1^2} = g^{(x - c_2)x - x^2} = y^{-c_2}$$

which is computable since $c_2$ is known to $\mathcal{B}$. Let us discuss the other 4 auxiliary elements. Set the number of $\mathtt{l.child}(1)$ to $\mathtt{num.lc}(1)$. Then, $\mathtt{num.lc}(1) > 2$ holds and

$$y_{1[1,\, \mathtt{l.child}(1)]} = \left(\frac{z_1}{z_{\mathtt{l.child}(1)}}\right)^{r_1} = g^{r_1^2 - r_{\mathtt{l.child}(1)} r_1} = g^{x^2 - (x - \sum_{j=2}^{\mathtt{num.lc}(1)} c_j)x} = g^{(\sum_{j=2}^{\mathtt{num.lc}(1)} c_j)x} = y^{\sum_{j=2}^{\mathtt{num.lc}(1)} c_j},$$

which is computable since $\sum_{j=2}^{\mathtt{num.lc}(1)} c_j$ is known to $\mathcal{B}$. The computation of the other 3 auxiliary elements follows the above.

In the case of $P_2$, set

$$y_{2[\mathtt{aunt}(2),\, 2]} = \left(\frac{z_{\mathtt{aunt}(2)}}{z_2}\right)^{r_2} = g^{r_1 r_2 - r_2^2} = g^{x(x - c_2) - (x - c_2)^2} = y^{c_2} g^{-c_2^2},$$

which is computable since $c_2$ is known to $\mathcal{B}$. The other 4 auxiliary elements can be computed in the same way.

In the case of $P_i (i \geq 3)$, $\mathcal{B}$ can compute 6 auxiliary elements in the same way as above. Let us discuss two auxiliary elements of $y_{i[\mathtt{parent}(i),\, i]}$ and $y_{i[\mathtt{aunt}(i),\, i]}$. Set the number of $\mathtt{parent}(i)$ to $\mathtt{num.par}(i)$. Then, $\mathtt{num.par}(i) < i$ holds and

$$y_{i[\mathtt{parent}(i),\, i]} = \left(\frac{z_{\mathtt{parent}(i)}}{z_i}\right)^{r_i} = g^{r_{\mathtt{parent}(i)} r_i - r_i^2} = g^{(x - \sum_{j=2}^{\mathtt{num.par}(i)} c_j)(x - \sum_{j=2}^{i} c_j) - (x - \sum_{j=2}^{i} c_j)^2}$$
$$= y^{-\sum_{j=\mathtt{num.par}(i)+1}^{i} c_j} g^{(\sum_{j=\mathtt{num.par}(i)+1}^{i} c_j)(\sum_{j=2}^{i} c_j)}$$

which is computable since $\forall c_j$ is known to $\mathcal{B}$. $y_{i[\mathtt{aunt}(i),\, i]}$ can be computed in the same way as above. Let us discuss the other 4 auxiliary elements. Set the number of $\mathtt{l.child}(i)$ to $\mathtt{num.lc}(i)$. Then, $\mathtt{num.lc}(i) > i$ holds and

$$y_{i[i,\, \mathtt{l.child}(i)]} = \left(\frac{z_i}{z_{\mathtt{l.child}(i)}}\right)^{r_i} = g^{r_i^2 - r_{\mathtt{l.child}(i)} r_i} = g^{(x - \sum_{j=2}^{i} c_j)^2 - (x - \sum_{j=2}^{\mathtt{num.lc}(i)} c_j)(x - \sum_{j=2}^{i} c_j)}$$
$$= y^{-\sum_{j=i+1}^{\mathtt{num.lc}(i)} c_j} g^{(\sum_{j=2}^{i} c_j)(\sum_{j=i+1}^{\mathtt{num.lc}(i)} c_j)}$$

which is computable since $\forall c_j$ is known to $\mathcal{B}$. The computation of the other 3 auxiliary elements follows the above.

As the $c_i (i \geq 2)$ are distributed uniformly at random, the distribution of $z_i$ and $y_{i[i,\, j]}$ is identical to that in $\Pi$. The transcript consists of

$$T = \{z_i,\, y_{i[\mathtt{parent}(i),\, i]},\, y_{i[\mathtt{aunt}(i),\, i]},\, y_{i[i,\, \mathtt{l.child}(i)]},\, y_{i[i,\, \mathtt{r.child}(i)]},\, y_{i[i,\, \mathtt{l.niece}(i)]},\, y_{i[i,\, \mathtt{r.niece}(i)]}\},$$

for each party $P_i$. Let $\mathtt{ActiveList}_2$ be the list of indices of all parties who complete the 2nd round. Upon the Test request, $\mathcal{B}$ issues the shared key $K$ as follows,

$$K = g^{r_1 r_2} = g^{x(x - c_2)} = g^{x^2} y^{-c_2} = h y^{-c_2}.$$

If $K$ is the shared group key, then $h = g^{x^2}$, i.e. $(g, y, h)$ is a valid Square-DDH set. Therefore, $\mathcal{B}$ succeeds with the same advantage as $\mathcal{A}$ by $(13n - 15)EM$ additional computational time to generate $T$.

We consider the case in which $\mathcal{A}$ makes a single Execute query, since $\mathcal{B}$ can easily generate another set of $(g^r, y^r, h^{r^2})$ of the same type as $(g, y, h)$ for a random exponent $r \in \mathbb{Z}_p$. Bounding the number $n$ by the total number of parties, the claim follows. ∎

In the same way, we can show that a passive adversary that breaks the generalized CH-GKE (Protocol 5) is used to solve the Square-DDH Problem, whose proof will be shown in the final paper.

**Theorem 4.** *Assuming the Square-DDH over* $\mathbb{G}$ *is hard, CH-GKE (general) (Protocol 5), denoted simply by* $\Pi$*, is a secure group GKE protocol. Namely,*

$$\mathtt{Adv}_{\Pi}^{\mathtt{GKE}}(t, q_{ex}) \leq \mathtt{Adv}_{\mathbb{G}}^{\mathtt{Squre\text{-}DDH}}(t'),$$

*where* $\mathtt{Adv}^{\mathtt{GKE}}_{\Pi}(t, q_{ex})$ *is an adversary to* $\Pi$ *with* $q_{ex}\mathsf{Execute}$ *queries and in* $t$ *time;* $\mathtt{Adv}^{\mathtt{Squre-DDH}}_{\mathrm{G}}(t')$ *is an adversary to Square-DDH in* $t' = t + q_{ex}((2\mathsf{max}_{\Pi} + 1)n + 9 - 4\mathsf{max}_{\Pi})EM$ *time for the number of parties* $n$*; and* $\mathsf{max}_{\Pi}$ *is the maximum number of auxiliary elements constructed by a single party.*

## 5 $T$-Robust GKE with $O(\log n)$ complexity

CH-GKE, shown in Section 4, achieves robustness with $O(\log n)$ message size. The generalized CH-GKE tolerates if one of paths from any alive party to $P_1$ or $P_2$ is alive. However, it needs to locate positions where parties have failed. In fact, either $P_1$ or $P_2$ need to be alive. On the other hand, JKT tolerates any $T$-party faults in any position, however, it needs $O(n)$ computational complexity.

In this section, we present $T$-robust GKE with $O(\log n)$ communicational and computational complexity. Our idea is to combine both JKT and CH-GKE considering their advantages: JKT is a symmetric structure and tolerates any $T$-party fault in any position but works in $O(n)$ computational complexity, while CH-GKE is an asymmetric structure and works in $O(\log n)$ communicational and computational complexity. Our strategies to combine both $T$-robust JKT and CH-GKE are: from the point of any $T$-party fault, the shared key is computed in the procedure of $T$-robust JKT. Then, auxiliary elements for parties not in JKT procedure to compute the shared key are generated and multicasted in the procedure of CH-GKE. According to this strategy, parties are arranged to a circle of JKT or trees of CH-GKE; and compute JKT-and-CH-GKE-like auxiliary elements or CH-GKE-like auxiliary elements, respectively. Figure 5.1 shows an arrangement of parties, where parties in a circle execute JKT part, parties in trees execute CH-GKE part, dotted lines in the circle represent a flow that the party 1 computes auxiliary elements in JKT part, and bold lines from the party 1 to nodes in level 1 of trees represent a flow that the party 1 computes auxiliary elements in CH-GKE part. Let us show how we combine JKT and CH-GKE briefly, then present our $T$-robust GKE (Protocol 6):

1. *JKT part*
   $T + 2$ parties are arranged in the circle of JKT[4]. They compute auxiliary elements to execute $T$-robust JKT, which are in total for $T + 1$ parties. They also compute ones to execute CH-GKE, which are in total for $2(T+1)$ parties, because any party in level $\ell_i$ of CH-GKE needs to get auxiliary elements sent by $T + 1$ different parties in level $\ell_i - 1$, while the number of parties in level $\ell_i$ is twice as large as one in level $\ell_i - 1$. Those auxiliary elements are called JKT-and-CH-GKE-like auxiliary elements.
2. *CH-GKE part*
   Arrange $T + 2$ trees of CH-GKE under each party in the circle of JKT. $\frac{n-(T+2)}{T+2}$ parties are arranged in each tree of CH-GKE, where the height of tree is $\ell = \lceil \log_2 \frac{n}{T+2} + 1 \rceil - 1$. They execute CH-GKE and compute auxiliary elements for $2(T + 1)$ parties in total in the same reason as above.

[4] This is the smallest number of parties arranged in the circle, which can be set to more than $T + 2$.

**Protocol 6 ($T$-robust GKE)** *$T$-robust GKE among $n$ parties is a combination of $T$-robust JKT among $T+2$ parties and $T$-robust CH-GKE among $n-T-2$ parties.*

Initialization: Arrange parties to JKT or CH-GKE

*Set $T+2$ parties in the circle in JKT, where they are numbered from 1 to $T+2$. Arrange $T+2$ trees of CH-GKE by setting each party in the circle of JKT to each root. Finally, set $\frac{n-T-2}{T+2}$ parties in each tree.*

Group Key Exchange Protocol

1. *Each party $P_i$ computes $z_i = g^{r_i}$ for a (private) uniformly and randomly chosen $r_i \in \mathbb{Z}_q^*$ and sends it to its neighbors.*
2. *Let $\mathtt{ActiveList}_1^{JKT}$ (resp. $\mathtt{ActiveList}_1^{CH}$) be the list of indices of parties in the circle (resp. trees) who complete the 1st round.*

   Parties in the circle of JKT: *Let $P_i$ be a party in the circle, who has generalized children $\{\mathtt{G.child}(i)_j\}$ in trees. $P_i$ computes and broadcasts two types of auxiliary elements: auxiliary elements $x_{[k,\,i]}$ seen in the 2nd round in JKT(Protocol 2) and those seen in children parts of the 2nd round in CH-GKE between $P_i$ itself and generalized $2(T+1)$ children in the tree, $P_{\mathtt{G.child}(i)_j}$ (see Section 4.3 for $\mathtt{G.child}(i)_j$):*

$$x_{[k,\,i]} = \left(\frac{z_i}{z_k}\right)^{r_i} = g^{r_i^2 - r_k r_i}\ (\mathtt{ActiveList}_1^{JKT} \ni k \neq i)$$
$$y_{i[i,\,\mathtt{G.child}(i)_j]} = \left(\frac{z_i}{z_{\mathtt{G.child}(i)_j}}\right)^{r_i} = g^{r_i^2 - r_i r_{\mathtt{G.child}(i)_j}}\ (\mathtt{ActiveList}_1^{CH} \ni \mathtt{G.child}(i)_j).$$

   Parties in trees of CH-GKE: *Let $P_i$ be an inner-node party in level $\ell_i$ in a tree, who has generalized parents $\{\mathtt{G.parent}(i)_j\}$ in the circle (resp. trees) if $\ell_i = 1$ (resp. $\ell_i \geq 2$) and generalized children $\{\mathtt{G.child}(i)_j\}$ in trees. $P_i$ computes $(T+1)+2(T+1)$ auxiliary elements and multi-casts them to parties in level $\geq \ell_i + 1$ in the same way as in Protocol 5:*

$$y_{i[\mathtt{G.parent}(i)_j,\,i]} = \left(\frac{z_{\mathtt{G.parent}(i)_j}}{z_i}\right)^{r_i} = g^{r_{\mathtt{G.parent}(i)_j} r_i - r_i^2};\ y_{i[i,\,\mathtt{G.child}(i)_j]} = \left(\frac{z_i}{z_{\mathtt{G.child}(i)_j}}\right)^{r_i} = g^{r_i^2 - r_{\mathtt{G.child}(i)_j} r_i}.$$

3. *Let $\mathtt{ActiveList}_2^{JKT}$ (resp. $\mathtt{ActiveList}_2^{CH}$) be the list of indices of parties in the circle (resp. trees) who complete the 2nd round. Here we set $\#\mathtt{ActiveList}_2^{JKT} = T'$. Assume that the alive parties in $\mathtt{ActiveList}_2^{JKT}$ are sorted in the same order as before and ordered $\{P_{a_1}, P_{a_2}, \cdots, P_{a_{T'}}\}$.*

   Parties in the circle of JKT: *Each $P_{a_i}$ in the circle computes a shared key*

$$K = z_{a_{i-1}}^{r_{a_i} \cdot T'} \cdot X_{a_i}^{T'-1} \cdot X_{a_{i+1}}^{T'-2} \cdots X_{a_{i-2}} = g^{r_{a_1} r_{a_2} + r_{a_2} r_{a_3} + \cdots + r_{a_{T'}} r_{a_1}},$$

   *where $X_{a_j} = x_{[a_{j-1},\,a_j]} \cdot (x_{[a_{j+1},\,a_j]})^{-1} = g^{r_{a_j} r_{a_{j+1}} - r_{a_{j-1}} r_{a_j}}$ $(j \in \{1, \cdots, T'\})$.*

   Parties in trees of CH-GKE: *Each $P_i$ in level 1 picks up a party $P_{a_i} \in \mathtt{ActiveList}_2^{JKT}$ whose corresponding auxiliary element, $y_{a_i[a_i,\,i]}$, has been sent to $P_i$. A shared key is given as follows:*

$$K = (K')^{T'} \cdot X_{a_i}^{T'-1} \cdot X_{a_{i+1}}^{T'-2} \cdots X_{a_{i-2}},$$

*where* $K' = z_{a_i}^{r_i} \cdot y_{a_i[a_i,\, i]} \cdot (x_{[a_{i+1},\, a_i]})^{-1} = g^{r_{a_i} r_{a_{i+1}}}$.
*Each* $P_i$ *in level* $\ell_i \geq 2$*, first, picks up a set of indices from* $\mathtt{ActiveList}_2^{CH}$ $P_i$ *whose parties form a path from the (reset) parent of* $P_i$ *to the (reset) ancestor in level 1. The set is denoted by* $\underline{\mathtt{ancestor}}(i)$ *in the same way as Protocols 4 and 5. We also denote the index of the party in level 1 in* $\underline{\mathtt{ancestor}}(i)$ *by* $\underline{\mathtt{ancestor}}(i)[1]$*. Then,* $\underline{\mathtt{ancestor}}(i) = \{\underline{\mathtt{parent}}(i), \cdots, \underline{\mathtt{ancestor}}(i)[1]\}$*. If the number of fault parties is less than or equal to* $T$*, then* $\underline{\mathtt{ancestor}}(i)$ *exists. Next,* $P_i$ *picks up a party* $P_{a_i} \in \mathtt{ActiveList}_2^{JKT}$ *whose corresponding auxiliary element,* $y_{a_i[a_i,\, \underline{\mathtt{ancestor}}(i)[1]]}$*, has been sent to* $P_{\underline{\mathtt{ancestor}}(i)[1]}$*. A shared key is given as follows:*

$$K = (K')^{T'} \cdot X_{a_i}^{T'-1} \cdot X_{a_{i+1}}^{T'-2} \cdots X_{a_{i-2}},$$

*where* $K' = z_{\underline{\mathtt{parent}}(i)}^{r_i} \cdot \left(\Pi_{j \in \underline{\mathtt{ancestor}}(i)} Y_j\right) \cdot y_{a_i[a_i,\, \underline{\mathtt{ancestor}}(i)[1]]} \cdot (x_{[a_i+1,\, a_i]})^{-1} = g^{r_{a_i} r_{a_{i+1}}}$ *and* $Y_j = g^{r_{\underline{\mathtt{parent}}(j)} r_j - r_{\underline{\mathtt{child}}(j)} r_j}$.

Protocol 6 satisfies correctness. Example 2 shows how Party 28 computes a shared key in 5-robust GKE among 49 parties. See Figure 5.2, where black or white nodes correspond to parties alive or dead in the 2nd round, respectively; big nodes correspond to parties in the path $\underline{\mathtt{ancestor}}(28)$; bold edges correspond to the path of $\underline{\mathtt{ancestor}}(28)$ and $P_{a_i} \in \mathtt{ActiveList}_2^{JKT}$; and bold dotted edges correspond to the path of $\mathtt{ActiveList}_2^{JKT}$, where a shared key is computed.

**Example 2** *Let* $S$ *be a set of parties with* $\#S = n = 49$*, and* $F$ *be a set of fault parties, where* $F = \{1, 2, 5, 7, 9, 10, 11, 24, 31\}$*. Then* $\mathtt{ActiveList}_2^{JKT} = \{3, 4, 6\}$, $T' = 3$*, and a shared key* $K$ *is computed to* $K = g^{r_3 r_4 + r_4 r_6 + r_6 r_3}$*. Party 28 computes the shared key as follows. The reset parent of* $P_{28}$ *is* $P_8$ *and the reset parent of* $P_8$ *is* $P_3$*, which becomes the end of the path, and thus,* $\underline{\mathtt{ancestor}}(28) = \{8\}$*.* $P_{28}$ *picks up a party* $P_3 \in \mathtt{ActiveList}_2^{JKT}$*, and computes* $K' = (z_{\underline{\mathtt{parent}}(28)})^{r_{28}} \cdot Y_8 \cdot y_{3[3,\, 8]} \cdot x_{[6,\, 3]}^{-1} = g^{r_8 r_{28}} \cdot g^{r_3 r_8 - r_8 r_{28}} \cdot g^{r_3^2 - r_3 r_8} \cdot g^{r_6 r_3 - r_3^2} = g^{r_6 r_3}$ *for* $Y_8 = y_{8[3,\, 8]} \cdot y_{8[8,\, 28]} = g^{r_3 r_8 - r_8 r_{28}}$*, and thus, results in* $K = (K')^3 \cdot X_3^2 \cdot X_4 = g^{3(r_6 r_3)} \cdot g^{2(r_3 r_4 - r_6 r_3)} \cdot g^{r_4 r_6 - r_3 r_4} = g^{r_3 r_4 + r_4 r_6 + r_6 r_3}$.

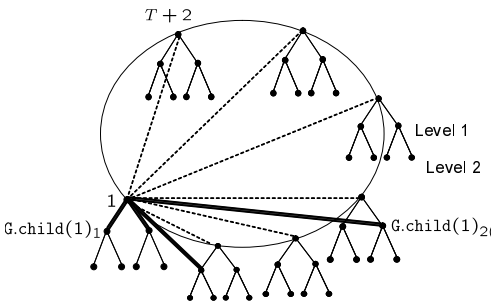


**Fig. 5.1.** $T$-robust GKE

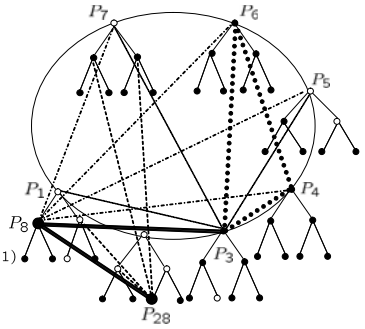


**Fig. 5.2.** Example of 5-robust GKE among 49 parties

The security of Protocol 6 is given in Theorem 5, whose proof will be shown in the final paper

**Theorem 5.** *Assuming the DDH and Square-DDH over $\mathbb{G}$ are hard, Protocol 6, denoted simply by $\Pi$, is a secure group GKE protocol.*

## 6 Comparison

This section compares our scheme with previous schemes from the view point of efficiency and robustness. Table 1 (resp. Table 2) summarizes the communicational (resp. computational) complexity per user and robustness of ours (Section 5), JKT, R-TDH1 (basic)[5], BDI, and BDII. Note that, $T$-robust GKE means GKE tolerates all patterns of party faults up to $T$. JKT (cycle) or JKT (path) means Hamilton-cycle JKT or Hamilton-path JKT, respectively (See Section 3 for the detailed differences). When we do not have to distinguish JKT (cycle) from JKT (path), JKT is used simply. The notation of $\mathbf{p}$ and $\mathbf{b}$ is defined in Section 2.2. In an asymmetric party setting seen in BDII and ours, parties can be in different environment and have different computational resources since efficiency is different to each party type. In such GKEs, comparison is done by parties with large or low computational resources. Here after we focus on efficiency of JKT and our GKE since these are the same paradigm, while neither BDI nor BDII is robust and R-TDH1 is fully robust.

Our GKE has advantages over JKT in the received message complexity for any party type and computational complexity for parties with low computational resources. In fact, the size of our received message is $O(T + \log n)$, while that of JKT is $O(n)$. As for sent message complexity, ours is slightly worse than that of JKT. On the other hand, the order of our computational complexity for parties with low computational resources is $O(T + \log n)$, while that of JKT is $O(T + n)$, due to our scalable party arrangement.

Let us compare both our GKE and JKT by using concrete parameter of $(n, T)$. We firstly demonstrate the received message size comparison in Figures 6.1 and 6.2. Figure 6.1 simulates the case that the ratio of fault parties, $T/n$, is fixed to 0.1 and group size changes from $10^4$ to $10^7$. In any case, our GKE has better performance than JKT (path)[6]. Figure 6.2 simulates the case that the group size is fixed to $n = 10^7$, and $T/n$ changes from 0.1 to 0.8. When $T/n < 0.67$, received message size of our GKE for parties with large resources is better than that of JKT. As for parties with low resources, our GKE is more efficient than JKT in any $T/n$.

We next compare them in the computational complexity, shown in Figures 6.3 and 6.4. Note that, the computational complexity is estimated with complexity of a single multiplication on $\mathbb{G}$ ($M$), where estimation is done by: $|\mathbb{G}| = 1,024$ bits, $EM = 1,536M$, and $I = 30M$[7]. Figure 6.3 simulates the case that $T/n$ is

[5] In our comparison, R-TDH1 in [3] is simplified to only key establishment for the estimation of its basic complexity.

[6] JKT (path) is slightly better than JKT (cycle) in the received message size, and, thus, only JKT (path) is simulated.

[7] The basic binary method is assumed for an exponentiation in $\mathbb{G}$.

**Table 1.** Sent/received message complexity of several GKEs among $n$ parties

| Party Type | Sent Messages (Large / Low) | Received Message (Large / Low Computational Resources) |
|---|---|---|
| BDI | $\mathbf{b} + 2\mathbf{p}$ | $(n-1)\mathbf{b} + 2\mathbf{p}$ |
| BDII | $2\mathbf{b} + 3\mathbf{p}$ / $\mathbf{p}$ | $\log_2 n\mathbf{b} + 3\mathbf{p}$ / $\log_2 n\mathbf{b} + \mathbf{p}$ |
| JKT(cycle) | $2(T+1)\mathbf{b} + 2(T+1)\mathbf{p}$ | $2(n-1)\mathbf{b} + 2(T+1)\mathbf{p}$ |
| JKT(path) | $2T\mathbf{b} + 2T\mathbf{p}$ | $2(n-2)\mathbf{b} + 2T\mathbf{p}$ |
| R-TDH1 | $n\mathbf{b}$ / $\mathbf{b}$ | $2(n-1)\mathbf{b}$ / $\mathbf{b}$ |
| ours | $3(T+1)\mathbf{b} + 3(T+1)\mathbf{p}$ / $(T+1)\mathbf{p}$ | $2(T+\log_2 n+1)\mathbf{b} + 3(T+1)\mathbf{p}$ / $2(T+\log_2 n+1)\mathbf{b} + (T+1)\mathbf{p}$ |

**Table 2.** Computational complexity and robustness(max faults) of GKE among $n$ parties

| Party Type | Large Computational Resources | | | Low Computational Resources | | | Robustness |
|---|---|---|---|---|---|---|---|
| | $\#EM$ | $\#I$ | $\#M$ | $\#EM$ | $\#I$ | $\#M$ | |
| BDI | 3 | 1 | $2(n-1)$ | 3 | 1 | $2(n-1)$ | 0 |
| BDII | 4 | 2 | $\log_2 n$ | 2 | 0 | $\log_2 n$ | 0 |
| JKT(cycle) | $2(T+2)$ | 2 | $6n+8T-4$ | $2(T+2)$ | 2 | $6n+8T-4$ | $T$ |
| JKT(path) | $2(T+1)$ | 2 | $4(3n+T-6)$ | $2(T+1)$ | 2 | $4(3n+T-6)$ | $T$ |
| R-TDH1 | $3(n-1)$ | 0 | 0 | 4 | 0 | 0 | $n-2$ |
| ours | $3T+5$ | 3 | $18T+9$ | 2 | 2 | $6T+2\log_2 n+5$ | $T$ |

fixed to 0.1 and group size changes from $10^4$ to $10^7$. In any case computational complexity of our GKE for parties with low resources is smaller than that of JKT. That for parties are larger than that of JKT. Figure 6.4 simulates the case that the group size $n$ is fixed to $n = 10^7$ and $T/n$ changes from 0.002 to 0.01. In the same way as Figure 6.3, the computational complexity for parties with low resources in our GKE is extremely reduced than that of JKT. That for parties with large resources is slightly better than JKT in the range of $T/n < 0.004$.

The above evaluations can be summarized as follows: (i) From the view point of received message complexity, our protocol has an advantage over JKT in received message complexity for parties with low computational resources, while JKT has an advantage over ours in sent message complexity (although both are $O(T)$). (ii) From the view point of computational complexity, our protocol has an advantage over JKT for parties with low computational resources, while JKT has an advantage over our GKE for parties with large resources. (iii) Our GKE has very nice scalability in both computational and communicational complexity and, thus, can be available to parties with relatively low CPU or battery.

Finally, we show the optimal robust GKE for given parameter $(n, T)$, seen in Figures 6.5 and 6.6. Note that, our GKE is more efficient than JKT (cycle) in even received message size in any range of $(n, T/n)$. Thus, our GKE is compared with only JKT (path) from the point of view of received message size and computational complexity. JKT (path) has smaller received message size and computational complexity than our GKE for parties with large resources only if $T/n$ is rather high. Note that, in any range, received message size and computational complexity of our GKE for parties with low resources is smaller

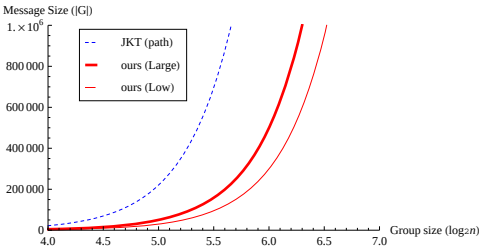


**Fig. 6.1.** Received message size ($T/n = 0.1$)

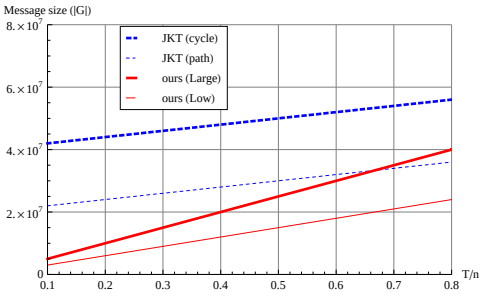


**Fig. 6.2.** Received message size ($n = 10^7$)

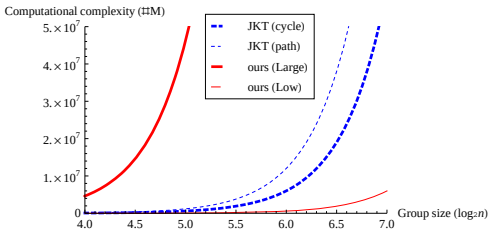


**Fig. 6.3.** Computational complexity ($T/n = 0.1$)

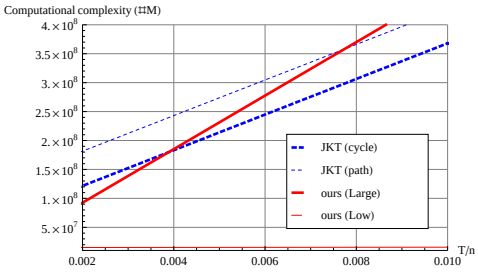


**Fig. 6.4.** Computational complexity ($n = 10^7$)

than those of JKT. By using our results, we can choose the optimal $T$-robust GKE for given $(n, T/n)$.

## 7 Conclusions

JKT was developed to achieve a robust GKE based on BDI, and thus, it suffers communicational and computational complexity $O(n)$ per party for the group size $n$. Another robust GKE [3] also suffers communicational complexity $O(n^2)$ although it satisfies fully robustness. Note that, up to now, GKE with communicational and computational complexity $O(\log n)$ does not have any robustness.

We have proposed a new robust GKE, CH-GKE, with communicational and computational complexity $O(\log n)$. We have also shown that our robust GKE is secure in the standard model under the Square-DDH assumption. By combining both CH-GKE and JKT, we have proposed $T$-robust GKE with communicational and computational complexity $O(\log n)$, which tolerates any $T$-party fault in any position.

## References

1. Y. Amir, Y. Kim, C. Nita-Rotaru, J. Schultz, J. Stanton, and G. Tsudik, "Exploring robustness in group key agreement", *In ICDCS'01*, 399-409, IEEE CS, 2001.

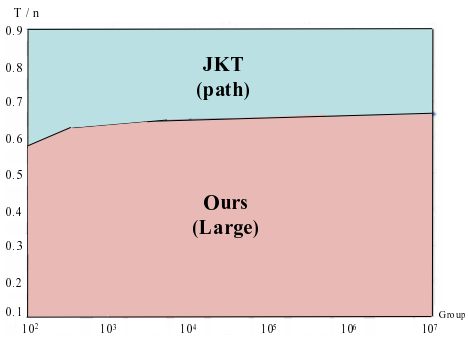


**Fig. 6.5.** Optimal protocol (received message size)

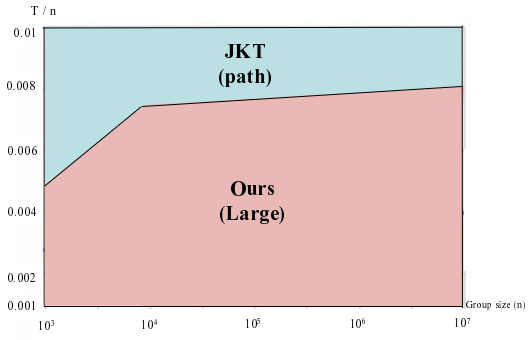


**Fig. 6.6.** Optimal protocol (computational complexity)


2. A. Abdel-Hafez, A. Miri, and L. Orozco-Barbosa, "Authenticated group key agreement protocols for ad hoc wireless networks", International Journal of Network Security, Vol.4, No.1, 90-98, 2007.
3. T. Brecher, E. Bresson, and M. Manulis, "Fully Robust Tree-Diffie-Hellman Group Key Exchange", *In CANS'09*, LNCS **5888**(2009), 478-497, Springer-Verlag.
4. M. Burmester and Y. Desmedt, "A secure and efficient conference key distribution system", *In Eurocrypt'94*, LNCS **950**(1994), 275-286, Springer-Verlag.
5. M. Burmester and Y. Desmedt, "Efficient and secure conference key distribution", *In Security Protocols*, LNCS **1189**(1997), 119-130, Springer-Verlag.
6. C. Cachin and R. Strobl, "Asynchronous group key exchange with failures", *In Proceedings of PODC'04*, 357-366, ACM press, 2004.
7. Y. Desmedt, T. Lange and M. Burmester, "Scalable authenticated tree based group key exchange for ad-hoc groups", *In FC'07*, LNCS **4886**(2007), 104-118, Springer-Verlag.
8. Y. Desmedt and A. Miyaji, "Redesigning Group Key Exchange Protocol based on Bilinear Pairing Suitable for Various Environments", *Inscrypt 2010*, Springer-Verlag, to appear.
9. S. Jarecki, J. Kim and G. Tsudik, "Robust Group Key Agreement Using Short Broadcast", *In Proceedings of ACM CCS 2007*, 411-420, ACM 2007.
10. Y. Kim, A. Perrig, and G. Tsudik, "Group Key Agreement Efficient in Communication", *IEEE Trans. on Comp*, Vol. 53 (No.7), 905-921, 2004.
11. J. Katz and M. Yung, "Scalable Protocols for Authenticated Group Key Exchange", *In CRYPTO 2003*, LNCS **2729**(2003), 110-125, Springer-Verlag.
12. E. Konstantinou, "Cluster-based group key agreement for wireless ad hoc networks", *In Proceedings of ARES 2008*, 550 - 557.