Survival in the Wild:

Robust Group Key Agreement in Wide-Area Networks

Jihye Kim* and Gene Tsudik

University of California, Irvine

{jihyek,gts}@ics.uci.edu

May 7, 2008

Abstract

Group key agreement (GKA) allows a set of *players* to establish a shared secret and thus bootstrap secure group communication. GKA is very useful in many types of peer group scenarios and applications. Since all GKA protocols involve multiple rounds, robustness to player failures is important and desirable. A *robust* group key agreement (RGKA) protocol runs to completion even if some players fail during protocol execution.

Previous work yielded constant-round RGKA protocols suitable for the LAN setting, assuming players are homogeneous, failure probability is uniform and player failures are independent. However, in a more general wide-area network (WAN) environment, heterogeneous hardware/software and communication facilities can cause wide variations in failure probability among players. Moreover, congestion and communication equipment failures can result in correlated failures among subsets of GKA players.

In this paper, we construct the first RGKA protocol that supports players with different failure probabilities, spread across any LAN/WAN combination, while also allowing for correlated failures among subgroups of players. The proposed protocol is efficient (2 rounds) and provably secure. We evaluate its robustness and performance both analytically and via simulations.

Keywords: Group Key Agreement, Fault Tolerance, Robustness, Wide-Area Networks, Heterogeneous Players

Submission Type: Regular Paper

*Contact Author.

1 Introduction

The last decade has witnessed a sharp spike in the popularity of collaborative applications, such as multi-media conferencing, distributed simulations, multi-user games and replicated servers. Such application often operate across the insecure and unstable "wilderness" of the global Internet. To be effective, collaborative applications need robust and secure communication. However, basic security services (such as confidentiality, integrity and authentication) require key management as the foundation.

A number of group key management techniques have been proposed. They generally fall into three categories: 1) centralized, 2) distributed and 3) contributory.

Centralized group key management involves a single entity that generates and distributes keys to group members via a pair-wise secure channel established with each group member. This is generally inappropriate for secure peer group communication, since a central key server must be, continuously available and present in every possible subset of a group in order to support continued operation in the event of arbitrary network partitions. Continuous availability can be addressed by using fault-tolerance and replication techniques. Unfortunately, the omni-presence issue is difficult to solve in a scalable and efficient manner.[1]

Distributed group key management is more suitable to peer group communication over unreliable networks. It involves dynamically selecting a group member that acts as a key distribution server. Although robust, this approach has a notable drawback in that it requires a key server to maintain long-term pairwise secure channels with all current group members in order to distribute group keys. Some schemes take advantage of data structures to minimize the number of encryption operations and messages generated whenever the group key changes. When a new key server is selected all these data structures need to be constructed.

In contrast, *contributory* group key agreement requires every group member to contribute an equal share to the common group secret, computed as a function of all members' contributions. This is particularly appropriate for dynamic peer groups since it avoids the problems with the single point(s) of trust and failure. Moreover, some contributory methods do not require establishing pairwise secret channels among group members. Also, unlike most group key distribution protocols, they offer strong key management security properties such as key independence and perfect forward secrecy (PFS) [11]. More detailed discussion can be found in [10].

In the rest of this paper we focus on *contributory* group key agreement and refer to it as GKA from here on.

[1] However, that the centralized approach works well in one-to-many multicast scenarios since a trusted third party (TTP) placed at, or very near, the source of communication, can support continued operation within an arbitrary partition as long as it includes the source.

1.1 Prior Work on Robust GKA

Most early work in GKA focused on security and efficiency. A number of basic protocols were proposed , notably: STR [10, 14], BD [4], GDH [15] and TGDH [9]. All of these protocols are provably secure with respect to passive adversaries (eavesdroppers). Each protocol is efficient in its own way, while none is efficient in all respects (e.g., number of messages, rounds and cryptographic operations). To protect against active adversaries, *authenticated* versions of the above protocols were later constructed, e.g., [2, 3, 8].

All current GKA protocols involve multiple communication rounds. Since no one-round GKA has ever been proposed, the issue of robustness applies to all current GKA protocols; in fact, none of them is inherently robust. In this context, robustness means the ability to complete the protocol despite player and/or communication failures during protocol execution.

The robustness issue has been identified several years ago. In 2001, Amir, et al. [1] proposed the first robust GKA (RGKA) technique based on a (non-robust) group key agreement protocol (called GDH) by Steiner, et al. [15], and a view-based group communication system (GCS) which provides the abstraction of consistent group membership. Since the GCS can detect crashes among players during the execution of GKA, the protocol can react accordingly. However, its round complexity is $O(n)$ and it requires $O(n^2)$ broadcasts where n is the number of players.

Subsequently, Cachin and Strobl (CS) proposed a very different constant-round RGKA technique operating over asynchronous networks [5]. It tolerates both player and link failures. The exact communication and infrastructure assumptions of the CS protocol depend on the choice of the consensus sub-protocol which the CS protocol invokes. However, assuming a reliable broadcast channel, the CS protocol takes 2 rounds, each player broadcasts $O(n)$-sized messages and performs $O(n)$ public key operations.[2]

More recently, Jarecki, et al. [7] proposed a 2-round RGKA protocol (called JKT) which operates over a reliable broadcast channel, and tolerates up to $O(T)$ player failures using $O(T)$-sized messages, for any $T < n$. It achieves a natural trade-off between message size and desired level of fault-tolerance. However, JKT assumes that: (1) every player has the same fault probability, and (2) all fault probabilities are random and independent.

1.2 Starting Point

The JKT protocol is well-suited for a local area network (LAN) environment. This is because the assumption of independent and random faults is valid in a typical LAN, where a group of players communicate directly via

[2] Assuming reliable broadcast, the CS protocol works as follows: First every player broadcasts its public encryption key. Then every player picks its contribution to the shared key, encrypts it under each broadcasted public key, and broadcasts a message containing the resulting n ciphertexts. The shared key is computed by each player as the sum of all broadcasted contributions.

broadcast and are not directly bothered by failures of other players. Furthermore, JKT is geared for a homogeneous environment where each player runs on the same hardware/software platform. These two assumptions limit its scope. Specifically, JKT is a poor match for settings where players with heterogeneous hardware/software are spread across a wide-area network (WAN).

By allowing each player to piggyback its individual fault probability onto its first broadcast message, JKT can be used in a heterogeneous environment by trivially replacing the fault probability of every player by the highest fault probability. Such a protocol would be safe in terms of robustness, but it would cause larger messages, incurring higher costs than necessary for a specified level of robustness. Moreover, if there is a player with a very high fault probability, the protocol will always produce a maximum-sized messages for full robustness.

Also, a router failure in the WAN (e.g., due to a misconfiguration or congestion) increases the probability of network partitioning [12], which in turn increases the failure probability of the GKA protocol. Specifically, router failure results in the communication failure of all players which use that router as a gateway. Assuming the router's fault probabilities are given (e.g., from historical statistics), one naive solution might be to determine a player's overall fault probability by combining player failure and router failure probabilities and then computing the suitable message size according to the result. However, it is unclear how to combine these two different types of probabilities: individual player's faults are independent, while player faults stemming from router crashes are correlated. Moreover, performance is not only determined by fault probabilities, but also by the order of the players. For example, there is a higher chance of partition for the same-sized message if players are randomly ordered, as in [7]. This prompts the question of how to order players and how to treat two different failures in order to obtain good performance.

1.3 Scope

We consider applications where a peer group of players is distributed over any combination of LANs and WANs. The group is a long-term entity and a group-wide secret key is needed to bootstrap secure communication. We assume that router failure probability can be computed from historical statistics. Examples of the kind of groups we focus on are as follows:

- **Replicated File Systems.** File servers acting as data sharing and storage stations are critical in modern network environments. To mitigate potential failures of, attacks on, and overload of, a single file server, it is necessary to distribute replicated data.

- **Reservation systems.** Airline and hotel reservation systems must support distributed bookings even if some of the system components fail or are interrupted. Sophisticated reservation systems have been developed (using heuristics) based only on local data, that aim to maximize the number of tickets and rooms that can be sold while minimizing the risk of overbooking.

- **Collaborative workspace system.** A collaborative workspace is an distributed environment, wherein participants in disparate locations can access each other's data and interact just as they would inside a single entity. However, participants should be able to continue functioning using local data even if some peer participants are malfunctioning or are suspended.

1.4 Contributions

First, we investigate how to efficiently use prior work in a setting with heterogeneous players and construct a protocol that supports more flexible control parameters. We localize the message size parameter for each player and allow a player to compute its message size adaptively depending on reliability level of its neighbors.

Second, we address the challenge of combining two different types of fault probabilities (individual player and sub-group of players) by treating each type at a different layer. Basically, we plug two types of control parameters into our protocol: one (computed from a player fault probability) increases player connectivity within a clustered subgroup, and the other (computed from a router fault probability) increases connectivity among subgroups.

Third, since player ordering affects performance in a heterogeneous setting, we determine – through step-by-step simulations – which ordering is preferable in order to maximize efficiency. Simulation results show that random player ordering in the same subgroup outperforms non-random order, e.g., topological order or fault probability increasing order. In addition, keeping the topological order of a player among subgroups outperforms random order of players beyond its subgroup range.

Finally, we construct the first RGKA protocol that supports players with different failure probabilities, spread across any LAN/WAN combination, while also allowing for correlated failures among subgroups of players. The proposed protocol is efficient (2 rounds) and provably secure. We evaluate its robustness and performance both analytically and via simulations.

1.5 Organization

The rest of this paper is organized as follows: Section 2 presents our terminology, notation, communication and adversarial models, as well as necessary security definitions and cryptographic assumptions. Next, Section 3 describes

the new RGKA protocol. Section 4 evaluates its performance and section 5 concludes the paper.

2 Preliminaries

2.1 Terminology and Notation

We now summarize our notation and terminology.

- **Players.** Participating set of n players are denoted: P_1, ..., P_n where the ordering is determined by the protocol itself.
- **Sub-group.** A subset of players who can communicate directly, i.e., those on the same LAN.
- **(Border) Router.** A device that forwards data between sub-groups. It might be a player. Router failure causes the entire sub-group to fail (become disconnected) from the perspective of other players.
- **Failures/Faults.** Any player and any router can crash. A player crash results from hardware/software failure. A router crash results from network misconfiguration or traffic congestion and causes the communication failure of all players that use that router as a gateway to the rest of the world.
- **Multicast Communication.** We assume that all communication takes place over *reliable* and authenticated multicast channels [6, 13] where all non-faulty players have the same view of the broadcasted message (which can be null if the sender is faulty). We assume weak synchrony, i.e., players have synchronized clocks and execute the protocol in synchronized rounds. Messages from non-faulty players must arrive within some fixed time window, which we assume is large enough to accommodate clock skews and reasonable communication delays.
- **Adversary.** We assume an honest-but-curious outside adversary which can also impose arbitrary stop faults on the (otherwise honest) players. (We note, however, that using standard zero-knowledge proofs our protocols can easily be strengthened to tolerate malicious insiders at the price of a small constant increase in communication and computation.) Also, although the adversary can make each player stop at any time during protocol execution, such player failure can not violate the contract imposed by the reliable multicast assumption. The goal of the adversary is to learn the group key(s).
- **Gadget $X_{[j,i,k]}$:** The value multicast in the second round of the protocol which corresponds to the path of length two connecting nodes P_j, P_i, and P_k. (Refer to section 3.1).

The following notation is used from here on:

n	number of players
ν	player fault probability
μ	subgroup fault probability
f	failure probability of a single execution of the protocol
T	number of gadgets applicable to both right- and left-side neighbors per player
EXP(R)	expected number of rounds
EXP(MS)	expected message size per player

2.2 System Model

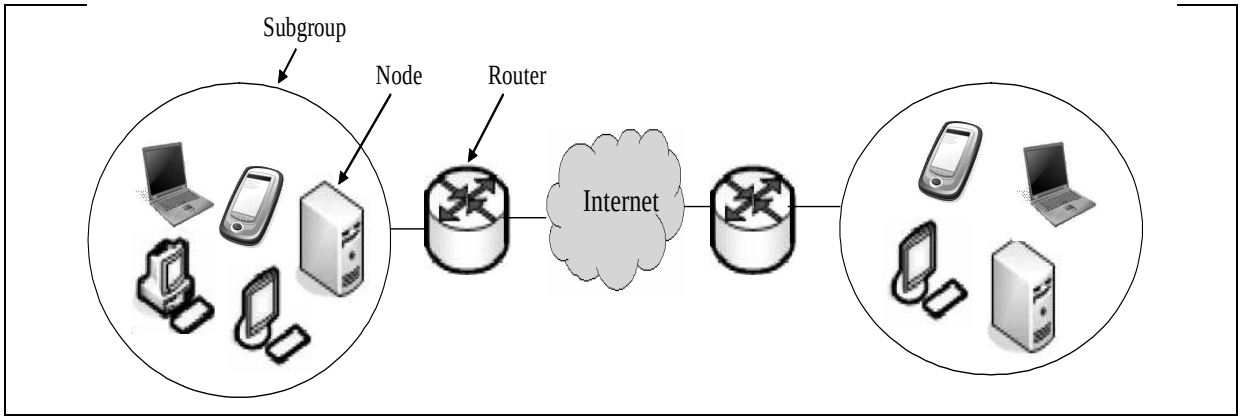


Figure 1: System Model

Figure 1 illustrates our system model. Our security model is a standard model for GKA protocols executed over authenticated links[3]. Since players in our setting do not use long-term secrets, we define GKA security (following [4, 7, 8]), as semantic security of the session key created in a single instance of the GKA protocol executed among honest parties. Specifically, the adversary can not distinguish between the key and a random value (with probability negligibly over $1/2$). The formal definition is as follows:

Definition 1. ***(GKA Security)*** *Consider an adversarial algorithm $\mathcal{A}$ which observes an execution of the GKA protocol between n honest players, and, depending on bit b, is given the session key computed by this protocol (if $b = 1$) or a random value chosen from the same domain as the session key (if $b = 0$). The adversary $\mathcal{A}$ outputs a single bit b'. We define adversary's advantage:*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{GKA}} = |\Pr[b' = b] - 1/2|$$

[3]Note that there are standard and inexpensive "compilation" techniques which convert any GKA protocol into an *authenticated* GKA protocol [8].

where the probability goes over the random execution of the protocol, the adversary $\mathcal{A}$, and the random choice of bit b.

We call a GKA protocol secure *if, for all adversaries $\mathcal{A}$, $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{GKA}}$ is negligible.*

2.3 Cryptographic Setting

We now describe our cryptographic assumptions. This section is included for the sake of completeness and can be skipped without any loss of continuity.

Let $\mathbb{G}$ be a cyclic group of prime order q, and let g be its generator. We assume that both Decision Diffie-Hellman (DDH) and Square-DDH problems are hard in $\mathbb{G}$. For example, $\mathbb{G}$ could be a subgroup of order q in the group of modular residues $\mathbb{Z}_p^*$ s.t. $p-1$ divides q, $|p| = 1024$ and $|q| = 160$, or it can be a group of points on an elliptic curve with order q for $|q| = 160$.

Definition 2. *The DDH problem is hard in $\mathbb{G}$, if, for every algorithm A, we have:* $|\Pr[x, y \leftarrow \mathbb{Z}_q : A(g, g^x, g^y, g^{xy}) = 1] - \Pr[x, y, z \leftarrow \mathbb{Z}_q : A(g, g^x, g^y, g^z) = 1]| \leq \epsilon$ *and ϵ is negligible.*

Definition 3. *The Square-DDH problem is hard in $\mathbb{G}$ if for every A we have:* $|\Pr[x \leftarrow \mathbb{Z}_q : A(g, g^x, g^{x^2}) = 1] - \Pr[x, z \leftarrow \mathbb{Z}_q : A(g, g^x, g^z) = 1]| \leq \epsilon$ *and ϵ is negligible.*

3 Robust Group Key Agreement Protocol in a WAN

In this section, we show the construction of a WAN-oriented RGKA protocol. As mentioned earlier, our work builds on [7]. We thus begin by describing the JKT protocol in more detail.

3.1 Overview of JKT protocol

JKT is basically a robust version of the 2-round GKA protocol by Burmester and Desmedt (BD) [4]. (We describe the BD protocol in appendix A.) BD succeeds only if the second-round message values called *gadgets* form a circular path through the graph of all "live" players. The idea behind adding robustness is simple: In the second round, players send out *additional* gadget values, such that, even if some players fail in the broadcast stage, gadgets broadcasted by the live players can be ordered to still form a circular path through those live players. In the following, we briefly describe a T-robust GKA protocol and then a fully robust GKA protocol which repeats the T-robust protocol until it succeeds.

T-robust GKA Protocol. As in [7], this protocol operates in two rounds assuming n players are ordered *cyclically* and *randomly*: P_1,...,P_n. In practice, the order can be determined by the hash of the first-round message: each player P_i broadcasts a public version $z_i = g^{t_i}$ of its (secret) contribution t_i to the group key. (We assume, for simplicity and without loss of generality, that all players survive the first round of the protocol.) In the second round, each P_i broadcasts gadgets: $X_{[k,i,i']} = (z_k/z_i)^{t_i}$ for $|k-i| \leq T$. (Note that gadgets $X_{[k,i,j]}$ for $|k-i| \leq T$ and $|j-i| \leq T$ can be also constructed since $X_{[k,i,j]} = X_{[k,i,i']}/X_{[j,i,i']}$.)

A gadget $X_{[k,i,j]}$ corresponds to the path of length two connecting players P_k, P_i, and P_j. Two gadgets are **connectable** if there exists a overlapping path. For example, for every i, gadgets $X_{[a_{i-1},a_i,a_{i+1}]}$ and $X_{[a_i,a_{i+1},a_{i+2}]}$ are connectable because the path of P_{a_i} and $P_{a_{i+1}}$ overlaps. Using graph terminology, the gadgets sent by the live players form a partially connected graph instead of a fully connected graph, as described in [7].

Each player ends up computing the same group key if the sequence of gadgets sent by all live players forms a circular path through the graph of all live players. If all live players form a *path*, a cycle can be also constructed by visiting every player twice as described in [7]. Let $P_{a_1}, ..., P_{a_m}$ denote the players who survive after the second broadcast round and form a circular path. Each P_{a_i} computes session key as:

$$\mathsf{sk}_{a_i} \;=\; (z_{a_{i-1}})^{m \cdot t_{a_i}} \cdot X_{a_i}^{m-1} \cdot X_{a_{i+1}}^{m-2} \cdot \cdots \cdot X_{a_{i-2}} \;=\; \mathsf{sk} = g^{t_{a_1}t_{a_2}+t_{a_2}t_{a_3}+\ldots+t_{a_m}t_{a_1}}$$

where $X_{a_i} = X_{[a_{i-1},a_i,a_{i'}]}/X_{[a_{i+1},a_i,a_{i'}]}$. The actual protocol – as viewed by a single player – is shown in Figure 2.

[**Round 1**]:

1.1 Each P_i picks a random $t_i \in Z_q$ and broadcasts $z_i = g^{t_i}$.

[**Round 2**]:

2.1 Let ActiveList be the list of indices of all players who complete Round 1.

2.2 Each P_i broadcast gadgets: $X_{[k,i,i']} = (z_i/z_k)^{t_i}$ for T nearest neighbors to the right and T nearest neighbors to the left, among players $k \in$ ActiveList. Define $X_{[i,i',k]}$ as $(X_{[k,i,i']})^{-1}$.

[**Key Computation**]:

3.1 Let ActiveList be the list of indices of all players who complete Round 2.

3.2 Every P_i sorts ActiveList in the same order and connects each pair of connectable gadgets. The session key can be computed only if P_i can construct a cycle either from a true Hamiltonian cycle or from a Hamiltonian path taken twice; wlog, we assume that the path is formed as $\{P_{a_1}, ..., P_{a_m}\}$, where for some i, j we have $a_i = a_j$ and $m \leq n$.

3.3 Each P_{a_i} computes $\mathsf{sk}_{a_i} = (z_{a_{i-1}})^{m \cdot t_{a_i}} \cdot X_{a_i}^{m-1} \cdot X_{a_{i+1}}^{m-2} \cdot \cdots \cdot X_{a_{i-2}}$ where $X_{a_i} = X_{[a_{i-1},a_i,a_{i'}]} \cdot X_{[a_i,a_{i'},a_{i+1}]}$.

(Note that $\mathsf{sk}_{a_i} = g^{t_{a_1}t_{a_2}+t_{a_2}t_{a_3}+\ldots+t_{a_m}t_{a_1}}$.)

Figure 2: The robust GKA Protocol with homogeneous players in a LAN

Fully Robust GKA with *Homogeneous* and *Random* Faults. A fully robust (but *not* constant-round) GKA protocol simply repeats the T-robust protocol above, with some parameter T, which we fix from the player fault probability and the expected number of rounds, until the T-robust protocol succeeds. Repeating the protocol increases the number of rounds and the protocol communication complexity, i.e., EXP(R)$= 1 + 1/(1 - f)$, EXP(MS)$= 1 + 2T/(1 - f)$, respectively [4]. Assuming that player faults are *random* and *independent*, the protocol failure probability f is upper-bounded by: $f \leq n^2/2 * \nu^{2T}$.

Therefore, given n, v, and f, we can compute a minimal gadget size – T – with which the protocol fails with probability at most f. As a result, the protocol will have at most $1+1/(1-f)$ rounds. This is described in Algorithm 1.

Algorithm 1: Optimal T Selection in random fault model

```
   Input: (n, ν, f)
   Output: T
   for (T' ← 1 to n/2) do
1.1    f' ← n^2/2 * ν^(2T)
1.2    if f' < f then
         break
2  MinMS ← 1 + 2T'/(1 − f')
   for (T' ← T' + 1 to n/2) do
3.1    f' ← n^2/2 * ν^(2T)
3.2    MS ← 1 + 2T'/(1 − f')'
3.3    if MinMS > MS then
         MinMS ← MS
   return T
```

The JKT protocol can upper-bound protocol failure probability f and compute optimal message size T using approximation techniques, assuming homogeneous players. However, it is not clear how to compute f and T in a heterogenous player model. Moreover, while individual player faults are independent, subgroup faults are correlated.

Even if we could compute optimal message size, it would work only for a particular order of participating players. In other words, for each message size, the order of players changes the performance of the protocol. Recall that the JKT protocol computes gadgets for T nearest neighbors hoping that at least one of them survives all protocol steps. If a given player is surrounded by other players with high fault probabilities and gadgets connects only those players, the said player will very likely end up disconnected.

In the following two sections, we explore how to order players and how to compute the message size heuristically.

[4] Note that the protocol restarts only the second round since the messages from the first round are safely reusable.

3.2 Random or Non-random Order?

Heterogeneous Players on a *LAN*. Basically, there are two extreme cases: ordering players by their fault probabilities and ordering them randomly.[5] For the same number of gadgets T, to see which way of ordering provides better performance, we simulate the protocol for each case, compute the expected number of rounds and the expected message size, and then compare them.

We use a simple scenario with two subgroups of 25 players with a low and a high fault probability, respectively. In this scenario step, we assume that every player is on the same LAN and thus do not consider router failures. The summary of this scenario is in Table 1.

Group ID	A	B
n	25	25
ν	0.01	0.3
μ	0	0

Table 1: Two subgroups of players with different but *independent* fault probabilities.

Ordering	Topological Order			Random Order		
T	2	3	4	2	3	4
EXP(R)	2.913	2.052	2.005	2.266	2.010	2.000
EXP(MS)	8.652	7.312	9.040	6.064	7.060	9.000

Table 2: Expected number of rounds and expected message size with three different T values on two different player orders.

We simulate the above scenario with three different values of T (2, 3, and 4) in topological and random orders, respectively. The results are summarized in Table 2. For every T, random order outperforms topological order. We believe that this is because random order uniformly distributes players with low fault rate and increases the probability for every player to have non-faulty players among its T nearest neighbors. Thus, random order can be a practical solution for players with different, but independent, fault probabilities.

Heterogeneous Players in a *WAN*. To see how correlated failures affect performance, we simulate another scenario with two subgroups of 25 players. Each subgroup connects to the WAN via a router. We assume that one router has low, and the other – high, failure probability. Since we now focus on correlated failures, we also assume that an individual player never fails, but only its communication can fail due to the failure of the router connecting its subgroup to the WAN. This scenario is summarized in Table 3.

[5] Of course, other ordering criteria are possible, e.g., by bit error rates of player interfaces. However, there is considerably less intuitive justification for considering these criteria.

Group ID	A	B
n	25	25
ν	0	0
μ	0.01	0.3

Table 3: Two subgroups of players are given with different but *correlated* fault probabilities.

Ordering	Topological Order			Random Order		
T	2	3	4	2	3	4
EXP(R)	2.000	2.000	2.000	2.427	2.427	2.007
EXP(MS)	5.000	7.000	9.000	6.708	9.562	9.056

Table 4: Expected number of rounds and expected message size with three different T values, with two player orders.

Again, we simulate the scenario with three different T values (2, 3 and 4) and summarize results in Table 4. Interestingly, unlike the first scenario, performance is much better when players are ordered topologically. The results show that, when players are topologically ordered, the protocol never fails because of a router failure, regardless of T. This is because there is no partition when only one subgroup fails. (For an inter-group partition to occur, there should be at least four subgroups.) The situation is very different with random order: if some scattered players fail at the same time, a partition is very likely.

Heterogeneous Players in a LAN/WAN. As shown above, random player order improves performance with independent player faults, while topological order achieves better performance with correlated subgroup faults. The natural next step is to consider the case of *both* independent and correlated faults, e.g., in a mixed LAN/WAN setting. To this end, we simulate a scenario with 4 subgroups, each composed of 15 players. Each player has an independent fault probability (among players) and each subgroup also has an independent fault probability (among subgroups). A player thus fails either alone or as part of its subgroup failure. In the latter case, the disconnected subgroup (containing a given player) might still complete the protocol; however, from the perspective of outside players, all players in the failed subgroup are gone. This scenario is summarized in Table 5.

Group ID	A	B	C	D
n	15	15	15	15
ν	0.2	0.01	0.1	0.3
μ	0.01	0.1	0.3	0.1

Table 5: 4 subgroups of 15 players each with different fault probabilities.

Simulation results in Table 6 show that topological order has fewer expected rounds and lower expected message size. We conclude that, to obtain better performance, players should be ordered topologically by subgroups and randomly within a subgroup.

Ordering	Topological Order			Random Order		
T	2	3	4	2	3	4
EXP(R)	3.371	2.165	2.045	3.701	2.404	2.165
EXP(MS)	10.484	7.99	9.360	11.804	9.424	10.32

Table 6: Expected number of rounds and message size with 3 T values, with two different player orders.

3.3 Player-Specific Message Size

In this section, we describe how to compute optimal message sizes for different players. We assume that players are grouped into subgroups topologically and randomly within subgroups. We determine message size in terms of both player-to-player and subgroup-to-subgroup communication, respectively.

Player-to-Player. Our approach is to localize T depending on the reliability of neighboring players. Specifically, a player has a larger T with less reliable neighbors, and a lower T with more reliable neighbors. If player fault probabilities are evenly distributed, we can obtain the best performance by simply letting each player compute the same number of gadgets T using algorithm 1. Whereas, more realistically, if player fault probabilities are unevenly spaced, to make every point equally reliable, each player has to adaptively compute the number of gadgets depending on the robustness of its neighbors. (Recall that a player is disconnected if all T nearest neighbors fail.)

We introduce three new variables:

- OT_i: optimal number of gadgets, applicable to both right- and left-side neighbors of P_i, *assuming* that each player has the same fault probability as P_i.
- RT_i and LT_i: localized numbers of gadgets applicable to P_i's right and left side neighbors, respectively.

We estimate P_i's robustness from its OT_i value. For example, a player with $OT = 1$ is most robust, while a player with $OT = n/2$ is least robust (where n is the number of players in a subgroup. In our algorithm, whenever a player computes a gadget which makes a connection to one of its neighbors, the robustness level of the player increases in inverse proportion to its neighbor's OT. P_i computes RT_i and LT_i according to its right and left neighbor's OT values (OT_{i-1}, OT_{i+1}), respectively, until its robustness level reaches a specified level, which is 1 in our algorithm. A more precise description is shown in Algorithm 2.

Note that a gadget that a player computes for one neighbor can not be used if the neighbor does not compute a reciprocal gadget. (Recall that gadgets are connectable if there exists a overlapping path.) In fact, in Algorithm 2, for a given pair of players, either both compute a gadget for each other or neither does.

Algorithm 2: T Localization on player-to-player basis

Input: ($n, \nu_1, \cdots, \nu_n, f$**)**
Requirement 1: $P_1, \cdots, P_n$ are randomly ordered within subgroups and topologically across subgroups.
Requirement 2: n is the number of players and indices cycle modulo n, i.e. $P_{n+1} = P_1$
Requirement 3: P_i's fault probability is ν_i
Ensure: for each P_i, compute RT_i and LT_i

for *(*$i \leftarrow 1$ **to** n*)* **do**
  Compute optimal OT_i using Algorithm 1 on input (n, ν_i, f)

for *(*$i \leftarrow 1$ **to** n*)* **do**
  $reliability \leftarrow 0$
  $RT_i \leftarrow 0$
  for *(*$j \leftarrow i+1$*;* $reliability \geq 1$ *or* $j - i \geq n/2$*;* $j \leftarrow j+1$*)* **do**
    $RT_i \leftarrow RT_i + 1$ (Add P_j to the list of P_i's right neighbors)
    $reliability \leftarrow reliability + 1/OT_j$
  $reliability \leftarrow 0$
  $LT_i \leftarrow 0$
  for *(*$j \leftarrow i-1$*;* $reliability \geq 1$ *or* $i - j \geq n/2$*;* $j \leftarrow j-1$*)* **do**
    $LT_i \leftarrow LT_i + 1$ (Add P_j to the list of P_i's left neighbors)
    $reliability \leftarrow reliability + 1/OT_j$

Proposition 1. *In Algorithm 2, if a player computes a gadget for a neighbor, then the neighbor computes a reciprocal gadget.*

Proof. Since RT and LT are symmetric, we focus only on RT. Assume that P_i computes $RT_i = k$. Since P_{i+k} is added as one of its right-side neighbors, the summation of $reliability$ from P_{i+1} to P_{i+k-1} is less than one, i.e., , $\sum_{j=i+1}^{i+k-1} \frac{1}{OT_j} < 1$. Therefore, P_{i+k} computes $LT_{i+k} \geq k$ including P_i as one of its left neighbors, since $\sum_{j=i+1}^{i+k-1} \frac{1}{OT_j} < 1$. □

Subgroup-to-Subgroup. In the proposed algorithm, we logically treat each subgroup as a kind of a *super-player*. Specifically, in each subgroup, a player with the lowest fault probability becomes a representative and sends out extra gadgets for other representatives. A representative player becomes faulty if either the player itself or its subgroup fails. Thus, given player failure rate ν and subgroup failure rate μ, the failure probability [6] of a representative player is: $\nu + \mu - \nu\mu$. The algorithm is described in Algorithm 3.

Proposition 2. *In Algorithm 3, if a representative player computes a gadget value for a representative neighbor then the representative neighbor also computes a gadget value for the representative player.*

Proof. Identical to the proof of Proposition 1. □

[6] At the player-to-player level, a player also fails from either its own or its subgroup fault. However, since players are topologically ordered and gadgets connect only nearest neighbors, the subgroup fault is not considered in the player-to-player level robustness.

Algorithm 3: T Localization in subgroup-to-subgroup level

Input: $(m, \nu_1, \cdots, \nu_m, \mu_1, \cdots, \mu_m, f)$
Requirement 1: $P_1, \cdots, P_m$ are a set of representative players
Requirement 2: m is the number of subgroups and indices cycle mod m, i.e. $P_{m+1} = P_1$
Requirement 3: P_i's fault probability and subgroup fault probability are ν_i and μ_i, respectively
Ensure: for each P_i, compute RT_i and LT_i

for $(i \leftarrow 1$ **to** $m)$ **do**
  Compute optimal OT_i using Algorithm 1 on input $(m, (\mu_i + \nu_i - \mu_i \cdot \nu_i), f)$

for $(i \leftarrow 1$ **to** $m)$ **do**
  $reliability \leftarrow 0$
  $RT_i \leftarrow 0$
  for $(j \leftarrow i+1;\ reliability \geq 1$ *or* $j - i \geq m/2;\ j \leftarrow j+1)$ **do**
    $RT_i \leftarrow RT_i + 1$ (Add P_j to the list of P_i's right neighbors)
    $reliability \leftarrow reliability + 1/OT_j$
  $reliability \leftarrow 0$
  $LT_i \leftarrow 0$
  for $(j \leftarrow i-1;\ reliability \geq 1$ *or* $i - j \geq m/2;\ j \leftarrow j-1)$ **do**
    $LT_i \leftarrow LT_i + 1$ (Add P_j to the list of P_i's left neighbors)
    $reliability \leftarrow reliability + 1/OT_j$

3.4 RGKA in a LAN/WAN Setting

Based on Algorithms 2 and 3, we propose a *W-RGKA* protocol for heterogeneous players. *W-RGKA* allows each player to adaptively compute its message size depending on the reliability level of its neighbors. *W-RGKA* automatically defaults to the JKT protocol [7] in a setting with homogeneous players. Note that the homogeneous player setting is a special case of Algorithm 2, where every player computes the same $OT = RT = LT$. In other words, we succeed in extending the JKT protocol without losing its optimality in a homogeneous setting.

We also relax the way the key is computed such that *any* circular path that connects all live players can be used for key computation. The resulting graph that gadgets draw in two levels is more complex than the one (called T-th power of a circle) shown in the JKT protocol which builds either a Hamiltonian cycle or a Hamiltonian path on all live players. To enable stronger robustness, we relax the way of finding a circular path, so that the key is associated not necessarily with a Hamiltonian cycle or a Hamiltonian path, but any circular path where a player can be visited more than once. If there is no partition, there is a always a circular path. The resulting *W-RGKA* protocol is shown in Figure 3.

Theorem 1. *Assuming that the DDH problem and Square-DDH problem are hard, protocol* W-RGKA *is a secure Group Key Agreement.*

The security of the *W-RGKA* protocol which broadcasts RT and LT sized messages is implied by the security argument for the RGKA protocol in [7] which broadcasts maximum sized messages, thus revealing maximum amount of information. The only difference is that the resulting key in *W-RGKA* might contain each contribution

[**Round 1**]:

1.1 Same as in Figure 2 except that each P_i also broadcasts its fault probability ν_i.

[**Round 2**]: same as in Figure 2 except:

2.2 Each player P_i computes RT_i and LT_i using Algorithm 2 and broadcast gadgets $X_{[k,i,i']} = (z_i/z_k)^{t_i}$ for RT_i nearest neighbors to the right and LT_i nearest neighbors to the left among players $k \in \mathsf{ActiveList}$.

***2.3** Each representative P_i computes RT_i and LT_i using Algorithm 3 and broadcast gadgets $X_{[k,i,i']} = (z_i/z_k)^{t_i}$ for RT_i nearest neighbors to the right and LT_i nearest neighbors to the left among representative players $k \in \mathsf{ActiveList}$.

[**Key Computation**]: Same as in Figure 2 except:

3.2 The session key can be computed if there exists a circular path where every player is visited at least more than once. Wlog, we assume that the path is formed as $\{P_{a_1}, \cdots, P_{a_m}\}$, where for some i, j we can have $a_i = a_j$.

* Executed only by representative players

Figure 3: *W-RGKA* protocol with heterogeneous players in a LAN/WAN setting

of the form t_{a_i}, $t_{a_{i+1}}$ more than once. However, the resulting key equation is still linearly independent from the equations generated from gadgets. Thus, the key value is independent from gadgets values and the adversary cannot learn anything about the key from the messages it observes. For details, refer to Section 6.2 in [7].

4 Performance Evaluation

We first summarize the relevant aspects of protocol efficiency.

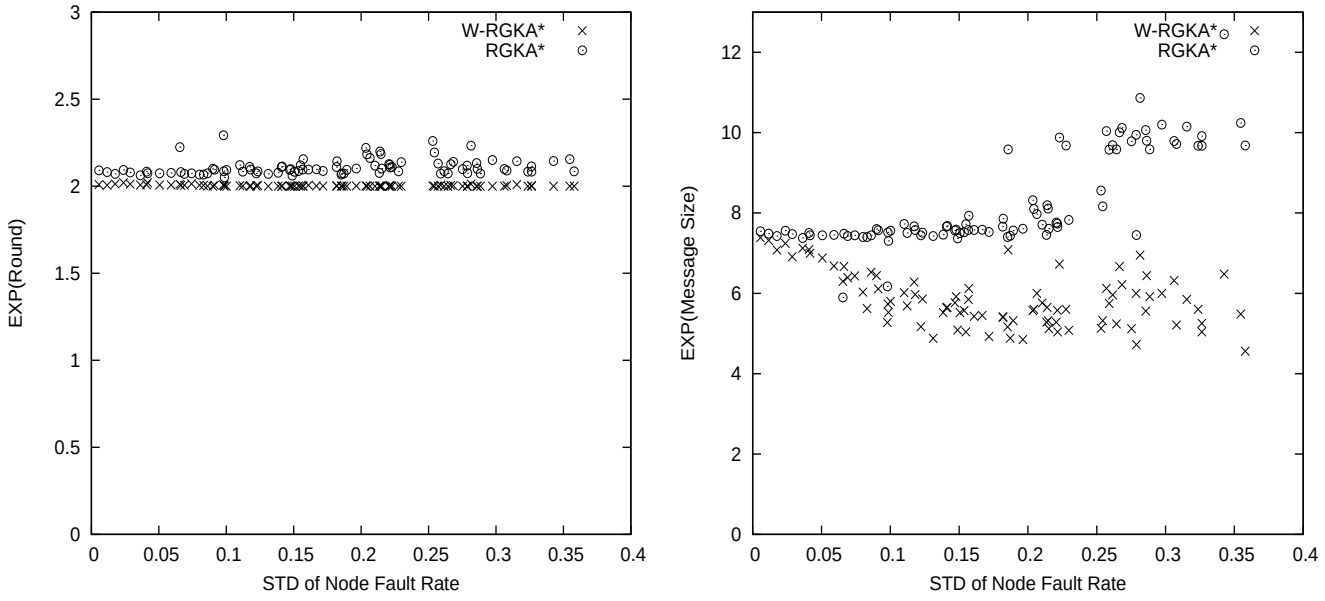
- **Round Complexity:** number of protocol rounds.
- **Communication Complexity:** (expected) total bit-length of all messages sent in the protocol.
- **Computational Complexity:** computation that must be performed by each player.

REMARK: In the specific GKA protocols we compare, computational complexity increases in proportion to communication complexity. Generally, one message unit incurs one exponentiation (which dominates computational cost). Thus we do not separate computational complexity from communication complexity in the following comparison.

We compare *W-RGKA* with the fully robust JKT protocol [7], as described in section 3.1. To make *W-RGKA* fully robust we repeat it until it succeeds; this is the same approach used to obtain a fully robust version of JKT in [7]. We denote our fully robust version as *W-RGKA** and the fully robust JKT version by *RGKA**. However, since *RGKA** works in a homogeneous setting, we simulate *RGKA** by taking the average of all player fault probabilities.

We analyze how player heterogeneity and correlated faults affect performance. We evaluate the protocols in a setting of 5 subgroups with 10 players for each. In the first simulation, we generate subgroup fault probabilities such that the average subgroup fault probability is around 0.1 (with a standard deviation less than 0.1) and generate

10 players for each group with random fault probability, such that the standard deviation varies between 0 and 0.4. Note that the standard deviation of player fault probability distribution indirectly shows the heterogeneity of the set of players. In the second simulation, we generate players randomly but with a small deviation (less than 0.1) and change subgroup fault probabilities such that the average ranges from 0 to 0.4.

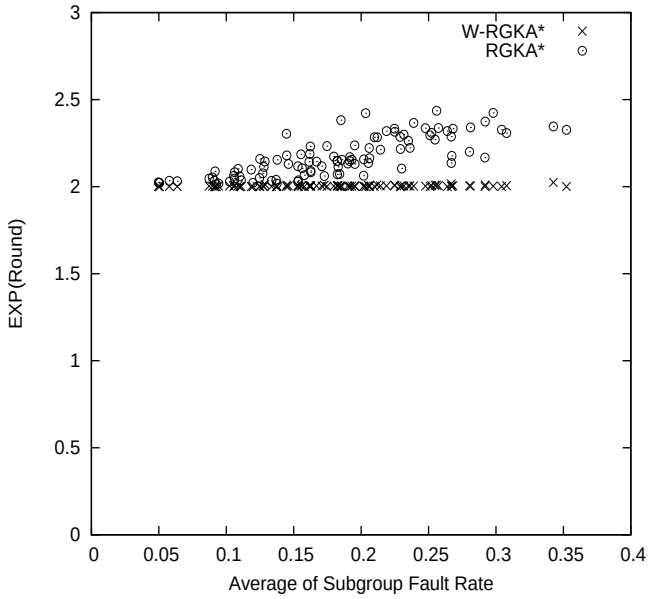


(a) expected number of rounds (b) expected message size

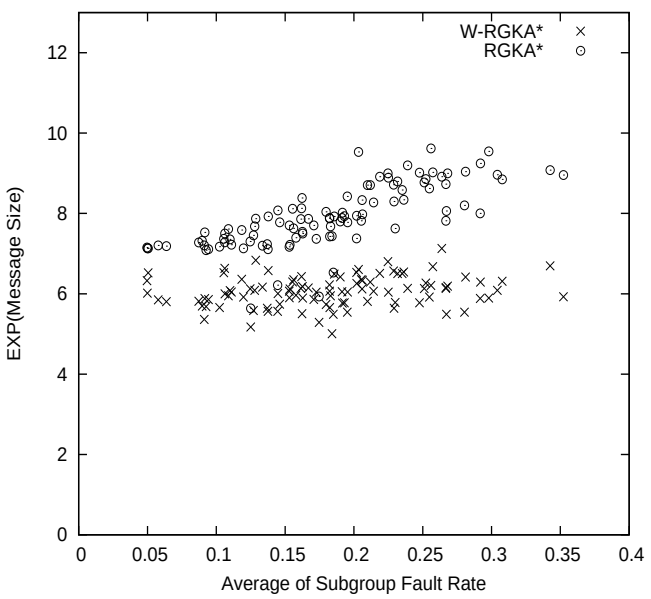
Figure 4: *W-RGKA** vs. *RGKA** for different standard deviations of player fault rate distribution. Results are based on simulating over 100 runs.

Figure 4 shows the expected number of rounds (a) and expected message size (b) for each protocol with different standard deviations. Overall, *W-RGKA** outperforms *RGKA** in both round and communication complexity. The number of rounds in *W-RGKA** tops out at 2.01, compared with 2.2 in *RGKA**. This might seem insignificant, however, considering that the underlying non-robust BD protocol always takes 2 rounds, the difference becomes more substantial. We also observe that the communication cost of *W-RGKA** is far lower than that of *RGKA**, particularly, with higher standard deviation in player fault rates.

Figure 5 shows the expected number of rounds (a) and expected message size (b) for both protocols, taking into account subgroup fault rates. Once again, *W-RGKA** exhibits better performance on both counts. The number of rounds in *W-RGKA** still lies below 2.01. Whereas, for *RGKA**, the number of rounds increases proportionally to averaged correlated fault rates, and thus quickly shoots up to 2.5. Also, communication complexity of *RGKA** increases as the average of correlated fault rates grows. This is mainly because *RGKA** does not consider correlated faults in its design.



(a) expected number of rounds



(b) expected message size

Figure 5: *W-RGKA** vs. *RGKA** for different average subgroup fault rates. Results are based on simulating over 100 runs.

5 Conclusions

This paper started off with the state-of-the-art in robust GKA protocols. Having identified certain limitations of prior work, i.e., assumptions about independent failures and homogeneous players, we demonstrated a step-by-step construction of a new protocol *W-RGKA* suitable for a mixed LAN/WAN setting. While the proposed protocol inherits the attractive features of its predecessor (JKT), it also heuristically determines per-player optimal message sizes and handles heterogeneous fault probabilities as well as correlated failures. Simulations help determine the preferred player order for different scenarios.

One obvious item for future work is to conduct a more extensive set of experiments and simulations. Another issue is that the current protocol does not take into account inter-subgroup delay. It is natural to consider this variable (assuming it is known ahead of time) in determining the optimal subgroup order.

References

[1] Y. Amir, C. Nita-Rotaru, J. Schultz, J. Stanton, Y. Kim, and G. Tsudik. Exploring robustness in group key agreement. In *ICDCS*, pages 399–408, 2001.

[2] E. Bresson, O. Chevassut, and D. Pointcheval. Provably authenticated group Diffie-Hellman key exchange — the dynamic case. In *Asiacrypt 2001*, Dec 2001.

[3] E. Bresson, O. Chevassut, D. Pointcheval, and J. Quisquater. Provably authenticated group Diffie-Hellman key exchange. In *ACM CCS*, Nov 2001.

[4] M. Burmester and Y. Desmedt. A secure and efficient conference key distribution system. In *EUROCRYPT*, pages 275–286, 1994.

[5] C. Cachin and R. Strobl. Asynchronous group key exchange with failures. In *PODC*, pages 357–366, 2004.

[6] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM ToN*, 5(6):784–803, 1997.

[7] S. Jarecki, J. Kim, and G. Tsudik. Robust group key agreement using short broadcasts. In *ACM CCS*, pages 411–420, 2007.

[8] J. Katz and M. Yung. Scalable protocols for authenticated group key exchange. *Journal of Cryptology*, 20(1):85–113, 2007.

[9] Y. Kim, A. Perrig, and G. Tsudik. Simple and fault-tolerant key agreement for dynamic collaborative groups. In *ACM Conference on Computer and Communications Security*, pages 235–244, 2000.

[10] Y. Kim, A. Perrig, and G. Tsudik. Group key agreement efficient in communication. *IEEE ToC*, 33(7), 2004.

[11] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[12] L. Moser, Y. Amir, P. Melliar-Smith, and D. Agarwal. Extended virtual synchrony. In *ICDCS*, pages 56–65, 1994.

[13] S. Paul, K. Sabnani, J. Lin, and S. Bhattacharya. Reliable multicast transport protocol (rmtp). *IEEE JSAC*, 15(3):407–421, 1997.

[14] D. Steer, L. Strawczynski, W. Diffie, and M. Wiener. A secure audio teleconference system. *CRYPTO'88*, Aug 1990.

[15] M. Steiner, G. Tsudik, and M. Waidner. Key agreement in dynamic peer groups. *IEEE TPDS*, 11(8):769–780, 2000.

A Burmester-Desmedt GKA

The BD GKA protocol proceeds in two rounds (see Figure 6): First each player P_i broadcasts a public counterpart $z_i = g^{t_i}$ of its contribution t_i to the key. In the second round each P_i broadcasts gadget $X_{[i-1,i,i+1]} = g^{t_i t_{i+1} - t_{i-1} t_i}$ (which it can compute as $X_{[i-1,i,i+1]} = (z_{i+1}/z_i)^{t_i}$). Given the set of gadget values $X_{[n,1,2]}, X_{[1,2,3]}, ..., X_{[n-1,n,1]}$, each player P_i can use its contribution t_i to locally compute the common session key $\mathsf{sk} = g^{t_1 t_2 + t_2 t_3 + ... + t_n t_1}$.

[**Round 1**]:

Each player P_i picks a random $t_i \in Z_q$ and broadcasts $z_i = g^{t_i}$.

[**Round 2**]:

Each P_i broadcasts its gadget value $X_{[i-1,i,i+1]} = (z_{i+1}/z_{i-1})^{t_i}$, where the indices are taken in a cycle.

[**Key Computation**]:

Each P_i computes the key as $\mathsf{sk}_i = (z_{i-1})^{n t_i} \cdot X_i^{n-1} \cdot X_{i+1}^{n-2} \cdots X_{i-2}$, where $X_i = X_{[i-1,i,i+1]}$.

(Note that for all i we have $\mathsf{sk}_i = g^{t_1 t_2 + t_2 t_3 + ... + t_n t_1}$.)

Figure 6: Burmester-Desmedt's Group Key Agreement Protocol (BD GKA)

As we explain in section 3.1, a sequence of gadgets *forms a path through the graph* if each two consecutive gadgets in the sequence are connectable. By inspecting the formula for deriving the secret key in the BD GKA protocol we can observe that each player derives the same key because the set of gadgets broadcasted in the second round of the protocol forms a Hamiltonian cycle (i.e. a circular path) through the graph of all players.