Asynchronous Group Key Exchange with Failures

Christian Cachin Reto Strobl
IBM Research
Zurich Research Laboratory
CH-8803 Rüschlikon, Switzerland
cca,rts@zurich.ibm.com

ABSTRACT

Group key exchange protocols allow a group of servers communicating over an asynchronous network of point-to-point links to establish a common key, such that an adversary which fully controls the network links (but not the group members) cannot learn the key. Currently known group key exchange protocols rely on the assumption that all group members participate in the protocol and if a single server crashes, then no server may terminate the protocol. In this paper, we propose the first purely asynchronous group key exchange protocol that tolerates a minority of servers to crash. Our solution uses a constant number of rounds, which makes it suitable for use in practice. Furthermore, we also investigate how to provide forward secrecy with respect to an adversary that may break into some servers and observe their internal state. We show that any group key exchange protocol among n servers that tolerates $t_c > 0$ servers to crash can only provide forward secrecy if the adversary breaks into less than $n - 2t_c$ servers, and propose a group key exchange protocol that achieves this bound.

Keywords

Group Key Exchange, Group Communication, Provable Security, Universal Composability

General Terms

Security, Reliability

Categories and Subject Descriptors

D.4.6 [**Security and Protection**]: Cryptographic controls;
C.2.4 [**Distributed Systems**]: Distributed applications

1. INTRODUCTION

Group Key Exchange (GKE) protocols allow a group of servers communicating over a complete network of point-to-point links to establish a common *session key* such that anyone outside the group that can only observe the network traffic cannot learn this key. Such a session key can later be used to achieve cryptographic goals like for example multicast message confidentiality, or multicast data integrity. Hence, GKE protocols are essential for applications such as secure video- or tele-conferencing, or other collaborative applications. To model environments like the Internet, one assumes an asynchronous network where the scheduling of messages is determined by an adversary, and where the servers do not have access to a common clock.

The main goals of a GKE protocol are to ensure secrecy of the session key, and to ensure that every member of the group eventually terminates the protocol and computes the session key. So far, GKE protocols have been designed to meet these goals only as long as all members of the group follow the protocol specification [8, 26, 3, 7]. These solutions have the drawback that if only a single server crashes, then no member of the group will terminate the protocol. This makes such protocols specifically vulnerable to denial of service attacks, as the execution time of the protocol is determined by the slowest member of the group.

One way to solve this problem, explored by Amir et al. [2], is to base GKE on a view-based group communication system (GCS), which provides the abstraction of the "currently live nodes" to all servers in a consistent way (Chockler et al. [15] provide a survey of GCS). Since the GCS can detect crashes among the servers also during the execution of a GKE protocol, the protocol can react accordingly; as GCSs rely on timeouts to detect crashed participants, the approach leads to solutions that are not purely asynchronous and subject to timing attacks, however.

In this paper we propose the first GKE protocol in a *purely asynchronous* model that terminates for every member as long as at least a majority of the participants remain up, which is optimal for this model. Our solution is conceptually simple and efficient, which makes it suitable for practice. In particular, it consists of the following two stages. In the first stage, the group members exchange keying information using two communication rounds and a total of $O(n^2)$ messages, where n denotes the size of the group. In the second stage, they execute a consensus protocol to select the contributions from the first stage from which the session key is computed. The protocol may use randomized asynchronous consensus in the fully asynchronous model or a consensus protocol in the asynchronous model augmented with a failure detector [14]. In the latter case, our approach yields a modular solution for GKE in the same model as the GCS-based protocol mentioned above.

Comparing the efficiency of our construction with the most efficient solution for GKE without failures [8] shows that the

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
PODC'04, July 25–28, 2004, St. Johns, Newfoundland, Canada.
Copyright 2004 ACM 1-58113-802-4/04/0007 ...$5.00.

price we pay for tolerating failures lies only in the consensus protocol executed in the second stage. We also show that the communication complexity of our construction is nearly optimal. In particular, we show that given an optimal solution for consensus, our construction yields a solution for GKE that uses only $O(n^2)$ messages and $O(1)$ communication rounds more on average than an optimal solution for GKE with failures.

Following the approach of Steiner [27], we analyze the security of our protocol in the framework for asynchronous reactive systems proposed by Pfitzmann and Waidner [23]. In particular, we first specify the target behavior of a GKE protocol in terms of an idealized service, and then show that our protocol has the same input-output behavior as this idealized service. This approach has the benefit of guaranteeing composability, i.e., the security of any application relying on an ideal service for GKE remains the same when our real protocol is used to implement the ideal service.

We first prove the security of our construction in the so-called *weak corruption* model with failures, where the adversary may schedule and observe the network, crash servers, but not break into a server and observe its internal state. We show that in this model, our protocol tolerates $t_c < n/2$ servers that crash, which is optimal for this setting.

We then investigate how to provide *forward secrecy* in the *strong corruption* model, where the adversary is additionally allowed to break into some servers and to observe their internal state. Such break-ins should only compromise the security of the session keys that are being generated during the attack, but not the keys generated previously or afterwards. We first show that if a GKE protocol tolerates $t_c > 0$ servers that crash, then it can tolerate strictly less than $n - 2t_c$ break-ins. We then show how to build a GKE protocol with this optimal resilience. Our construction is also practical: It uses one execution of a consensus protocol, and additionally $O(n^2)$ messages in three rounds.

Related Work. One can classify previous work on GKE along two dimensions: The assumptions made on the communication links, and the framework used for proving security. Along the first dimension, one can distinguish between GKE protocols that assume an authentic network (as we do) [19, 8, 26, 24, 27], and GKE protocols that rely on a-priori distributed public and private keys (as for example provided by a public-key infrastructure) [20, 6, 3, 28]. A recent paper of Katz and Yung [21] closes the bridge between these two approaches by showing how any GKE protocol built for an authentic network can be "compiled" into a GKE protocol for an insecure network with a-priori distributed public and private keys. The compilation only adds one additional communication round and $O(n^2)$ messages.

With respect to the framework used for proving security, one can identify two approaches. One approach, taken by Bresson et al. [7] and by Katz and Yung [21], is to extend the framework for modeling two-party key exchange proposed by Bellare et al. [5] to the n-party case. Another approach, taken by Steiner [27] and by this work, is to use a general framework for modeling asynchronous reactive systems — such as the one of Pfitzmann and Waidner [23] or the one of Canetti [12] — and define and prove security therein. The advantage of using such a framework is that security is preserved under modular composition.

Some of the above-mentioned works [3, 26, 24, 7] also address the *dynamic* case of GKE, where servers may join or leave the group and the session key must be updated whenever this occurs. Recently, Amir et al. [1] showed how the dynamic join and leave protocols of Steiner et al. [26] can be integrated with a view-based GCS to maintain a common group key, which is later used to encrypt the communication among the group. The group key has to be updated whenever the underlying GCS detects a change in the group structure. This is accomplished as outlined in [2], i.e., by running the corresponding dynamic GKE protocol whenever servers join or leave. In case that the GCS detects *nested leaves*, i.e., detects leaving servers *during* the execution of a (dynamic) GKE protocol, this protocol is aborted and a basic GKE protocol is run from scratch among the remaining servers. Establishing a key in case of nested leaves can also be seen as crash-tolerant GKE. Note that if t_c servers crash one by one, this approach leads to t_c sequential executions of a basic GKE protocol. If the basic GKE protocol of Steiner et al. [26] is used, this results in $O(t_c n)$ messages and $O(t_c n)$ rounds. Furthermore, as GCSs rely on timeouts to detect crashes, this approach is not purely asynchronous and subject to timing attacks.

Organization. In the next section, we introduce our system model and give the formal definitions for GKE and consensus. In Section 3, we present our construction for GKE with failures in the weak corruption model, and elaborate on its optimality. In Section 4, we first introduce the strong corruption model and adjust the definition of GKE to this model. We then prove an upper bound on the number of break-ins one can tolerate for GKE with failures in the strong corruption model, and finally show how to build a GKE protocol with this optimal resilience.

2. PRELIMINARIES

2.1 The Framework

Our computational model is parameterized by a security parameter k; a function $\epsilon(k)$ is called *negligible* if for all $c > 0$ there exists a k_0 such that $\epsilon(k) < \frac{1}{k^c}$ for all $k > k_0$. Two ensembles $\{var_k\}_k \in \mathbb{N}$ and $\{var'_k\}_{k\in\mathbb{N}}$ of random variables (or probability distributions) are called *computationally indistinguishable* if for every algorithm $\mathcal{D}$ (the distinguisher) that runs in probabilistic polynomial-time in its first input, the following quantity is negligible: $|\Pr[\mathcal{D}(1^k, var_k) = 1] - |\Pr[\mathcal{D}(1^k, var'_k) = 1]|$. Throughout the paper, we abbreviate this by saying that "var_k and $var_{k'}$ are indistinguishable", and write "$var_k \approx var_{k'}$".

We will study our protocols in the framework for universally composable asynchronous reactive systems of Pfitzmann and Waidner [23]. We sketch a simplified version of the model.

Overview. We model a protocol π as a collection of n probabilistic polynomial-time (in k) interactive Turing machines (PPT ITM) $\mathsf{M}_1^\pi, \ldots, \mathsf{M}_n^\pi$ called the *servers*, which communicate over an authentic network NET modeled as a PPT ITM. We call the collection $\{\mathsf{NET}, \mathsf{M}_1^\pi, \ldots, \mathsf{M}_n^\pi\}$ a *real system* for protocol π in network NET and denote it by $Sys_n^{\mathsf{real},\pi}$. We model protocol-specific input and output of a server M_i^π in terms of messages that occur at M_i^π's input and output connections in_i and out_i, respectively. We call the set I of the input-output connections of all servers the *interface*[1] of $Sys_n^{\mathsf{real},\pi}$.

[1]For readers familiar with [23]: The interface corresponds to the *specified ports*.

We model an execution of a protocol π in a network NET as a run of the real system $Sys_n^{\mathsf{real},\pi}$ augmented with two PPT ITMs: a user H and an adversary $\mathsf{A}^{\mathsf{real}}$. We call the collection $\{Sys_n^{\mathsf{real},\pi}, \mathsf{H}, \mathsf{A}^{\mathsf{real}}\}$ a *real configuration* for π. The *user* H represents a higher-level application that builds on top of the servers. It may interact with the servers through the interface (or a subset thereof), and may also communicate with $\mathsf{A}^{\mathsf{real}}$ at arbitrary points during the protocol. $\mathsf{A}^{\mathsf{real}}$ may attack the servers and schedule the network. We model attacks on a server M_i^π in terms of special messages that may occur at a designated input connection $\mathsf{cor\_in}_i$ of M_i^π (also part of the interface). For now, we consider the *weak corruption model*, where the only available attack of the adversary is to *crash* a server; if this happens, the server halts, i.e., does not participate in the protocol anymore. In Section 4, we will discuss the *strong corruption model*, where also break-ins are allowed.

We describe the security properties of π in terms of a service f that the corresponding real system $Sys_n^{\mathsf{real},\pi}$ should guarantee at its interface. Formally, we define the service f in terms of an ideal system $Sys_n^{\mathsf{ideal},\mathsf{f}}$. This ideal system has the same interface I as the real system, but comprises only a single PPT ITM called the *trusted host* $\mathsf{TH}_n^{\mathsf{f}}$ that serves the interface. It also runs in a configuration with a user H and an adversary $\mathsf{A}^{\mathsf{ideal}}$ (both modeled as PPT ITM), where $\mathsf{A}^{\mathsf{ideal}}$ may communicate with $\mathsf{TH}_n^{\mathsf{f}}$ at arbitrary points during the protocol. This allows to model the non-determinism in the ideal service f which may be controlled by the adversary.

The security of the protocol π is then defined by requiring that whatever can happen to an arbitrary user H in the real system $Sys_n^{\mathsf{real},\pi}$ could also happen to the same user in the ideal system $Sys_n^{\mathsf{ideal},\mathsf{f}}$, i.e., H cannot distinguish an ideal configuration from a real one. In this case, we say that $Sys_n^{\mathsf{real},\pi}$ is *as secure as* $Sys_n^{\mathsf{ideal},\mathsf{f}}$. The standard argument how to prove this notion of security is a constructive one. Specifically, one shows how to construct for any user H and any adversary $\mathsf{A}^{\mathsf{real}}$ an ideal adversary $\mathsf{A}^{\mathsf{ideal}}$ such that H cannot distinguish the corresponding real configuration from the ideal configuration.

The framework also allows modular composition of protocols by describing a protocol π that runs on top of an ideal service g. We model this as a *hybrid system* $Sys_n^{\mathsf{hybrid},\pi_\mathsf{g}}$ consisting of the real system $Sys_n^{\mathsf{real},\pi}$ for π with an ideal sub-system $Sys_n^{\mathsf{ideal},\mathsf{g}}$. In such a hybrid system, the servers $\mathsf{M}_1^\pi, \ldots, \mathsf{M}_n^\pi$ have access to $Sys_n^{\mathsf{ideal},\mathsf{g}}$ as if they were the user of this system. An important property of the framework is that if a real system $Sys_n^{\mathsf{real},\rho}$ is as secure as $Sys_n^{\mathsf{ideal},\mathsf{g}}$, then the real system $Sys_n^{\mathsf{real},\pi_\rho}$ (where the servers have access to $Sys_n^{\mathsf{real},\rho}$ instead of $Sys_n^{\mathsf{ideal},\mathsf{g}}$) is as secure as $Sys_n^{\mathsf{hybrid},\pi_\mathsf{g}}$. Below, we add a few more details. For a more elaborate treatment of the framework we refer to [23].

Ideal and Real Configurations. On an abstract level, a real or an ideal configuration can be seen as a set of PPT ITM called *machines*, that are connected with each other through communication tapes [18] and may interact. For every communication tape there is exactly one machine that can write to the tape, and another machine that can read from the tape. We also call such a communication tape a *connection*. Figure 1 shows an example of a configuration of the real and the ideal system for a protocol π and a service f, respectively; it shows the connections (denoted by arrows) and the interface (the connections that cross the dashed lines) of the configurations.

Executing a Configuration: The execution of a configuration is called a *run* and is defined as follows.[2] At the beginning of the run, every machine is initialized with the security parameter k. Then, the run proceeds in *steps*. In the first step, a designated machine called the *master scheduler* is activated (for the configurations in this paper, the master scheduler is the adversary). In every step, the currently active machine may read its communication input tapes, perform some computation (possibly involving random choices), and then write a message to one of its communication output tapes. It then either *halts* or *terminates* the activation. In either case, the machine which can read from the tape that has been written to is activated and proceeds with the next step. If this machine has halted before, or no message was written to a tape, then the master scheduler is activated instead. This process continues until the master scheduler halts.

The Network: The network NET provides authentic communication among the servers with scheduling determined by the adversary. It connects to every server M_i^π through connections $\mathsf{net\_out}_i$ and $\mathsf{net\_in}_i$, and connects to the adversary $\mathsf{A}^{\mathsf{real}}$ through the connections $\mathsf{to\_Adv}_{net}$ and $\mathsf{from\_Adv}_{net}$, and works as follows. A server M_i^π can *send a message* m *to* M_j^π by sending the message (m, j) on the connection $\mathsf{net\_out}_i$ to NET. If this happens, NET stores (m, i, j) in a collection M, and sends the tuple (m, i, j) on the connection $\mathsf{to\_Adv}_{net}$ to the adversary. Similarly, $\mathsf{A}^{\mathsf{real}}$ can schedule a message m to be delivered from M_i^π to M_j^π by sending (m, i, j) on the connection $\mathsf{from\_Adv}_{net}$. If this happens, and if $(m, i, j) \in M$, then NET removes (m, i, j) from M, and outputs (m, i) to M_j^π; we also say M_j^π *receives* m *from* M_i^π. We say that the adversary *delivers all messages among* M_i^π *and* M_j^π, if at the end of the run, the collection M does not contain any tuples (m, i, j) or (m, j, i) for any message m.

The Interface: The interface of a real and an ideal system comprises the connections in_i, out_i, and $\mathsf{cor\_in}_i$ for $i \in [1, n]$. The first two connections are used to invoke the service(s) provided by the systems, and to receive service-specific output from the systems, respectively. The connections $\mathsf{cor\_in}_i$ for $i \in [1, n]$ are used to model attacks of the adversary on a server i. These connections have to be part of the interface because attacks on the servers will affect the service that a system provides. For now, we describe the *weak corruption* model, where the adversary may only *crash* a server i, modeled as a message (crash) sent on the connection $\mathsf{cor\_in}_i$ (see Section 4 for the *strong corruption model*, where also break-ins are allowed). If this happens in a real configuration, then M_i^π outputs (crash) at $\mathsf{cor\_out}_i$ and halts. If this happens in an ideal configuration, then $\mathsf{TH}_n^{\mathsf{f}}$ outputs (crash, i) at $\mathsf{to\_Adv}_{th}$.

Other Connections: The trusted host $\mathsf{TH}_n^{\mathsf{f}}$ also connects to $\mathsf{A}^{\mathsf{ideal}}$ through connections $\mathsf{to\_Adv}_{th}$ and $\mathsf{from\_Adv}_{th}$. These connections are used to model the non-determinism

[2] The model of execution described here is not as general as the model described in the original work [23], but will be sufficient for our purpose.

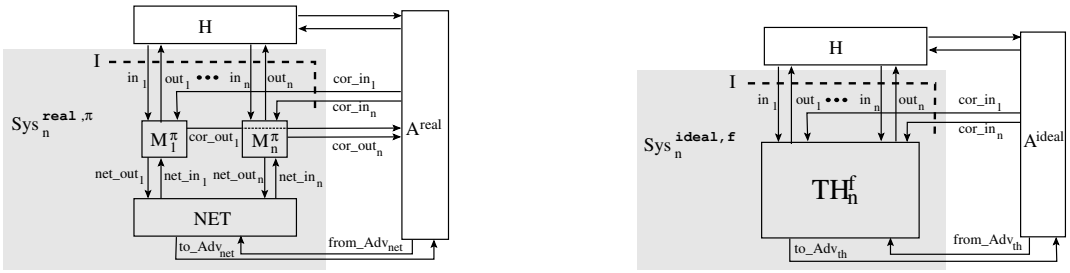

Figure 1: Configurations of the real and ideal systems for protocol π and service f, respectively.

in the ideal service f, which may be controlled by the adversary. Furthermore, every server M_i^π connects to $\mathsf{A}^{\mathsf{real}}$ through connections $\mathsf{cor\_out}_i$. These connections are used to model effects of an attack, e.g., in Section 4 we will use these connections for revealing server internal data to the adversary as a result of a break-in.

We are now ready to state the definition for a real system $Sys_n^{\mathsf{real},\pi}$ to be *as secure as* an ideal system $Sys_n^{\mathsf{ideal},\mathsf{f}}$. For this purpose, let $\mathsf{V}^{\mathsf{real},\pi}_{n,\mathsf{H},\mathsf{A}^{\mathsf{real}}}(k)$ denote the probability distribution of the view of H (the internal state of H and all messages that H sees) in a run of $Sys_n^{\mathsf{real},\pi}$ with security parameter k, configured with H and $\mathsf{A}^{\mathsf{real}}$, and let $\mathsf{V}^{\mathsf{ideal},\mathsf{f}}_{n,\mathsf{H},\mathsf{A}^{\mathsf{ideal}}}(k)$ be defined analogously.

Definition 1 *We say that $Sys_n^{\mathsf{real},\pi}$ is* as secure as *$Sys_n^{\mathsf{ideal},\mathsf{f}}$, if for all users H, and all real adversaries $\mathsf{A}^{\mathsf{real}}$, there exists an ideal adversary $\mathsf{A}^{\mathsf{ideal}}$ such that the distribution ensembles $\{\mathsf{V}^{\mathsf{real},\pi}_{n,\mathsf{H},\mathsf{A}^{\mathsf{real}}}(k)\}_{k\in\mathbb{N}}$ and $\{\mathsf{V}^{\mathsf{ideal},\mathsf{f}}_{n,\mathsf{H},\mathsf{A}^{\mathsf{ideal}}}(k)\}_{k\in\mathbb{N}}$ are computationally indistinguishable.*

Composition. The framework allows to describe protocols in a modular way, i.e., a protocol π may build on a sub-protocol ρ. Such a composition is modeled as a real system $Sys_n^{\mathsf{real},\pi_\rho}$ comprising the two systems $Sys_n^{\mathsf{real},\pi}$ and $Sys_n^{\mathsf{real},\rho}$, where the servers of $Sys_n^{\mathsf{real},\pi}$ have access to the sub-system $Sys_n^{\mathsf{real},\rho}$ as if they were the user of this system. To ensure that an attack on a server M_i^π also affects the sub-system, we assume that the connection $\mathsf{cor\_out}_i$ of a server M_i^π is linked to the connection $\mathsf{cor\_in}_i$ of server M_i^ρ, i.e., if M_i^π outputs (crash) on $\mathsf{cor\_out}_i$, then M_i^ρ receives (crash) on $\mathsf{cor\_in}_i$. The sub-system may also be an ideal system, in which case we call the entire system *hybrid*. The following *composition theorem* is an important property of the framework [23].

Theorem 1 (Composition Theorem [23]) *If a hybrid system $Sys_n^{\mathsf{hybrid},\pi_\mathsf{g}}$ consisting of a real system $Sys_n^{\mathsf{real},\pi}$ with sub-system $Sys_n^{\mathsf{ideal},\mathsf{g}}$ is as secure as an ideal system $Sys_n^{\mathsf{ideal},\mathsf{f}}$, and if a real system $Sys_n^{\mathsf{real},\rho}$ is as secure as the ideal system $Sys_n^{\mathsf{ideal},\mathsf{g}}$, then the real system $Sys_n^{\mathsf{real},\pi_\rho}$ consisting of $Sys_n^{\mathsf{real},\pi}$ with sub-system $Sys_n^{\mathsf{real},\rho}$ is at least as secure the ideal system $Sys_n^{\mathsf{ideal},\mathsf{f}}$.*

Complexity Measures. We will measure the complexity of a protocol π in terms of its *expected message complexity* $\mathsf{M}(\pi)$, and its *expected round complexity* $\mathsf{R}(\pi)$. The first measure represents the bandwidth required by the protocol, and is defined as follows. Let $M_{\mathsf{H},\mathsf{A}^{\mathsf{real}}}(\pi)$ denote an upper bound on the expected number of messages that the servers send across the network in a run of $Sys_n^{\mathsf{real},\pi}$ configured with H and $\mathsf{A}^{\mathsf{real}}$, where the expectation is taken over the random choices of the servers. Then, $\mathsf{M}(\pi)$ is the maximum of $M_{\mathsf{H},\mathsf{A}^{\mathsf{real}}}(\pi)$ over all users H and adversaries $\mathsf{A}^{\mathsf{real}}$.

The round complexity measures the running time of the protocol. To define it, we assign round numbers to the messages sent by servers across the network as follows. Note that a server only sends a message m across the network in response to an input from the user or in response to a message m' from the network. In the first case, we assign round number 0 to m, and in the second case, we assign round number $r+1$ to m, where r is the round number of m'. Let $R_{\mathsf{H},\mathsf{A}^{\mathsf{real}}}(\pi)$ denote an upper bound on the expected highest round number assigned to a message in a run of $Sys_n^{\mathsf{real},\pi}$ configured with H and $\mathsf{A}^{\mathsf{real}}$, where the expectation is taken over the random choices of the servers. Then, $\mathsf{R}(\pi)$ is the maximum of $R_{\mathsf{H},\mathsf{A}^{\mathsf{real}}}(\pi)$ over all users H and adversaries $\mathsf{A}^{\mathsf{real}}$.

2.2 The Ideal System for Group Key Exchange

Our ideal system $Sys_n^{\mathsf{ideal},\mathsf{gke}}$ for group key exchange models how n servers repeatedly establish a session key. In particular, we say a server i *starts a session with tag ID* when an input (start, ID) occurs at in_i (ID is an arbitrary bit string and represents a unique identifier for the session). Similarly, we say a server i *finishes a session with tag ID* when an output $(\mathsf{finish}, ID, key)$ occurs at out_i.

Our trusted host $\mathsf{TH}_n^{\mathsf{gke}}$ models the traditional security properties that one expects from a GKE protocol. In particular, it guarantees that the session key of every session ID is generated independently at random; this property is sometimes also called *key freshness*. Furthermore, it guarantees *mutual key authentication*, which means that every server computes the same key in a session ID. Definition 2 below captures these ideas more formally. We do not yet address *forward secrecy*, as this becomes only an issue if the adversary can break into a server and learn its internal state (we discuss this model in Section 4).

Definition 2 *The ideal system $Sys_n^{\mathsf{ideal},\mathsf{gke}}$ for* group key exchange *consists of the trusted host $\mathsf{TH}_n^{\mathsf{gke}}$ given by the following transition rules:*

Init: *At system initialization, it sets $S[i] \leftarrow \emptyset$, $F[i] \leftarrow \emptyset$ for $i \in [1,n]$, $\kappa[ID] \leftarrow \perp$ for all ID.*

Start: *When a server i starts a session with tag ID, the trusted host adds ID to the set $S[i]$, and outputs the message $(\mathsf{started}, ID, i)$ to the adversary.*

Finish: *When the adversary inputs (finish, ID, i) where $ID \in S[i] \setminus F[i]$, then $\mathsf{TH}_n^{\mathsf{gke}}$ first adds ID to the set $F[i]$. Next,*

if $\kappa[ID] = \perp$, it chooses $\kappa[ID]$ at random over $\{0,1\}^k$. Finally, it outputs $(\mathsf{deliver}, ID, \kappa[ID])$ *at* out_i.

Recall that our goal is to build a GKE protocol π that is not only secure, but also guarantees to terminate for every server even if up to t_c servers crash. This is captured by the following definition of a t_c-resilient group key exchange protocol.

Definition 3 *We call a protocol π a t_c-*resilient group key exchange protocol*, if $Sys_n^{\mathsf{real},\pi}$ is as secure as $Sys_n^{\mathsf{ideal},\mathsf{gke}}$, and if for every run of a configuration of $Sys_n^{\mathsf{real},\pi}$ where at most t_c servers crash, the following holds: If all non-crashed servers start a session ID, then they all finish session ID, provided that the adversary delivers all messages among non-crashed servers.*

2.3 The Ideal System for Consensus

In a *consensus* protocol, every server receives as input a bit string of some length $l(k)$, and produces as output some bit string of length $l(k)$. The goal is that all servers output the same bit string, and that this bit string corresponds to the input of at least one server.

Below we give the ideal system for consensus, which will serve as building block in our construction for a GKE protocol. It models how n servers repeatedly and concurrently agree on bit strings of some length $l(k)$, where every consensus instance is identified by a tag ID. We model that a server starts a consensus instance with tag ID and input v by a message $(\mathsf{propose}, ID, v)$ that occurs at in_i. If this happens, we also say *server i proposes v for ID*. Similarly, we model that a server terminates a consensus with tag ID and value v' by a message $(\mathsf{decide}, ID, v')$ that occurs at out_i. In this case, we say *server i decides v' in ID*.

Definition 4 *The ideal system $Sys_n^{\mathsf{ideal},\mathsf{cons}}$ for* consensus *consists of the trusted host $\mathsf{TH}_n^{\mathsf{cons}}$ given by the following transition rules:*

Init: *At system initialization, $\mathsf{TH}_n^{\mathsf{cons}}$ sets $P[ID] \leftarrow \emptyset$ and $\delta[ID] \leftarrow \perp$ for all ID.*

Propose: *If a server i proposes v_i for ID, $\mathsf{TH}_n^{\mathsf{cons}}$ adds the tuple (i, v_i) to the set $P[ID]$, and outputs* $(\mathsf{propose}, ID, i, v)$ *to the adversary.*

Decide: *When $\mathsf{TH}_n^{\mathsf{cons}}$ receives* $(\mathsf{decide}, ID, i, v)$ *from the adversary, it verifies (by consulting $P[ID]$ and $\delta[ID]$) that*

- *server i has proposed some value for ID*
- *at least one server proposed v for ID*
- *no other server has decided another value for ID*

If all checks succeed, it sets $\delta[ID] \leftarrow v$, and outputs (decide, ID, v) *to the user at* out_i.

We will need a consensus protocol κ that is not only secure, but also guarantees to terminate for every server even if up to t_c servers crash. The following definition captures this more formally.

Definition 5 *We call a protocol π a t_c-*resilient consensus *protocol, if $Sys_n^{\mathsf{real},\pi}$ is as secure as $Sys_n^{\mathsf{ideal},\mathsf{cons}}$, and if for every run of a configuration of $Sys_n^{\mathsf{real},\pi}$ where at most t_c servers crash, the following holds: If all non-crashed servers propose a value for some ID, then they all decide some value for ID, provided that the adversary delivers all messages among non-crashed servers.*

The best-known randomized asynchronous t_c-resilient consensus protocol can be derived from the protocol of Canetti and Rabin [13], which actually solves the harder problem of *Byzantine agreement*, where the servers may not only crash but behave arbitrarily, and is unconditionally secure. A consensus protocol can be derived from this as described in [4, Section 14.3.2]. It uses an expected number of $O(n^3)$ messages, proceeds in expected $O(1)$ rounds, and has resilience $t_c < n/3$. Assuming a trusted dealer that initializes the system and working in a realistic model with a computationally bounded adversary, there exist cryptographic protocols due to Cachin et al. [10, 9] and to Nielsen [22], which use $O(n^2)$ messages and $O(1)$ rounds on average, and have optimal resilience $t_c < n/2$.

As mentioned in the introduction, it is also possible to implement consensus in the *failure-detector model* [14], where some very efficient protocols exist. A failure detector is a local module available to every server that periodically outputs a list of servers that it suspects to have crashed and is usually based on a timing assumption. We do not pursue this further and focus on the fully asynchronous model, but note that in certain practical settings, such protocols might actually be more efficient than the fully asynchronous protocols mentioned.

3. IMPLEMENTATION

3.1 Protocol ω for GKE

We now describe our protocol ω for GKE. It builds on a sub-system for consensus, and on a semantically secure encryption scheme [17] $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ for elements of $\{0,1\}^k$ with key-generation algorithm $\mathcal{K}$, encryption algorithm $\mathcal{E}$, and decryption algorithm $\mathcal{D}$. In the following, all computations are done over $\mathbb{F}_{2^k}$ if not indicated otherwise.

When a server i starts a session with tag ID, it first chooses a contribution y_i randomly from $\{0,1\}^k$; the goal is to compute the session key as $sk = \sum_{j \in G} y_i$ for some set G of $n - t_c$ servers. It then runs $\mathcal{K}$ to generate a pair of public key/private key (p_i, s_i), and sends p_i to every other server.

When server i receives such a public key p_j from another server j, it sends the contribution value y_i encrypted under p_j to server j. Once it has received the contribution values $y_{u_1}, \ldots, y_{u_{n-t_c}}$ of $n - t_c$ servers like this, it computes the differences $d_1 \leftarrow y_{u_1} - y_{u_2}, d_2 \leftarrow y_{u_2} - y_{u_3}, \ldots, d_{n-t_c} \leftarrow y_{u_{n-t_c}} - y_{u_1}$, and proposes the sequences $\langle u_1, \ldots, u_{n-t_c} \rangle$ and $\langle d_1, \ldots, d_{n-t_c} \rangle$ for $ID|\mathsf{cs}$ in the consensus sub-system, where cs is an arbitrary constant string. Note that the difference between any pair of contribution values may be leaked to the adversary through this, but since no other information is revealed, all contribution values remain secret.

When a server i decides two sequences $\langle \bar{u}_1, \ldots, \bar{u}_{n-t_c} \rangle$ and $\langle \bar{d}_1, \ldots, \bar{d}_{n-t_c} \rangle$ in the consensus instance $ID|\mathsf{cs}$, it computes the session key as follows. It first chooses an arbitrary index $m \in [1, n-t_c]$ such that it has received $y_{\bar{u}_m}$ before (notice that such an m exists, as it has received at least $n - t_c$ values y_j at this point, and it holds that $t_c < n/2$). It then computes the session key $sk = (\sum_{j=1}^{n-t_c-1} j\bar{d}_{m+j}) + (n-t_c) y_{\bar{u}_m}$, where $\bar{d}_l \equiv \bar{d}_{l-(n-t_c)}$ for $n - t_c < l < 2(n - t_c)$. The detailed protocol is given in Algorithm 1. In this description, we make the convention that $\bar{d}_l \equiv \bar{d}_{l-(n-t_c)}$ and $u_l \equiv u_{l-(n-t_c)}$ for $n - t_c < l < 2(n - t_c)$.

It is easy to see that every server terminates. It is also easy to verify that every server computes the same session key $sk =$

upon *initialization*:

$y_l \leftarrow \perp$ for $l \in [1,n]$; $u_m \leftarrow \perp$ for $m \in [1, n-t_c]$;
$p_i \leftarrow \perp$, $s_i \leftarrow \perp$, $ctr \leftarrow 0$

upon *input* (start, ID)**:**

choose y_i uniformly at random from $\{0,1\}^k$
$(p_i, s_i) \leftarrow \mathcal{K}$
send $(\mathsf{enc\_key}, ID, p_i)$ to every server

upon *receiving* $(\mathsf{enc\_key}, ID, p_j)$ **from server** j**:**

send $(\mathsf{key\_part}, ID, z_{ij})$ to server j, where $z_{ij} \leftarrow \mathcal{E}_{p_j}(y_i)$

upon *receiving* $(\mathsf{key\_part}, ID, z_{ji})$ **from server** j**:**

$ctr \leftarrow ctr + 1$; $u_{ctr} \leftarrow j$; $y_j \leftarrow \mathcal{D}_{s_i}(z_{ji})$
if $ctr = n - t_c$ **then**
  $d_j \leftarrow y_{u_j} - y_{u_{j+1}}$ for $j \in [1, n-t_c]$;
  propose $(\langle u_1, \ldots, u_{n-t_c}\rangle, \langle d_1, \ldots, d_{n-t_c}\rangle)$ for $ID|\mathsf{cs}$

upon *deciding* $(\langle \bar{u}_1, \ldots, \bar{u}_{n-t_c}\rangle, \langle \bar{d}_1, \ldots, \bar{d}_{n-t_c}\rangle)$ **in** $ID|\mathsf{cs}$**:**

choose $m \in [1, n-t_c]$ such that $y_{\bar{u}_m} \neq \perp$
$sk \leftarrow \sum_{j=1}^{n-t_c-1} j\bar{d}_{m+j} + (n-t_c)y_{\bar{u}_m}$
output $(\mathsf{deliver}, ID, sk)$

Algorithm 1: Protocol ω for server i, implementing GKE with crashes.

$\sum_{j=1}^{n-t_c} y_{\bar{u}_j}$, regardless of which m it chooses. Finally, since all contribution values remain secret (as argued above), the same holds for the session key.

This technique is a fault-tolerant abstraction of the GKE protocol of Burmester and Desmedt [8]. In their protocol the public-key encryption scheme is instantiated with the ElGamal scheme. The servers choose y_i in the same way, and then jointly compute the values $g^{d_j} = g^{y_j y_{j-1} - y_j y_{j+1}}$ for $j \in [1,n]$ (here, g is a generator of a multiplicative group of prime order q). The session key is then derived from these values (and one contribution value y_j) as $sk = g^{y_1 y_2 + y_2 y_3 + \cdots + y_n y_1}$. We prove the following theorem in the next section.

Theorem 2 *If κ is a t_c-resilient consensus protocol, then the real system $Sys_n^{\mathsf{real},\omega_\kappa}$ consisting of $Sys_n^{\mathsf{real},\omega}$ with sub-system $Sys_n^{\mathsf{real},\kappa}$ is a t_c-resilient group key exchange protocol.*

Further Improvements. For repeatedly generating session keys, there is a faster way than running protocol ω for every session. The idea is to use a family $\Psi_k = \{\psi_i\}_{i \in \{0,1\}^k}$ of *pseudorandom functions* [16], where a function ψ_i maps bit strings used for the session tags ID to bit strings of length k. Pseudorandom function families have the property that one cannot distinguish $\psi_i(ID)$ from a value randomly chosen from $\{0,1\}^k$ without knowing the index i. This allows the servers to repeatedly generate session keys by running the protocol ω only once to get a secret index s. The session key for a session with tag ID can then simply be computed as $\psi_s(ID)$. We remark that this construction is only secure in the weak corruption model, where the adversary cannot break into a server and learn the index s.

3.2 Security Analysis

To establish Theorem 2 and prove the security of Protocol ω, we have to show that for a t_c-resilient consensus protocol κ, $Sys_n^{\mathsf{real},\omega_\kappa}$ is as secure as the ideal system $Sys_n^{\mathsf{ideal},\mathsf{gke}}$, and that $Sys_n^{\mathsf{real},\omega_\kappa}$ is live, i.e., that if all servers U that do not crash during a run of $Sys_n^{\mathsf{real},\omega_\kappa}$ start a session ID, then all servers U finish session ID, provided that at most t_c servers crash and all messages among non-crashed servers are delivered.

To show liveness of $Sys_n^{\mathsf{real},\omega_\kappa}$, we argue as follows. By the assumption that at most t_c servers crash, it follows that every server in U receives $n - t_c$ contribution values y_j and proposes some values for $ID|\mathsf{cs}$. By the assumption that κ is a t_c-resilient consensus protocol, every server therefore also decides some sequences $\langle \bar{u}_1, \ldots, \bar{u}_{n-t_c}\rangle$ and $\langle \bar{d}_1, \ldots, \bar{d}_{n-t_c}\rangle$ for $ID|\mathsf{cs}$. By construction, every server knows at that step $n - t_c$ different values $\{y_j\}$. By $t_c < n/2$ it follows that for every server there exists at least one index $m \in [1, n-t_c]$ such that it knows the contribution value $y_{\bar{u}_m}$. Hence, all servers in U will be able to compute the key and finish the session ID.

To show security of $Sys_n^{\mathsf{real},\omega_\kappa}$, we consider the hybrid system $Sys_n^{\mathsf{hybrid},\omega_{\mathsf{cons}}}$ consisting of the real system $Sys_n^{\mathsf{real},\omega}$ with the ideal sub-system $Sys_n^{\mathsf{ideal},\mathsf{cons}}$ for consensus. Note that it suffices to show that $Sys_n^{\mathsf{hybrid},\omega_{\mathsf{cons}}}$ is as secure as $Sys_n^{\mathsf{ideal},\mathsf{gke}}$, since the security of $Sys_n^{\mathsf{real},\omega_\kappa}$ then follows by the composition theorem and the assumption that κ is a secure consensus protocol.

We now show that $Sys_n^{\mathsf{hybrid},\omega_{\mathsf{cons}}}$ is as secure as $Sys_n^{\mathsf{ideal},\mathsf{gke}}$, using a constructive argument. In particular, we provide for every user H, and every adversary $\mathsf{A}^{\mathsf{hybrid}}$ against the hybrid system $Sys_n^{\mathsf{hybrid},\omega_{\mathsf{cons}}}$ the construction of an adversary $\mathsf{A}^{\mathsf{ideal}}$ against the ideal system such that the views of H in a run of the ideal and hybrid system configured with H, $\mathsf{A}^{\mathsf{ideal}}$, and $\mathsf{A}^{\mathsf{hybrid}}$, respectively, are computationally indistinguishable.

Recall that the view of H in a run of the hybrid system also comprises messages that it exchanges with $\mathsf{A}^{\mathsf{hybrid}}$. In order to ensure that these messages have the same distribution as the messages that H exchanges with $\mathsf{A}^{\mathsf{ideal}}$ in a run of the ideal system, we construct $\mathsf{A}^{\mathsf{ideal}}$ using a technique called *black-box simulation*. Specifically, we assume that we are given the hybrid adversary $\mathsf{A}^{\mathsf{hybrid}}$ as a black-box, and then construct $\mathsf{A}^{\mathsf{ideal}}$ out of a simulator SIM and the given black-box $\mathsf{A}^{\mathsf{hybrid}}$. The idea is that the simulator feeds the black box $\mathsf{A}^{\mathsf{hybrid}}$ with a simulated view of a run of the hybrid system, where SIM must compute this view based on the information it receives from $\mathsf{TH}_n^{\mathsf{gke}}$. If the simulated view is indistinguishable from what the hybrid adversary would see in a corresponding hybrid run, we can use the outputs of the black-box $\mathsf{A}^{\mathsf{hybrid}}$ to simulate the messages exchanged with H. The construction is illustrated in Figure 2 (w.l.o.g. we only show how to build $\mathsf{A}^{\mathsf{ideal}}$ for a configuration where H interacts with the ideal system through the entire interface).

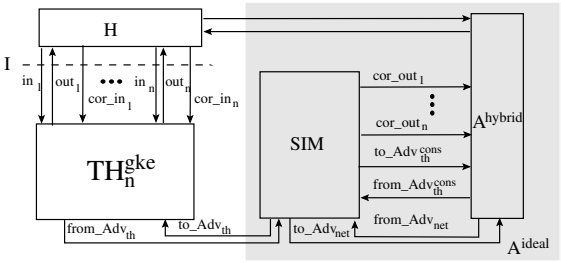

Figure 2: The Construction of $\mathsf{A}^{\mathsf{ideal}}$ with Black Box Access to $\mathsf{A}^{\mathsf{hybrid}}$

The Simulator. As usual in black-box simulations, we construct the simulator SIM as the combination of a simulated

user $\bar{\mathsf{H}}$ and a simulated hybrid system $\bar{Sys}_n^{\mathsf{hybrid},\omega_{\mathsf{cons}}}$. $\bar{\mathsf{H}}$ interacts with $\mathsf{TH}_n^{\mathsf{gke}}$ through $\mathsf{to\_Adv}_{th}$ and $\mathsf{from\_Adv}_{th}$, and with the simulated hybrid system $\bar{Sys}_n^{\mathsf{hybrid},\omega_{\mathsf{cons}}}$ through its entire interface. $\bar{Sys}_n^{\mathsf{hybrid},\omega_{\mathsf{cons}}}$ interacts with $\mathsf{A}^{\mathsf{hybrid}}$ through connections $\mathsf{to\_Adv}_{net}$, $\mathsf{from\_Adv}_{net}$, $\mathsf{to\_Adv}_{\mathsf{TH}_n^{\mathsf{cons}}}$, $\mathsf{from\_Adv}_{\mathsf{TH}_n^{\mathsf{cons}}}$, and $\mathsf{cor\_out}_j$ for $j \in [1,n]$. $\bar{\mathsf{H}}$ provides the same inputs to $\bar{Sys}_n^{\mathsf{hybrid},\omega_{\mathsf{cons}}}$ as H provides to $\mathsf{TH}_n^{\mathsf{gke}}$ and, moreover, schedules $\mathsf{TH}_n^{\mathsf{gke}}$ such that whenever a server $\bar{\mathsf{M}}_i^{\omega_{\mathsf{cons}}}$ outputs a session key for ID, then $\mathsf{TH}_n^{\mathsf{gke}}$ also outputs a session key for ID at out_i. More precisely, $\bar{\mathsf{H}}$ works as follows.

upon *receiving* $(\mathsf{started}, ID, i)$ **from** $\mathsf{TH}_n^{\mathsf{gke}}$**:**
send (start, ID) to $\bar{\mathsf{M}}_i^{\omega_{\mathsf{cons}}}$
upon *receiving* $(\mathsf{deliver}, ID, k)$ **from** $\bar{\mathsf{M}}_i^{\omega_{\mathsf{cons}}}$**:**
send (finish, ID, i) to $\mathsf{TH}_n^{\mathsf{gke}}$
upon *receiving* (crash, i) **from** $\mathsf{TH}_n^{\mathsf{gke}}$**:**
send (crash) to $\bar{\mathsf{M}}_i^{\omega_{\mathsf{cons}}}$

Comparing the Views. To show that for every H and $\mathsf{A}^{\mathsf{hybrid}}$, the view of H in a run of the hybrid system $Sys_n^{\mathsf{hybrid},\omega_{\mathsf{cons}}}$ configured with H and $\mathsf{A}^{\mathsf{hybrid}}$ is indistinguishable from the view of H in a run of the ideal system $Sys_n^{\mathsf{ideal},\mathsf{gke}}$ configured with the adversary $\mathsf{A}^{\mathsf{ideal}}$, it suffices to show that the *joint* view of H and $\mathsf{A}^{\mathsf{hybrid}}$ in a hybrid and an ideal run, respectively, are indistinguishable (note that by construction of $\mathsf{A}^{\mathsf{ideal}}$, this joint view is well-defined in the ideal run).

We argue inductively over the steps of the runs. The base case, i.e., indistinguishability of the initial states, follows by construction. It remains to show that if the joint view of H and $\mathsf{A}^{\mathsf{hybrid}}$ in a hybrid run up to a step $l > 0$ is indistinguishable from their view in an ideal run up to step l (induction hypothesis), then the same holds for step $l+1$ (inductive step).

First note that the joint view of H and $\mathsf{A}^{\mathsf{hybrid}}$ only changes if one of them either sends a message, or receives a message. In the first case, the inductive step follows directly by the induction hypothesis. In the second case, we argue as follows.

By the protocol specification, the messages received by H and $\mathsf{A}^{\mathsf{hybrid}}$ for different sessions are statistically independent of each other. Hence, it suffices to show that the probability distributions of the received messages associated with a *single* session in an ideal and a hybrid run, respectively, are indistinguishable. We first investigate the distribution in an ideal run. Let y_i, p_i, s_i, and z_{ij} for $j \in [1,n]$ denote the values computed by $\bar{\mathsf{M}}_i^{\omega_{\mathsf{cons}}}$ during the run. Then, the values contained in messages received by H and $\mathsf{A}^{\mathsf{hybrid}}$ up to and including step $l+1$ of the run are a subset of $\{\langle \bar{u}_1, \ldots, \bar{u}_{n-t_c}\rangle\} \cup \{sk, p_i, z_{ij} \mid i,j \in [1,n]\}$, and a subset of any linear combination of the values $y_1 - y_2, y_2 - y_3, \ldots, y_n - y_1$. We denote these values by the random variable $\mathsf{V}_k^{\mathsf{ideal}}$. Notice that we may ignore the index vectors $\langle u_1, \ldots, u_{n-t_c}\rangle$ proposed by the servers, as they are determined by the scheduling of $\mathsf{A}^{\mathsf{hybrid}}$, and thus, identically distributed in an ideal and a hybrid run. The distribution of $\mathsf{V}_k^{\mathsf{ideal}}$ is as follows:

$$y_i \xleftarrow{R} \{0,1\}^k,\ (p_i, s_i) \leftarrow \mathcal{K}, z_{ij} \leftarrow \mathcal{E}_{p_j}(y_i) \text{ for } i,j \in [1,n];$$
$$\langle \bar{u}_1, \ldots, \bar{u}_{n-t_c}\rangle \xleftarrow{\mathsf{A}^{\mathsf{hybrid}}} [1,n]^{n-t_c};\ sk \xleftarrow{R} \{0,1\}^k$$
$$\mathsf{V}_k^{\mathsf{ideal}} = \begin{pmatrix} p_1, \ldots, p_n, z_{11}, \ldots, z_{1n}, \ldots z_{n1}, \ldots, z_{nn}, sk, \\ y_1 - y_2, y_2 - y_3, \ldots, y_n - y_1, \bar{u}_1, \ldots, \bar{u}_{n-t_c} \end{pmatrix}$$

In a hybrid run, H and $\mathsf{A}^{\mathsf{hybrid}}$ receive the same messages, but with a different distribution. In particular, the session key is not drawn at random from $\{0,1\}^k$ but is the sum of the $n - t_c$ values $y_{\bar{u}_i}$. Specifically, the distribution $\mathsf{V}_k^{\mathsf{hybrid}}$ of the received messages in a hybrid run is as follows:

$$y_i \xleftarrow{R} \{0,1\}^k,\ (p_i, s_i) \leftarrow \mathcal{K}, z_{ij} \leftarrow \mathcal{E}_{p_j}(y_i) \text{ for } i,j \in [1,n];$$
$$\langle \bar{u}_1, \ldots, \bar{u}_{n-t_c}\rangle \xleftarrow{\mathsf{A}^{\mathsf{hybrid}}} [1,n]^{n-t_c};\ sk \leftarrow \sum_{l=1}^{n-t_c} y_{\bar{u}_l}$$
$$\mathsf{V}_k^{\mathsf{hybrid}} = \begin{pmatrix} p_1, \ldots, p_n, z_{11}, \ldots, z_{1n}, \ldots z_{n1}, \ldots, z_{nn}, sk, \\ y_1 - y_2, y_2 - y_3, \ldots, y_n - y_1, \bar{u}_1, \ldots, \bar{u}_{n-t_c} \end{pmatrix}$$

Let U_k denote the uniform distribution over $\{0,1\}^k$. To show that $\mathsf{V}_k^{\mathsf{ideal}} \approx \mathsf{V}_k^{\mathsf{hybrid}}$, it is sufficient to show that from $\mathsf{V}_k^{\mathsf{ideal}}$ one can compute values $\bar{y}_1, \ldots, \bar{y}_n$ such that the following holds:

$$\bar{y}_i - \bar{y}_{(i \bmod n)+1} = y_i - y_{(i \bmod n)+1} \text{ for } i \in [1,n] \quad (1)$$
$$sk = \bar{y}_{\bar{u}_1} + \cdots + \bar{y}_{\bar{u}_{n-t_c}} \quad (2)$$
$$\bar{y}_i \approx U_k \text{ for } i \in [1,n] \quad (3)$$
$$\mathcal{E}_{p_j}(\bar{y}_i) \approx \mathcal{E}_{p_j}(y_i) \text{ for } i,j \in [1,n] \quad (4)$$

The values $\bar{y}_i$ can be computed as follows. For $i \in [1,n]$, let δ_i denote the value $y_i - y_{(i \bmod n)+1}$ from $\mathsf{V}_k^{\mathsf{ideal}}$, and let $\bar{u}_{n-t_c+1} \equiv \bar{u}_1$. Furthermore, for $i \in [1, n-t_c]$, let $\bar{\delta}_i$ denote the value $y_{\bar{u}_i} - y_{\bar{u}_{i+1}}$. Note that every such value $\bar{\delta}_i$ can be computed as follows:

$$\bar{\delta}_i = \begin{cases} \delta_{\bar{u}_i} + \delta_{\bar{u}_i+1} + \cdots + \delta_{\bar{u}_{i+1}-1} & \text{if } \bar{u}_i < \bar{u}_{i+1}, \\ -\delta_{\bar{u}_{i+1}} - \delta_{\bar{u}_{i+1}+1} - \cdots - \delta_{\bar{u}_i-1} & \text{otherwise.} \end{cases}$$

One can now compute $\bar{y}_{\bar{u}_1}$ as $(sk - \sum_{m=1}^{n-t_c} m\bar{\delta}_m)(n-t_c)^{-1}$. The remaining values $\bar{y}_i$ for $i \in [1,n] \setminus \{\bar{u}_1\}$ can be computed as follows:

$$\bar{y}_i = \begin{cases} \bar{y}_{\bar{u}_1} + \delta_i + \delta_{i+1} + \cdots + \delta_{\bar{u}_1-1} & \text{if } \bar{u}_1 > i, \\ \bar{y}_{\bar{u}_1} - \delta_{\bar{u}_1} - \delta_{\bar{u}_1+1} - \cdots - \delta_{i-1} & \text{if } \bar{u}_1 < i. \end{cases}$$

It is easy to verify that the computed values $\bar{y}_i$ satisfy (1) and (2). Furthermore, (3) holds because y_i for $i \in [1,n]$ and sk are uniformly distributed over $\{0,1\}^k$. Finally, (4) holds by the semantic security of the encryption scheme used. □

3.3 Efficiency Analysis

We now analyze the expected message and round complexities of our construction for generating a single session key. In the first paragraph, we give the complexities for two concrete instantiations of consensus protocols found in the literature. In the second paragraph we show that our construction is almost optimal in the sense that given an optimal consensus protocol, our construction yields an almost optimal GKE protocol.

Concrete Efficiency. Let ω_κ denote our GKE protocol ω with sub-protocol κ for consensus. Recall that $\mathsf{R}(\pi)$ and $\mathsf{M}(\pi)$ denote the expected round and message complexities, respectively, of a protocol π (cf. Section 2.1). It is easy to see that $\mathsf{R}(\omega_\kappa) = 2 + \mathsf{R}(\kappa)$ and $\mathsf{M}(\omega_\kappa) = 2n^2 + \mathsf{M}(\kappa)$. The best known solution (in terms of round complexity) for GKE without failures [8] proceeds in two rounds and uses $2n + n^2$ messages; hence, the price we pay for tolerating crashes is essentially a single consensus execution.

Below we give the asymptotic complexities of ω_κ for some known protocols κ for consensus. Specifically, we consider the consensus protocol κ_{CR93} of Canetti and Rabin [13] and the consensus protocol κ_{CKS00} of Cachin et al. [10], simplified according to the remark in Section 2.3.

κ	$\mathsf{M}(\omega_\kappa)$	$\mathsf{R}(\omega_\kappa)$	resilience	dealer
κ_{CR93}	$O(n^3)$	$O(1)$	$t_c < n/3$	no
κ_{CKS00}	$O(n^2)$	$O(1)$	$t_c < n/2$	yes

Optimality of the Construction. We now investigate the optimality of our protocol in terms of the round and message complexities. In particular, let ω_{opt} denote an optimal protocol for GKE and let κ_{opt} denote an optimal protocol for consensus — optimal either in round complexity or in message complexity, respectively, depending on the context. Let $\omega_{\kappa_{opt}}$ denote our GKE protocol ω with sub-protocol κ_{opt} for consensus. We want to know how close the efficiency of $\omega_{\kappa_{opt}}$ is to the efficiency of ω_{opt}. We show that

$$\mathsf{R}(\omega_{\kappa_{opt}}) \leq \mathsf{R}(\omega_{opt}) + 8 \quad (5)$$

$$\mathsf{M}(\omega_{\kappa_{opt}}) \leq \mathsf{M}(\omega_{opt}) + 8n^2 \quad (6)$$

We argue as follows. By construction of Protocol ω, we have $\mathsf{R}(\omega_{\kappa_{opt}}) = 2 + \mathsf{R}(\kappa_{opt})$ and $\mathsf{M}(\omega_{\kappa_{opt}}) = 2n^2 + \mathsf{M}(\kappa_{opt})$. Substituting this in (5) and (6) gives

$$\mathsf{R}(\kappa_{opt}) \leq \mathsf{R}(\omega_{opt}) + 6 \quad (7)$$

$$\mathsf{M}(\kappa_{opt}) \leq \mathsf{M}(\omega_{opt}) + 6n^2. \quad (8)$$

Hence, it suffices to show that an optimal consensus protocol κ_{opt} uses at most 6 rounds and $6n^2$ messages more on average than an optimal solution ω_{opt} for GKE. We show this by a constructive argument. Specifically, we show how to build from any given GKE protocol ω' a consensus protocol $\kappa_{\omega'}$ that uses an average of 6 communication rounds and $6n^2$ messages more than a single execution of ω'. By the assumption that κ_{opt} is round-optimal, it follows that $\mathsf{R}(\kappa_{opt}) \leq \mathsf{R}(\kappa_{\omega'}) = 6 + \mathsf{R}(\omega')$ and by the assumption that κ_{opt} is message-optimal, it follows $\mathsf{M}(\kappa_{opt}) \leq \mathsf{M}(\kappa_{\omega'}) = 6n^2 + \mathsf{M}(\omega')$, respectively. Because this holds for *any* GKE protocol ω', it also holds for ω_{opt}, which implies (7) and (8).

We derive the protocol $\kappa_{\omega'}$ from the protocol $\kappa_{\pi_{coin}}$ proposed by Cachin et al. [10], which is based on a protocol π_{coin} for a *common coin*; such a common coin protocol provides every server with a random bit that is unpredictable by the adversary. The consensus protocol $\kappa_{\pi_{coin}}$ invokes π_{coin} twice on average, proceeds in expected $6 + 2 \cdot \mathsf{R}(\pi_{coin})$ rounds, and has an expected message complexity of $6n^2 + 2 \cdot \mathsf{M}(\pi_{coin})$.

The main idea behind our protocol $\kappa_{\omega'}$ is to modify $\kappa_{\pi_{coin}}$ as follows. At the beginning, the servers execute GKE protocol ω' once to get a secret key sk. Then, they follow the original protocol $\kappa_{\pi_{coin}}$, except that instead of invoking π_{coin} to get the i'th random bit c_i, they use the i'th bit of sk (if more than k random bits are used, the servers can also use sk as the seed to a pseudorandom generator, and derive the random bits c_i thereof).

It remains to show that if $\kappa_{\pi_{coin}}$ is a t_c-resilient consensus protocol, then the same holds for $\kappa_{\omega'}$. The result will follow from the composition theorem if we can show that the common coins derived from ω' are unpredictable by the adversary (as the same holds for the common coins derived from π_{coin}). But this follows directly from the definition of $\mathsf{TH}^{\mathsf{gke}}_n$, which guarantees that the adversary cannot predict a single bit of the session key, even after seeing all other bits of the key.

4. THE STRONG CORRUPTION MODEL

In this section, we investigate GKE in the presence of a stronger adversary that may also break into the servers and observe their internal state. We first formally describe such attacks and the desired security requirements of GKE in this model. We then prove an upper bound on the number of break-ins that a GKE protocol can tolerate, and describe an implementation of a protocol with optimal resilience.

4.1 Modeling Break-ins and Forward Secure GKE

We model a break-in as a message $(\mathsf{break\_in}, s)$ sent on a connection $\mathsf{cor\_in}_i$, where s represents an arbitrary bit string. If in a real system $Sys_n^{\mathsf{real},\pi}$ for a protocol π, a server M_i^π receives such a message, it computes a variable $state_i^\pi$ and sends $(\mathsf{done}, state_i^\pi)$ on $\mathsf{cor\_out}_i$ to $\mathsf{A}^{\mathsf{real}}$. When this happens we also say the adversary *breaks into* M_i^π. The variable $state_i^\pi$ comprises all internal data that has not explicitly been erased, including the string s received with the $\mathsf{break\_in}$ message. This ensures that if the protocol π is built on a sub-protocol ρ, then the adversary learns the internal state of M_i^π *and* M_i^ρ, as M_i^ρ receives (by the system model) on $\mathsf{cor\_in}_i$ every message $(\mathsf{break\_in}, state_i^\pi)$ that M_i^π outputs on $\mathsf{cor\_out}_i$, and hence, outputs the internal state of M_i^π and M_i^ρ to the adversary.

The desired security requirement of a GKE protocol in this model is called *forward secrecy*, and means that breaking into a server reveals only the session keys being currently computed by this server, but nothing about previously computed session keys. Here, we define a threshold version of forward secrecy, i.e., we require a GKE protocol to be forward-secure only as long as the adversary does not break into more than t_b different servers. Formally, we capture this notion of security in terms of an ideal system $Sys_{n,t_b}^{\mathsf{ideal},\mathsf{fs\_gke}}$ containing the trusted host $\mathsf{TH}^{\mathsf{fs\_gke}}_{n,t_b}$. This trusted host works exactly as $\mathsf{TH}^{\mathsf{gke}}_n$ (cf., Definition 2), except for maintaining an additional set B containing the indices of broken-into servers (initialized to $\emptyset$), and for the following additional transition rule:

Break-in: When the adversary breaks into a server i, $\mathsf{TH}^{\mathsf{fs\_gke}}_{n,t_b}$ adds i to the set B, and chooses for every $ID \in S[i]$ the key $\kappa[ID]$ at random over $\{0,1\}^k$, if this key is not defined yet. It then computes a set $\mathcal{K}$ of keys to be revealed as follows:

$$\mathcal{K} \leftarrow \begin{cases} \bigcup_{ID \in S[i] \cup F[i]} (ID, \kappa[ID]) & \text{if } |\mathcal{B}| > t_b, \\ \bigcup_{ID \in S[i] \setminus F[i]} (ID, \kappa[ID]) & \text{otherwise.} \end{cases}$$

Finally, it outputs $(\mathsf{keys}, \mathcal{K})$ to the adversary.

Note that the adversary is *adaptive* in the sense that she may break into a server at *any* point during a run, and not only at the beginning of the computation. The definition of a secure GKE protocol in this model is as follows.

Definition 6 *We call a protocol π a (t_c, t_b)-resilient group key exchange protocol, if $Sys_n^{\mathsf{real},\pi}$ is as secure as $Sys_{n,t_b}^{\mathsf{ideal},\mathsf{fs\_gke}}$, and if for every run of a configuration of $Sys_n^{\mathsf{real},\pi}$ where at most t_c servers crash, the following holds: If all non-crashed servers start a session ID, then they all finish ID, provided that the adversary delivers all messages among non-crashed servers.*

4.2 An Upper Bound on the Number of Break-ins

Given a GKE protocol that tolerates t_c servers that crash, we now investigate for how many break-ins one can prove this

protocol as secure as $\mathsf{TH}^{\mathsf{fs\text{-}gke}}_{n,t_b}$. For proving that a protocol π is as secure as an ideal service f, one has to show how to construct for every user H and every adversary $\mathsf{A}^{\mathsf{real}}$ against $Sys^{\mathsf{real},\pi}_n$ an adversary $\mathsf{A}^{\mathsf{ideal}}$ against $Sys^{\mathsf{ideal},\mathsf{f}}_n$, such that H cannot distinguish the corresponding ideal configuration from the real one. The difficulty in constructing $\mathsf{A}^{\mathsf{ideal}}$ is to ensure that it communicates with H in the same way as $\mathsf{A}^{\mathsf{real}}$ does. Currently, the only known way how this can be done is by constructing $\mathsf{A}^{\mathsf{ideal}}$ using the technique of *black-box simulation* as in Section 3.2.

We now argue that for $t_c > 0$, no protocol $\bar{\omega}$ can be proven to be a (t_c, t_b)-resilient GKE protocol using black-box simulation if $t_b \geq n - 2t_c$. In other words, we show that $n - 2t_c$ is an upper bound on the number of break-ins that one can tolerate when building forward-secure GKE.

To see this, assume toward a contradiction that there exists a (t_c, t_b)-resilient GKE protocol γ for $t_c > 0$ and $t_b = n - 2t_c$. We show how to build for a particular user H a real adversary $\mathsf{A}^{\mathsf{real}}$ against γ such that for *any* ideal adversary $\mathsf{A}^{\mathsf{ideal}}$ based on black-box access to $\mathsf{A}^{\mathsf{real}}$, the following holds: The joint view of H and $\mathsf{A}^{\mathsf{real}}$ in a run of the real configuration is efficiently distinguishable from the joint view of H and $\mathsf{A}^{\mathsf{real}}$ in a run of the ideal configuration. This contradicts the initial assumption that γ is a (t_c, t_b)-resilient GKE protocol.

We construct $\mathsf{A}^{\mathsf{real}}$ for a user that invokes a single session ID as follows. The adversary $\mathsf{A}^{\mathsf{real}}$ runs γ in two stages: In the first stage, it chooses an arbitrary set M of $n - t_c$ servers, and runs the protocol in an arbitrary way, but ensuring that:

- all messages sent among servers in M are delivered,
- no message is delivered which is either sent by or sent to a server not in M, and
- no server is crashed or broken into.

When every server in M has finished the session ID, the first stage ends. In the second stage, $\mathsf{A}^{\mathsf{real}}$ crashes t_c arbitrary servers in M and breaks into the remaining $n - 2t_c$ servers of M. It then continues to run the protocol in an arbitrary way, but ensuring that no server is crashed or broken into, and that all messages sent among non-crashed servers are delivered. Once every non-crashed server has finished the session ID, the second stage ends, and the adversary halts. Notice that by the assumption that γ is a (t_c, t_b)-resilient GKE protocol, both stages terminate, and moreover, every server that finishes session ID outputs the same key.

Let sk denote this key, let $state_j$ denote the internal state of a server M^{γ}_j at the end of stage one, and let $\mathcal{M}$ denote the set of all messages sent (but not yet delivered) to servers not in M during the first stage. The fact that the servers in M do not interact during the first stage with a server that is not in M implies that the session key sk is efficiently computable from $\mathcal{M}$ and the states $\{state_j\}$ of all servers in M; otherwise, a server not in M could not compute sk and output the key at the and of the second stage.

Suppose now we want to build an adversary $\mathsf{A}^{\mathsf{ideal}}$, i.e., we want to build a simulator that provides a black-box adversary $\mathsf{A}^{\mathsf{real}}$ the (simulated) information seen in a real run of γ, based on the information received by $\mathsf{TH}^{\mathsf{fs\text{-}gke}}_{n,t_b}$. By definition of $\mathsf{TH}^{\mathsf{fs\text{-}gke}}_{n,t_b}$ and $\mathsf{A}^{\mathsf{real}}$, the simulator will not receive any information on the session key sk' that the trusted host outputs to the user. Hence, the information that the simulator feeds to $\mathsf{A}^{\mathsf{real}}$ (comprising $\mathcal{M}$ and $\{state_j\}$) is statistically independent of sk', and therefore defines a key sk such that $\Pr[sk' = sk] = 1/2^k$. But in a real run, the key sk defined by this information is always equal to the key sk' that the user receives. We conclude that the joint views of H and $\mathsf{A}^{\mathsf{real}}$ in a run of an ideal and a real configuration, respectively, are efficiently distinguishable.

4.3 An Implementation with Optimal Resilience

We now describe a (t_c, t_b)-resilient GKE protocol ω^{fs} derived from ω. We first explain why ω is not forward-secure and then describe the necessary modifications to derive ω^{fs}.

In protocol ω, a server i continues to participate in the protocol after outputting the session key sk. Specifically, it still responds to arriving enc_key messages by sending back its encrypted contribution y_i. This is necessary to ensure that every non-crashed server eventually computes the session key. As a result, an adversary that breaks into a server i *after* this server has output sk has access to the server's contribution y_i and may compute the key sk. This contradicts forward secrecy, which requires that the adversary must not learn any information on sk when breaking into t_b servers after they have output the key sk.

To ensure forward secrecy, a server could simply *erase* all local data after outputting the key. However, the resulting protocol would not be live anymore, as the slowest servers might never receive enough contributions from other servers to compute the key. To ensure forward secrecy without compromising liveness, we extend ω as follows. In addition to computing the session key sk, every server i also computes a *share* s_i of sk such that any set of at most t_b shares do not reveal any information on sk, whereas any set of $t_b + 1$ shares allows to efficiently compute sk. Once a server has computed its share and the session key, it outputs the key, and erases all local data except for the share (this ensures forward secrecy). It then continues to send its share to every server that requests it. This ensures that every server eventually receives enough information to compute the session key, and hence, ensures liveness.

To compute the shares of sk, we use polynomial secret sharing as proposed by Shamir [25], where the shares s_i of a secret sk are computed by first choosing a random polynomial $f(\cdot)$ of degree t_b with $f(0) = sk$, and then computing every share as $s_i = f(i)$. This technique requires that every server i derives its share s_i of the session key sk using the same polynomial $f(\cdot)$. This can be ensured by generating $t_b + 1$ session keys $sk_0, \ldots, sk_{t_b}$ in parallel, and then defining the polynomial as $f(x) = sk_0 + \sum_{m=1}^{t_b} sk_m x^m$ (the session key sk is then defined as $sk = sk_0$). Generating $t_b + 1$ session keys in parallel can be done by modifying protocol ω such that a server i does not only choose a single contribution value $y_i \in \{0,1\}^k$, but a contribution vector $\mathbf{y}_i \in (\{0,1\}^k)^{t_b+1}$ and using this vector instead of y_i throughout the protocol (operations such as additions or encryptions are simply applied component-wise). As a result, the servers compute the desired $t_b + 1$ session keys in form of a vector $\mathbf{sk} \in (\{0,1\}^k)^{t_b+1}$. We could also have used a single session key sk in conjunction with a pseudorandom function $\{\phi_i\}_{i \in \{0,1\}^k}$ to derive the coefficients of the polynomial as $sk_m \leftarrow \phi_{sk}(m)$ for $m \in [0, t_b]$, but this could not be proven secure against an adaptive adversary.

For the same reason, one has to adjust ω such that a server i sends the contribution $\mathbf{y}_i$ to another server j in a non-committing way [11]. This can be done as follows. When server j receives server i's public key p_i, it chooses a random vector $\mathbf{r}_{ij}$ over

$(\{0,1\}^k)^{t_b+1}$, encrypts it (component-wise) under p_i to get $\mathbf{c}_{ij}$, and sends $\mathbf{c}_{ij}$ to server i. Upon receiving $\mathbf{c}_{ij}$, server i derives $\mathbf{r}_{ij}$ and sends $\mathbf{z}_{ij} \leftarrow \mathbf{r}_{ij} + \mathbf{y}_i$ to server j. Upon receiving this message, server j computes $\mathbf{y}_i \leftarrow \mathbf{z}_{ij} - \mathbf{r}_{ij}$.

Modifying protocol ω as outlined above yields our desired forward-secure protocol ω^{fs}. It uses one execution of a consensus protocol, and additionally, $3n^2$ messages in 3 communication rounds. A security proof for protocol ω^{fs} can be derived by modifying our proof for protocol ω accordingly.

5. REFERENCES

[1] Y. Amir, Y. Kim, C. Nita-Rotaru, J. Schultz, J. Stanton, and G. Tsudik, "Secure group communication using robust contributory key agreement," in *IEEE Transaction on Parallel and Distributed Systems*, to appear, 2004.

[2] Y. Amir, Y. Kim, C. Nita-Rotaru, J. Schultz, J. Stanton, and G. Tsudik, "Exploring robustness in group key agreement," in *Proc. 21st IEEE International Conference on Distributed Computing Systems*, pp. 399–409, 2001.

[3] G. Ateniese, M. Steiner, and G. Tsudik, "New multiparty authentication services and key agreement protocols," *Journal of Selected Areas in Communications IEEE*, vol. 18, no. 4, pp. 1–13, 2000.

[4] H. Attiya and J. Welch, *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*. McGraw-Hill, 1998.

[5] M. Bellare, D. Pointcheval, and P. Rogaway, "Authenticated key exchange secure against dictionary attacks," in *Advances in Cryptology: Eurocrypt '00*, 2000.

[6] C. Boyd, "On key agreement and conference key agreement," in *Proc. 2nd Australasian Conference on Information Security and Privacy (ACISP)*, 1997.

[7] E. Bresson, O. Chevassut, D. Pointcheval, and J. Quisquater, "Provably authenticated group Diffie-Hellman key exchange," in *Proc. 8th ACM Conference on Computer and Communication Secuirty (CCS)*, 2001.

[8] M. Burmester and Y. Desmedt, "A secure and efficient conference key distribution system," in *Advances in Cryptology: Eurocrypt '94*, 1994.

[9] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup, "Secure and efficient asynchronous broadcast protocols (extended abstract)," in *Advances in Cryptology: Crypto '01*, 2001.

[10] C. Cachin, K. Kursawe, and V. Shoup, "Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography," in *Proc. 19th ACM Symposium on Principles of Distributed Computing (PODC)*, pp. 123–132, 2000.

[11] R. Canetti, U. Feige, O. Goldreich, and M. Naor, "Adaptively secure computation," in *Proc. 28th Symposium on Theory of Computing (STOC)*, pp. 639–648, 1996.

[12] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Proc. 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, 2001.

[13] R. Canetti and T. Rabin, "Fast asynchronous Byzantine agreement with optimal resilience," in *Proc. 25th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 42–51, 1993.

[14] T. D. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems," *Journal of the ACM*, vol. 46, no. 4, pp. 685–722, 1996.

[15] G. Chockler, I. Keidar, and R. Vitenberg, "Group communication specifications: A comprehensive study," *ACM Computing Surveys*, vol. 4, pp. 427–469, December 2001.

[16] O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions," *Journal of the ACM*, vol. 33, pp. 792–807, Oct. 1986.

[17] S. Goldwasser and S. Micali, "Probabilistic encryption," *Journal of Computer and System Sciences*, vol. 28, pp. 270–299, 1984.

[18] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof-systems," *SIAM Journal of Computing*, vol. 18, pp. 186–208, Feb. 1989.

[19] I. Ingemarasson, D. Tang, and C. Wong, "A conference key distribution system," *IEEE Transactions on Information Theory*, vol. 28, no. 5, pp. 714–720, 1982.

[20] M. Just and S. Vaudenay, "Authenticated multi-party key agreement," in *Advances in Cryptology: Asiacrypt '96*, 1996.

[21] J. Katz and M. Yung, "Scalable protocols for authenticated group key exchange," in *Advances in Cryptology: Crypto '03*, 2003.

[22] J. Nielsen, "A threshold pseudorandom function construction and its applicatoins," in *Advances in Cryptology: Crypto '02*, 2002.

[23] B. Pfitzmann and M. Waidner, "A model for asynchronous reactive systems and its application to secure message transmission," in *Proc. 22nd IEEE Symposium on Security & Privacy*, pp. 184–200, 2001.

[24] O. Rodeh, K. P. Birman, and D. Dolev, "A study of group rekeying," Technical Report TR2000-1791, Cornell University Computer Science, March 2000.

[25] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, pp. 612–613, Nov. 1979.

[26] M. Steiner, G. Tsudik, and M. Waidner, "Key agrement in dynamic peer groups," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 8, pp. 769–780, 2000.

[27] M. Steiner, *Secure Group Key Agreement*. PhD thesis, Naturwissenschaftlich- Technische Fakultät der Universität des Saarlandes, Saarbrücken, March 2002.

[28] W. Tzeng, "A practical and secure fault-tolerant conference key agreement protocol," in *Proc. Third International Workshop on Practice and Theory in Public Key Cryptography (PKC)*, 2000.