Foundations of Group Key Management — Framework, Security Model and a Generic Construction

Naga Naresh Karuturi[1] and Ragavendran Gopalakrishnan[1] and Rahul Srinivasan[2] and Pandu Rangan Chandrasekaran[1]

[1] Department of Computer Science and Engineering
Indian Institute of Technology Madras
[2] Department of Computer Science and Engineering
Indian Institute of Technology Bombay

Abstract. Group Key Management (GKM) solves the problem of efficiently establishing and managing dynamic groups. Many of the GKM schemes that have been proposed so far have been broken, as they cite ambiguous arguments, lacking formal proofs. In fact, no concrete framework and security model for GKM exists in literature. This paper addresses this serious problem by providing firm foundations for Group Key Management. We provide a generalized framework for centralized GKM along with a formal security model and strong definitions for the security properties that are demanded by dynamic groups. We also show a generic construction of a centralized GKM scheme from any given multi-receiver ID-based Key Encapsulation Mechanism (mID-KEM). By doing so, we unify two concepts that are significantly different in terms of what they achieve. Our construction is simple and efficient. We prove that the resulting GKM inherits the security of the underlying mID-KEM up to CCA security. We also illustrate our general conversion using the mID-KEM proposed in 2007 by Delerablée.

Keywords: Provable Security, General Framework, Security Model, Group Communication, Multicast Security, Group Key Management, ID-based Cryptography, Generic Conversion

1 Introduction

The growth and commercialization of the Internet offers a large variety of scenarios where group communication using multicast will greatly save bandwidth and sender resources. Immediate examples include news feeds and stock quotes, video transmissions, teleconferencing, software updates, movie on demand and more. (See [6] for a more complete survey on multicast applications.) Secure multicast sessions can be implemented by applying encryption schemes. The messages are protected by encryption using a chosen key, which, in the context of group communication, is known as *Session Key* or *Data Encryption Key* (DEK). Only those who know the DEK can recover the original message. Therefore, the problem of securely sending data to authorized group members reduces to securely sending the DEKs to the authorized group. Furthermore, changes in membership may require that the group key be refreshed. Such a key refreshing procedure prevents a joining (leaving) member from decoding messages exchanged in the past (future), even if he has recorded earlier messages encrypted with the old (new) keys.

However, distributing the group key to valid members is a complex problem. Although refreshing the DEK before the join of a new member is trivial (send a new group key to the group members encrypted with the old group key), performing it after a member leaves is far more complicated. The old key cannot be used to distribute a new one, because the leaving member knows the old key. Therefore, a group key distributor must provide another scalable mechanism to refresh the DEK.

Group Key Establishment — Group Key Establishment (GKE) (which includes techniques of group key exchange and group key agreement) allows $n \geq 2$ principals to agree upon a common secret key. An excellent introduction and survey of GKE is given by Boyd and Mathuria [5]. But GKE stops with initial establishment of keys by the users of the group. Dynamic groups require not only initial key establishment but also auxiliary operations such as member addition and member exclusion.

Group Key Management — Group Key Management (GKM) provides a solution to this problem. As defined by Menezes et al. in [17], key management is the set of techniques and procedures supporting the establishment and maintenance of keying relationships between authorized parties. It plays an important role enforcing access control on the group key (DEK) (and consequently on the group communication). Since the authorized parties here form a group, the schemes which solve this problem are known as group key management schemes in literature. According to [19], group key management can be classified as follows.

- *Centralized Group Key Management* — In these schemes, there is a Key Distribution Center (KDC), also known as Central Authority (CA) who maintains the entire group, performing operations which involve allocating keys to members, communicating the Data Encryption Key (DEK) to the members, etc.
- *Decentralized Group Key Management* — In decentralized group key management schemes, members of a multicast group are split into several smaller subgroups which are managed by different subgroup controllers. This reduces the load on the KDC. Properties associated with decentralized group key management schemes are key independence, keys vs. data, type of communication, etc.
- *Distributed Group Key Management* — The distributed key management approach is characterized by having no group controller. The group key can be generated either in a contributory fashion or by one member. Parameters like the number of rounds, number of messages and computation during setup are used to evaluate the efficiency of such protocols.

Security Properties. Any secure GKM scheme must satisfy certain desired security properties. We briefly discuss each of these properties informally below. Later, we will define them formally.

1. **Perfect Forward Secrecy** — It ensures that when a rekey operation is performed for the group, a member cannot decipher previous messages encrypted with any of the older keys.
2. **Group Forward Secrecy** — It prevents a leaving or expelled group member from continuing to access group communication.
3. **Group Backward Secrecy** — It prevents a new member from decoding messages exchanged before he joined the group.
4. **Collusion Resistance** — It ensures that even if all the members who currently do not belong to the group collude, they will not be able to decipher group messages encrypted with the current key.

Multi-receiver ID-based Key Encapsulation Mechanism (mID-KEM) — A multi-receiver key encapsulation mechanism (mKEM) enables a cryptographic key (which may be used subsequently for other purposes) to be securely sent across to a set of receivers. Smart [22] introduced the notion of mKEM in 2004. It was extended later, in [2, 3], to multi-receiver ID-based Key Encapsulation Mechanism (mID-KEM), i.e. mKEM in the ID-based setting. Later, [11] proposed an mID-KEM that has an efficient trade-off between the ciphertext size and the private key size. Recently, Abdalla et al. [1] proposed an mID-KEM construction where ciphertexts are of constant size, but private keys grow quadratic in the number of receivers. Furukawa [20] and Delerablée [13] independently proposed an mID-KEM scheme which achieves constant size ciphertext at the cost of the public key size growing linearly in the number of receivers.

1.1 Related Work on Centralized Group Key Management

One of the major contributions of this paper is a generic framework and concrete security model for centralized GKM. Here, we discuss the related work done in the area of centralized GKM, and

highlight the major drawbacks of various existing schemes, so as to better emphasize the need for such a formal security model.

The key generation concept used by *Group Key Management Protocol* (GKMP) [15] is a cooperative generation between two protocol entities. There are several key generation algorithms viable for use in GKMP (i.e., RSA, Diffe-Hellman, elliptic curves). All these algorithms use asymmetric key technology to pass information between two entities to create a single cryptographic key. Apart from protocols like GKMP, the centralized group key management schemes can be *hierarchical tree* based and *flat-table* based. We briefly mention a few tree based group key management protocols below (a detailed description of all these protocols can be found in [19]).

- *Logical Key Hierarchy* (LKH) [26] — Here, the KDC is the root of the tree and it maintains a tree of keys. The leaves of the tree are the group members, and each node is associated with a *Key Encryption Key* (KEK). Each group member (leaf) maintains a copy of the KEKs associated with all the nodes that are part of the unique path from itself to the root. If a member joins or leaves, the KDC updates the KEKs of all the nodes that are part of the corresponding root-to-leaf path, preserving group secrecy.
- *One-Way Function Tree* (OFT) [21] — Here, a node's KEK is generated rather than just attributed. The KEKs held by a node's children are blinded using a one-way function and then mixed together using a mixing function, resulting in the KEK held by the node.
- *One-Way Function Chain Tree* [7] — Here, a pseudo random generator is used to generate the new KEKs rather than a one-way function and it is done only during user removal.
- *Hierarchical a-ary Tree with Clustering* [9] — Here, the group with n members is divided into clusters of size m and each cluster is assigned to a unique leaf node, resulting in n/m clusters. All members in a cluster share the same cluster KEK. Every member of a cluster is also assigned a unique key which is shared only with the KDC.

The group rekeying method proposed in [16] uses the Chinese Remainder Theorem (CRT) to construct a secure lock that is used to lock the decryption group key. Because the lock is common among all valid members, the transmission efficiency of the message decryption key is $O(1)$ if the message size is disregarded. However, this method suffers from scalability problems.

Cliques [24] provides a way to distribute group session keys in dynamic groups. However, it doesn't scale well to a large group. Molva et al. [18] proposed a scalable solution for dynamic groups. Nevertheless, the scheme has to modify the structure of intermediate components of the multicast communication such as routers or proxies and it suffers from collusion attacks.

In the flat-table based schemes proposed by Waldvogel et al. [25], a table is used to reduce the number of keys stored at the KDC. When a member leaves, all the keys associated with that member are changed by the KDC. The rekeying method in the scheme by Chang et al. [10] uses the Boolean function minimization to minimize the number of messages needed to rekey the group. However, this method suffers from collusion attacks. There are other schemes based on attribute based encryption, like FT (CP-ABE) by Cheung et al. [12] which provide security against collusion as well.

Drawbacks. In most of the schemes that are cited above, there is no formal security proof presented in a suitable security model. Therefore, most of them base their security claims on informal arguments. Even though [24] presents a somewhat formal proof, it is not clear from the proof as to how each security property is satisfied. Waldvogel et al. [25] argue how their scheme is secure only against certain types of attacks such as denial-of-service, man-in-the-middle, etc. And almost all the tree based schemes lack security against *perfect forward secrecy*. Some of the flat-table based schemes are not secure against collusion attacks.

1.2 Our Contribution

To the best of our knowledge, we are the first to propose a generic framework for centralized Group Key Management (GKM) and more importantly, to present a formal security model, defining each of the security properties (*forward secrecy*, *backward secrecy*, *perfect forward secrecy* and *collusion resistance*) formally. None of the existing GKM schemes have been formally proven secure due to the lack of such a formal security model. Numerous attacks [23] have been mounted on various GKM schemes proposed so far. We construct adversarial games for each of the security properties mentioned above, to provide a framework in which one can formally prove a GKM scheme secure. Next, we construct a generalized conversion from any multi-receiver ID-based Key Encapsulation Mechanism to a full-fledged centralized Group Key Management scheme, which is so simple (yet powerful) that there is no significant overhead while going from mID-KEM to GKM. Thus we show that any efficient mID-KEM is enough to obtain an efficient GKM. Further, we proceed to use formal reduction techniques to establish the security of the GKM scheme, using our own security model. We prove forward secrecy, backward secrecy and collusion resistance of our GKM scheme by reduction to the underlying mID-KEM. For perfect forward secrecy, we build our proof on one-way functions. We also illustrate our generalization by extending the mID-KEM proposed in [13], which achieves constant-size ciphertext for communicating the Data Encryption Key (DEK) to GKM. This is the first GKM scheme to achieve constant-size rekeying message length.

1.3 Organization

The rest of the paper is organized as follows. First, in Section 2, we review basic concepts like one-way functions and bilinear maps, which are necessary for our construction. Next, in Section 3, we give the formal framework for generic Group Key Management, namely the assumptions and algorithms involved in a general GKM scheme. The corresponding formal security model for GKM, which includes the description of oracles, the adversarial games and concrete definitions of security for the required security properties, is presented in Section 4. Next, we quickly present the general framework and formal security model of an mID-KEM in Section 5. Following this, we use our formal framework and its accompanying security model for GKM to describe the construction of a GKM scheme from any given mID-KEM and formally prove its security in Sections 6 and 7 respectively. Finally, in Section 8 we illustrate our construction by converting the efficient mID-KEM proposed recently by Delerablée [13] to the most efficient GKM proposed till date. We conclude with some open problems in Section 9.

2 Preliminaries

In this section, we review important concepts like *one-way functions*, *bilinear maps* and *negligible functions* that are used in the forthcoming sections.

2.1 One-Way Functions

A function $\mathcal{F} : \{0,1\}^* \rightarrow \{0,1\}^*$ is called *one-way* if the following conditions hold.

- **Easy to Compute.** There exists a (deterministic) polynomial time algorithm $\mathcal{A}$ such that on input x, algorithm $\mathcal{A}$ outputs $\mathcal{F}(x)$.
- **Hard to Invert.** Let U_n denote a random variable uniformly distributed over $\{0,1\}^n$. For every probabilistic polynomial time algorithm $\mathcal{A}'$, every polynomial $p(.)$, and all sufficiently large n,

$$Pr\left[A'(f(U_n), 1^n) \in f^{-1}(f(U_N))\right] \leq \frac{1}{p(n)}$$

We denote the advantage of an adversary $\mathcal{B}$ in inverting a one-way function $\mathcal{F}$ as

$$Adv_{\mathcal{F}}^{inv} = Pr\left[\mathcal{F}(\mathcal{B}(\mathcal{F}(x))) = \mathcal{F}(x) | x \leftarrow \{0,1\}^n\right]$$

2.2 Bilinear Maps

We present the necessary facts about bilinear maps and bilinear map groups. Let $\mathbb{G}$ be an additive cyclic group and $\mathbb{G}_1$ be a multiplicative cyclic group, both of prime order p. A *bilinear map* or a *bilinear pairing* is a map $\hat{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_1$ with the following properties.

- **B**ilinearity. For all $P, Q, R \in \mathbb{G}$,
  - $\hat{e}(P + Q, R) = \hat{e}(P, R) \cdot \hat{e}(Q, R)$
  - $\hat{e}(P, Q + R) = \hat{e}(P, Q) \cdot \hat{e}(P, R)$
  - $\hat{e}(aP, bQ) = \hat{e}(P, Q)^{ab}$
- **N**on-Degeneracy. There exist $P, Q \in \mathbb{G}$ such that $\hat{e}(P, Q) \neq I_{\mathbb{G}_1}$, where $I_{\mathbb{G}_1}$ is the identity element of $\mathbb{G}_1$.
- **C**omputability. There exists an efficient algorithm to compute $\hat{e}(P, Q)$ for all $P, Q \in \mathbb{G}$.

Modified Weil pairing [4] and Tate pairing [14] are examples of cryptographic bilinear maps where $\mathbb{G}$ is an elliptic curve group and $\mathbb{G}_1$ is a subgroup of a finite field.

2.3 Negligible Functions

We call a function $\mu : \mathbb{N} \rightarrow \mathbb{R}$ *negligible* if, for every possible polynomial $p(\cdot)$, there exists an N such that for all $n > N$, we have $\mu(n) < \frac{1}{p(n)}$. Negligible functions remain negligible when multiplied by any fixed polynomial.

3 A Formal Framework for Group Key Management

We restrict our discussions in this paper to *centralized group key management* (centralized GKM) schemes. Here, there is an entity known as the *Central Authority* (CA), who maintains a dynamically changing group of members (users) by performing operations that include, but are not restricted to, allocating unique secret keys to members, establishing the common *Data Encryption Key* (DEK) to members, and ensuring and maintaining group secrecy at all times, especially when a member joins or leaves the group. Every group member is uniquely identified with an *identifier*. In the case of ID-based systems for example, this identifier may be the member's identity itself.

At an abstract level, GKM consists of initially establishing a group key and "managing" it throughout the lifetime of the group. By management, we mean activities that the CA carries out in order to preserve the desired security properties of the group. In centralized schemes, key establishment simplifies to secure key distribution, that is, the CA broadcasts a ciphertext which the group members decipher to obtain the key. A standalone cryptographic primitive that achieves this is *multi-receiver Key Encapsulation Mechanism* (mKEM).

It becomes natural, therefore, to think of centralized GKM schemes as being constructed out of mKEMs. Many GKM schemes do not explicitly view it this way. For example, in LKH [26], the KDC first distributes the KEKs which are then used to encrypt the DEK. The underlying mKEM here is a simple symmetric key encryption scheme. The FT (CP-ABE) scheme [12] explicitly uses a public key technique called *ciphertext policy - attribute based encryption* to establish the DEK. Normal multi-receiver encryption schemes also fall into the category of mKEMs; the difference lies in the fact that, in encryption schemes, the key that is *encrypted* is known beforehand and is a necessary input to the encryption algorithm. Whereas, in traditional KEMs, it is impossible to know the key that is *encapsulated* beforehand; the encapsulation algorithm outputs both the ciphertext and the key that would emerge during its decapsulation.

We now describe the algorithms that form the building blocks of a generic basic GKM scheme. The description is largely functional in nature; the implementation details are specific to the underlying mKEM and the GKM scheme using it.

1. **Setup**$(\mathbf{k}, \mathbf{N}, \mathcal{S}_{\mathbf{init}}, \mathcal{E})$
   - **Input.** k is a security parameter, N is the maximum number of group members (the capacity)[3], $\mathcal{S}_{init}$ is the set of identifiers of initial group members and $\mathcal{E}$ is an underlying multi-receiver key encapsulation mechanism, which is described by the following algorithms. Note that our description is that of a most general mKEM, including normal multi-receiver encryption schemes. Depending upon the specific mKEM that is used, some inputs to the algorithms may not actually be necessary.
     (a) $\mathbf{Setup}_{\mathcal{E}}(\mathbf{k}, \mathbf{N})$ — This algorithm takes as input a security parameter k and the maximum number of receivers N and outputs the public system parameters (or public key) as PK, the secret keys SK_i of users with identifiers i, and, if used, a master secret key MSK.
     (b) $\mathbf{Encapsulate}_{\mathcal{E}}(\mathbf{DEK}, \mathbf{PK}, \mathbf{MSK}, \mathcal{S})$ — This algorithm takes as input the key DEK to be encrypted[4], the public key PK, the master secret key MSK (if used) and the set $\mathcal{S}$ of receivers who alone can decrypt and recover DEK (known as authorized, privileged or intended receivers). It returns a ciphertext, more specifically known in our context as a header Hdr, and in the case of a non-trivial mKEM (mKEMs that are not simply encryption schemes), also returns the DEK corresponding to the header.
     (c) $\mathbf{Decapsulate}_{\mathcal{E}}(\mathbf{Hdr}, \mathbf{PK}, \mathbf{SK_i}, \mathcal{S})$ — This algorithm takes as input the ciphertext or header Hdr, the public key PK, the secret key SK_i of one of the authorized decrypting receivers whose identifier is i, and the set $\mathcal{S}$ of authorized receivers[5]. It returns the encrypted key DEK corresponding to the header Hdr.
   - The CA runs $\mathbf{Setup}_{\mathcal{E}}(\mathbf{k}, \mathbf{N}, \mathcal{S}_{\mathbf{init}})$ to obtain $PK^{\mathcal{E}}$, $SK_i^{\mathcal{E}}$ for all users with identifiers i, and $MSK^{\mathcal{E}}$. Using these, the CA generates the public key PK, the secret keys SK_i ($SK_i^{\mathcal{E}}$ must explicitly be part of SK_i as the users would need it for decapsulation) and the master secret key MSK of the GKM scheme.
   - Every member with identifier i in the set $\mathcal{S}_{init}$ of current group members is given through secure channels, his secret key SK_i and the initial *Data Encryption Key* (DEK), which may be chosen randomly from the key space $\mathcal{K}$.

2. **Rekey**$(\mathcal{S}, \mathbf{PK}, \mathbf{MSK}, \mathcal{E})$
   - **Input.** $\mathcal{S}$ is the set of identifiers of the current group members, PK is the public key, MSK is the master secret key, and $\mathcal{E}$ is the underlying mKEM.
   - Every group member first updates his secret key and securely erases the old one. The exact mechanism, for example, whether this updating process involves an input from the CA or is independent of it, would depend on the specific GKM scheme. Failure to securely erase the old key would enable someone who gains control of the group member's hardware to retrieve the old key using hardware forensics. If a group member does not securely erase the previous key, it is considered a violation of the protocol, meaning that he has already been compromised.
   - The CA runs $\mathbf{Encapsulate}_{\mathcal{E}}(\mathbf{DEK}^{\mathcal{E}}, \mathbf{PK}^{\mathcal{E}}, \mathbf{MSK}^{\mathcal{E}}, \mathcal{S})$, at the end of which he has with him, the pair $(Hdr^{\mathcal{E}}, DEK^{\mathcal{E}})$. Using this, he computes the pair (Hdr, DEK) for the group and broadcasts Hdr.

[3] This is an optional input as there may be GKM schemes which can accommodate any number of group members and do not require an upper bound to be specified before *Setup*.

[4] This input will not be required (indeed, it would be impossible to know the key being encrypted beforehand) when the mKEM used is not a normal multi-receiver encryption scheme (where the key would simply be encrypted (just like a message) and sent to the user(s)).

[5] While most existing mKEMs require the specification of this set, there may be some which do not require that $\mathcal{S}$ be specified.

- The group members with identifiers i retrieve $Hdr^{\mathcal{E}}$ from Hdr (using SK_i) and decrypt it using $\mathbf{Decapsulate}_{\mathcal{E}}(\mathbf{Hdr}^{\mathcal{E}}, \mathbf{PK}^{\mathcal{E}}, \mathbf{SK}_{\mathbf{i}}^{\mathcal{E}}, \mathcal{S})$ to obtain $DEK^{\mathcal{E}}$, from which DEK is recovered. Again, the exact mechanism is specific to the GKM scheme.

3. $\mathbf{Join}(\mathbf{i}, \mathcal{S}, \mathbf{PK}, \mathbf{MSK}, \mathcal{E})$
   - **Input.** i is the identifier of the member who wishes to join the group, $\mathcal{S}$ is the set of identifiers of current group members, PK is the public key, MSK is the master secret key, and $\mathcal{E}$ is the underlying mKEM.
   - A member with identifier $i \notin \mathcal{S}$ who wishes to join the group establishes a secure connection with the CA who may perform some checks before authorizing the user to join the group.
   - The CA updates the set $\mathcal{S} \leftarrow \mathcal{S} \cup \{i\}$, and gives SK_i to the joining member through a secure channel.
   - The CA then runs $\mathbf{Rekey}(\mathcal{S}, \mathbf{PK}, \mathbf{MSK}, \mathcal{E})$.
4. $\mathbf{Leave}(\mathcal{L}, \mathcal{S}, \mathbf{PK}, \mathbf{MSK}, \mathcal{E})$
   - **Input.** $\mathcal{L}$ is the set of identifiers of the members who wish to leave the group or are being banned (revoked), $\mathcal{S}$ is the set of identifiers of current group members, PK is the public key, MSK is the master secret key, and $\mathcal{E}$ is the underlying mKEM.
   - The CA updates the set $\mathcal{S} \leftarrow \mathcal{S} - \mathcal{L}$.
   - The CA then runs $\mathbf{Rekey}(\mathcal{S}, \mathbf{PK}, \mathbf{MSK}, \mathcal{E})$.

   Note. Many GKM schemes exist that specify different techniques for **Leave** depending on whether $\mathcal{L}$ is singleton or not.

Note. The CA may choose to perform the **Rekey** operation periodically even if no member joins or leaves the group, in order to maintain the "freshness" of the group and the data encryption key. This measure is necessary to ensure perfect forward secrecy.

4 Security Model for Group Key Management

In this section, we present formally, the security model for GKM. We proceed as follows. First, we describe the notations that are used throughout the rest of this paper. Then, we describe the oracles that are used in the adversarial games, following which we formally describe these games for each of the four security properties that were informally discussed above.

4.1 Notations

We stress that it is vital that the notations that are presented here are understood beyond doubt, as we have used them liberally in the rest of this paper. We use $\mathcal{S}_t$ to denote the set of identifiers of group members at time instant t. We have introduced time as a variable in order to model the dynamics of GKM. Table 4.1 summarizes the notations dealing with time.

t	An arbitrary instant of time
t_{now}	The current time instant (the present time)
$t_{Corrupt}$	The time at which the corrupt query was issued[6]
$t_{Challenge}$	The time for which the challenge ciphertext is to be generated[7]
$t_{Join}(i)$	The time at which the user with identifier i most recently joined the group
$t_{Leave}(i)$	The time at which the user with identifier i most recently left the group
t_{now}^{-}	The time instant just before t_{now}

Table 4.1. Time-Related Notations

[6] There is no ambiguity because, as we shall see, in every adversarial game, the adversary makes at most one corrupt query

[7] In other words, the group parameters used in generating the challenge ciphertext will be those at time $t_{Challenge}$

4.2 Oracles

The adversarial games involve a challenger to present the adversary with an interface consisting of the oracles that model the algorithms of the real scheme. Below, we describe, again only in functional terms, the oracles to be implemented by a challenger of a generic GKM scheme.

1. $\mathcal{O}_{\mathbf{Join}}(\mathbf{i})$ — This oracle simulates the *Join* algorithm of the GKM, to include the member i in the current group.
   - **Input.** i should be the identifier of a member who is not currently part of the group.
   - The oracle aborts if $i \in \mathcal{S}_{t^-_{now}}$.
   - The set of identifiers of current group members is updated as $\mathcal{S}_{t_{now}} \leftarrow \mathcal{S}_{t^-_{now}} \cup \{i\}$.
   - The **Rekey** algorithm is run and the new ciphertext is recorded.

2. $\mathcal{O}_{\mathbf{Leave}}(\mathbf{i})$ [8] — This oracle simulates the *Leave* algorithm of the GKM, to expel the member i from the current group.
   - **Input.** i should be the identifier of a member who is currently part of the group.
   - The oracle aborts if $i \notin \mathcal{S}_{t^-_{now}}$.
   - The set of identifiers of current group members is updated as $\mathcal{S}_{t_{now}} \leftarrow \mathcal{S}_{t^-_{now}} - \{i\}$.
   - The **Rekey** algorithm is run and the new ciphertext is recorded.

3. $\mathcal{O}_{\mathbf{Ciphertext}}(\mathbf{t})$ — This oracle is used to retrieve the broadcasted ciphertext of *Rekey* operations.
   - **Input.** t should be the present time or a time in the past.
   - The oracle aborts if $t > t_{now}$.
   - The ciphertext (header) corresponding to time t is returned. By "corresponding to", we mean the following.
     - If a *Rekey* operation was done at time t, then the ciphertext broadcasted during that *Rekey* operation is returned.
     - Otherwise, the ciphertext broadcasted during the most recent *Rekey* operation done before time t is returned.

4. $\mathcal{O}_{\mathbf{Decrypt}}(\mathbf{Hdr}, \mathbf{t})$ — This oracle is used to retrieve the DEK from its encrypted form.
   - **Input.** Hdr should be a ciphertext and t should be the present time or a time in the past.
   - The oracle aborts if $t > t_{now}$.
   - The set $\mathcal{S}_t$ of group members at time t is recalled and the secret key SK_i corresponding to a user with identifier $i \in \mathcal{S}_t$ at time t is obtained.
   - $Hdr^{\mathcal{E}}$ and $SK_i^{\mathcal{E}}$ are derived from Hdr and SK_i respectively.
   - $\mathbf{Decapsulate}_{\mathcal{E}}(\mathbf{Hdr}^{\mathcal{E}}, \mathbf{PK}^{\mathcal{E}}, \mathbf{SK}_{\mathbf{i}}^{\mathcal{E}}, \mathcal{S}_{\mathbf{t}})$ is run, and the resultant DEK is returned.

5. $\mathcal{O}_{\mathbf{Corrupt}}(\mathbf{i}, \mathbf{type})$ — This oracle simulates the compromise of a member.
   - **Input.** i should be the identifier of a member, and *type* should be one of `fs` (forward security), `bs` (backward security) or `pfs` (perfect forward security), indicating the type of security that is being attacked using this compromised member.
   - The oracle aborts if $type =$ `pfs` and $i \notin \mathcal{S}_{t_{now}}$ because, for *perfect forward secrecy*, the member who is to be corrupted must be part of the group when he is compromised.
   - Depending on whether *type* is `fs`, `bs` or `pfs`, the secret key corresponding to the user with identifier i at time $t_{Leave}(i)$, $t_{Join}(i)$ or t_{now} respectively is returned.

Note. The challenger who runs these oracles must have some mechanism of recording the set of group members, secret keys and ciphertexts as time progresses. The most natural way of doing this is to maintain lists (indexed by time) for each of these variables and keep appending the new values to the respective lists whenever changes occur.

[8] For a set $\mathcal{L}$ of leaving members, this oracle can be called repeatedly on each member in $\mathcal{L}$

4.3 Formal Definitions of Security

Before describing the adversarial games involved, we formally define the four security notions that were informally discussed in Section 1.

Definition 1. *A $(k, N) - GKM$ scheme is forward secure against adaptive chosen ciphertext attacks (secure in the sense of fs-CCA2) if for all polynomials $N(\cdot)$, the advantage $Adv_{GKM}^{fs-CCA2}$ of any probabilistic polynomial time adversary $\mathcal{A}^{fs-GKM}$ in the game $\mathcal{G}_{CCA2}^{fs-GKM}$ against a challenger $\mathcal{C}^{fs-GKM}$ is negligible in the security parameter k.*

Definition 2. *A $(k, N) - GKM$ scheme is backward secure against adaptive chosen ciphertext attacks (secure in the sense of bs-CCA2) if for all polynomials $N(\cdot)$, the advantage $Adv_{GKM}^{bs-CCA2}$ of any probabilistic polynomial time adversary $\mathcal{A}^{bs-GKM}$ in the game $\mathcal{G}_{CCA2}^{bs-GKM}$ against a challenger $\mathcal{C}^{bs-GKM}$ is negligible in the security parameter k.*

Definition 3. *A $(k, N) - GKM$ scheme is perfect forward secure against adaptive chosen ciphertext attacks (secure in the sense of pfs-CCA2) if for all polynomials $N(\cdot)$, the advantage $Adv_{GKM}^{pfs-CCA2}$ of any probabilistic polynomial time adversary $\mathcal{A}^{pfs-GKM}$ in the game $\mathcal{G}_{CCA2}^{pfs-GKM}$ against a challenger $\mathcal{C}^{pfs-GKM}$ is negligible in the security parameter k.*

Definition 4. *A $(k, N) - GKM$ scheme is collusion resistant against adaptive chosen ciphertext attacks (secure in the sense of cr-CCA2) if for all polynomials $N(\cdot)$, the advantage $Adv_{GKM}^{cr-CCA2}$ of any probabilistic polynomial time adversary $\mathcal{A}^{cr-GKM}$ in the game $\mathcal{G}_{CCA2}^{cr-GKM}$ against a challenger $\mathcal{C}^{cr-GKM}$ is negligible in the security parameter k.*

These definitions are not complete because we have neither described the adversarial games nor defined the advantage of an adversary. First, in Game 4.1, we describe formally a generic adversarial CCA2 game $\mathcal{G}_{CCA2}^{GKM}$. Then we define the games $\mathcal{G}_{CCA2}^{fs-GKM}$, $\mathcal{G}_{CCA2}^{bs-GKM}$ and $\mathcal{G}_{CCA2}^{pfs-GKM}$ as special cases of this generic game. Following this, in Game 4.2, we describe formally the game $\mathcal{G}_{CCA2}^{cr-GKM}$ for collusion resistance.

We define the adversarial games that model attacks against *forward secrecy*, *backward secrecy*, *perfect forward secrecy* and *collusion resistance* as follows.

- *Forward Secrecy* — $\mathcal{G}_{\mathbf{CCA2}}^{\mathbf{fs-GKM}} = \mathcal{G}_{\mathbf{CCA2}}^{\mathbf{GKM}}(\mathcal{C}^{\mathbf{fs-GKM}}, \mathcal{A}^{\mathbf{fs-GKM}}, \mathtt{fs})$. In this adversarial game, we allow the adversary to corrupt any member of his choice at any time he wishes (before the challenge phase). Meanwhile, he can also query other oracles to learn about the system. A GKM scheme satisfies forward secrecy, if a member who has left the group cannot decipher any future ciphertexts intended to the group when he is not part of the group. Therefore, we allow the adversary to enter the challenge phase at any time of his choice. In particular, he may choose to make the challenge query at the time he thinks is most convenient for him to win the challenge. Of course, since we are dealing with forward secrecy, when the adversary makes the challenge query, the corrupted member should not be part of the group.
- *Backward Secrecy* — $\mathcal{G}_{\mathbf{CCA2}}^{\mathbf{bs-GKM}} = \mathcal{G}_{\mathbf{CCA2}}^{\mathbf{GKM}}(\mathcal{C}^{\mathbf{bs-GKM}}, \mathcal{A}^{\mathbf{bs-GKM}}, \mathtt{bs})$. In this adversarial game, we allow the adversary to corrupt any member of his choice at any time he wishes (before the challenge phase). Meanwhile, he can also query other oracles to learn about the system. A GKM scheme satisfies backward secrecy, if a member who has joined the group cannot decipher any past ciphertexts intended to the group when he is not part of the group. To model this, during the challenge phase, we allow the adversary to specify any time of his choice (before the corrupted member last joined the group) as the time during which the challenge is to be generated. Of course, since we are dealing with backward secrecy, when the adversary makes the challenge query, the corrupted member should not be part of the group.

Game 4.1 $\mathcal{G}^{\mathbf{GKM}}_{\mathbf{CCA2}}(\mathcal{C}^{\mathbf{GKM}}, \mathcal{A}^{\mathbf{GKM}}, \mathbf{type})$

This generic game is played between a challenger $\mathcal{C}^{GKM}$ and an adversary $\mathcal{A}^{GKM}$. The variable *type* signifies the type of security that the adversary claims he can break, and can take on any of three values `fs`, `bs`, or `pfs`.

Both the challenger and the adversary are given the security parameter k, the maximum number of group members N, and the specification of the underlying mKEM $\mathcal{E}$. The game consists of the following phases which are presented in the order in which they occur. In addition to carrying out these phases, the challenger takes care of simulating the *Rekey* operation periodically (if periodic rekey is carried out in the GKM scheme that is being attacked).

Setup Phase — The challenger runs $\mathbf{Setup}(\mathbf{k}, \mathbf{N}, \mathcal{S}_{\mathbf{init}}, \mathcal{E})$, for any choice of $\mathcal{S}_{init}$ by the adversary. The public key PK is given to the adversary $\mathcal{A}^{GKM}$. A *Rekey* operation is simulated immediately after, and the time-line is started at this instant ($t = 0$).

Query Phase 1 — During this phase, the adversary is given access to the oracles as described below.

- Queries of the form $\mathcal{O}_{\mathbf{Join}}(\mathbf{i})$ and $\mathcal{O}_{\mathbf{Leave}}(\mathbf{i})$. The adversary can use these queries to control the group dynamics, i.e., he can make a member with identifier i join or leave the group using these queries.
- Queries of the form $\mathcal{O}_{\mathbf{Ciphertext}}(\mathbf{t})$. These queries help the adversary to retrieve the Hdr corresponding to the most recent *Rekey*[9] operation performed at or before a past time t.
- Queries of the form $\mathcal{O}_{\mathbf{Decrypt}}(\mathbf{Hdr}, \mathbf{t})$. The adversary can use these queries to learn the DEK corresponding to any Hdr of his choice, as decrypted at any time t in the past. The challenger responds by decrypting Hdr using the secret key SK_u of some user $u \in \mathcal{S}_t$.

Corrupt Phase — The adversary, at any time $t_{Corrupt}$ of his choice, invokes $\mathcal{O}_{\mathbf{Corrupt}}(\mathbf{i_c}, \mathbf{type})$, where i_c is the identifier of a member of the adversary's choice. The only constraint is that if *type* = `pfs`, then the member with identifier i_c must currently be part of the group. The adversary receives, in return, the secret key SK_{i_c} corresponding to time $t_{Leave}(i_c)$, $t_{Join}(i_c)$, or t_{now}, depending whether *type* is `fs, bs` or `pfs` respectively. Note that unlike in the other phases, the *Corrupt* oracle can be invoked only once in this phase.

Query Phase 2 — The description of this phase is identical to that of ***Query Phase 1*** — the adversary is given access to $\mathcal{O}_{\mathbf{Join}}$, $\mathcal{O}_{\mathbf{Leave}}$, $\mathcal{O}_{\mathbf{Ciphertext}}$ and $\mathcal{O}_{\mathbf{Decrypt}}$.

Challenge Phase — The adversary issues one challenge query to the challenger $\mathcal{C}^{GKM}$ specifying the time $t_{Challenge}$, subject to one of the following restrictions depending on the value of *type*.

- If *type* = `fs`, the restrictions are $t_{Challenge} = t_{now}$ and $i_c \notin \mathcal{S}_{t_{Challenge}}$.
- If *type* = `bs`, the restrictions are $t_{Challenge} < t_{Join}(i_c)$ and $i_c \notin \mathcal{S}_{t_{Challenge}}$.
- If *type* = `pfs`, the restrictions are $t_{Challenge} < t_{Corrupt}$ and $i_c \in \mathcal{S}_{t_{Challenge}}$.

The challenger runs $\mathbf{Encapsulate}_{\mathcal{E}}(\mathbf{DEK}^{\mathcal{E}}, \mathbf{PK}^{\mathcal{E}}, \mathbf{MSK}^{\mathcal{E}}, \mathcal{S}_{\mathbf{t_{Challenge}}})$, at the end of which he has the $(Hdr^{\mathcal{E}}, DEK^{\mathcal{E}})$ pair. Using this, he computes (Hdr^*, DEK^*) corresponding to time $t_{Challenge}$, following which he selects a random bit b, sets K_b to DEK^* and K_{1-b} to a random DEK from the key space $\mathcal{K}$ and challenges the adversary with $\langle Hdr^*, K_0, K_1 \rangle$.

Query Phase 3 — The adversary can continue to adaptively issue queries to all the oracles as in earlier query phases, subject to the restriction that $(Hdr^*, t_{Challenge})$ is not given as a query to $\mathcal{O}_{\mathbf{Decrypt}}$.

Guess Phase The adversary outputs a guess b' of b from $\{0, 1\}$ and he wins the game if $b' = b$. The adversary's advantage in winning the game is defined as $Adv^{CCA2}_{GKM} = |Pr[b = b'] - \frac{1}{2}|$

Note. We have provided two ***Query Phases*** before the ***Challenge Phase*** to model a situation in which the Adversary can corrupt a member at a time of his choice before receiving the challenge.

– *Perfect Forward Secrecy* — $\mathcal{G}^{\mathbf{pfs-GKM}}_{\mathbf{CCA2}} = \mathcal{G}^{\mathbf{GKM}}_{\mathbf{CCA2}}(\mathcal{C}^{\mathbf{pfs-GKM}}, \mathcal{A}^{\mathbf{pfs-GKM}}, \mathtt{pfs})$. In this adversarial game, we allow the adversary to corrupt any member of his choice at any time he wishes (before the challenge phase). A constraint that we impose here is that this member should be part of the group when he is being corrupted. This is because perfect forward secrecy deals with the situation when a member is compromised when he is part of the group. Meanwhile, he can also query other oracles to learn about the system. The compromised group member should not be able to decipher any past ciphertexts. So, we require that the time $t_{Challenge}$ at which the adversary wants the challenge to be generated occurs before the member was corrupted. Another constraint is that the corrupted member should be part of the group during $t_{Challenge}$. Otherwise, it would model backward secrecy.

– *Collusion Resistance* — $\mathcal{G}^{\mathbf{cr-GKM}}_{\mathbf{CCA2}}$. This game is described in Game 4.2. Collusion resistance means that at any point in time, even if all the members who are currently not part of the group collude, they will not be able to decipher the present ciphertext. To model this, in this adversarial game, during the challenge phase, we give the secret keys[10] of all the users who are currently not part of the group to the adversary.

Game 4.2 $\mathcal{G}^{\mathbf{cr-GKM}}_{\mathbf{CCA2}}$

This game is played between the challenger $\mathcal{C}^{cr-GKM}$ and the adversary $\mathcal{A}^{cr-GKM}$. Both the challenger and the adversary are given the security parameter k, the maximum number of group members N, and the specification of the underlying mKEM $\mathcal{E}$. The game consists of the following phases which are presented in the order in which they occur. In addition to carrying out these phases, the challenger takes care of simulating the *Rekey* operation periodically (if periodic rekey is carried out in the GKM scheme that is being attacked).

Setup Phase — Same as in $\mathcal{G}^{\mathbf{GKM}}_{\mathbf{CCA2}}(\mathcal{C}^{cr-GKM}, \mathcal{A}^{cr-GKM}, \cdot)$.

Query Phase 1 — Same as in $\mathcal{G}^{\mathbf{GKM}}_{\mathbf{CCA2}}(\mathcal{C}^{cr-GKM}, \mathcal{A}^{cr-GKM}, \cdot)$.

Challenge Phase — The adversary issues one challenge query to the challenger $\mathcal{C}^{cr-GKM}$ at any time instant $t_{Challenge}$. First, the adversary is given the secret keys SK_i corresponding to time $t_{Leave}(i)$ of all the group members with identifiers $i \notin \mathcal{S}_{t_{Challenge}}$. The challenger obtains the $(Hdr^{\mathcal{E}}, DEK^{\mathcal{E}})$ pair by running $\mathbf{Encapsulate}_{\mathcal{E}}(\mathbf{DEK}^{\mathcal{E}}, \mathbf{PK}^{\mathcal{E}}, \mathbf{MSK}^{\mathcal{E}}, \mathcal{S}_{\mathbf{t_{Challenge}}})$. Using this, he computes (Hdr^*, DEK^*) corresponding to time $t_{Challenge}$, following which he selects a random bit b, sets K_b to DEK^* and K_{1-b} to a random DEK from the key space $\mathcal{K}$ and challenges the adversary with $\langle Hdr^*, K_0, K_1 \rangle$.

Query Phase 2 — The adversary can continue to adaptively issue queries to all the oracles as in earlier query phases, subject to the restriction that $(Hdr^*, t_{Challenge})$ is not given as a query to $\mathcal{O}_{\mathbf{Decrypt}}$.

Guess Phase The adversary outputs a guess b' of b from $\{0, 1\}$ and he wins the game if $b' = b$. The adversary's advantage in winning the game is defined as $Adv^{cr-CCA2}_{GKM} = |Pr[b = b'] - \frac{1}{2}|$

Other Security Notions. We have defined only adaptive CCA2 security for GKM. Now, without going into detailed definitions for other security definitions, which would result in considerable repetition, we explain the intuition behind them. We consider adaptive CCA and adaptive CPA security as well as static versions of these security notions.

– *Adaptive CCA Security* — The adversarial game $\mathcal{G}^{(\cdot)-GKM}_{CCA}$ for adaptive CCA security is the same as the game $\mathcal{G}^{(\cdot)-GKM}_{CCA2}$, except that in the *Query* phase that follows the *Challenge* phase, the adversary is denied access to the *Decryption* oracle altogether.
– *Adaptive CPA Security* — The adversarial game $\mathcal{G}^{(\cdot)-GKM}_{CPA}$ for adaptive CPA security is the same as the game $\mathcal{G}^{(\cdot)-GKM}_{CCA}$, except that in all the *Query* phases, the adversary is denied access to the *Decryption* oracle.

[10] Since secret keys are time dependent, we give the adversary the secret keys of the members corresponding to the time when they last left the group.

– *Static Security* — The adversarial games $\mathcal{G}_{sCCA2}^{(\cdot)-GKM}$, $\mathcal{G}_{sCCA}^{(\cdot)-GKM}$ and $\mathcal{G}_{sCPA}^{(\cdot)-GKM}$ for static security are the same as the respective games for adaptive security, except that the adversary must submit to the challenger, the following in the beginning of the *Setup* phase.
  - In the case of the games $\mathcal{G}_{(\cdot)}^{GKM}$, the identifier i_c of the user he will corrupt during the *Corrupt* phase.
  - In the case of the games $\mathcal{G}_{(\cdot)}^{cr-GKM}$, the set $\mathcal{S}_{t_{Challenge}}$.

5 Multi-receiver ID-based Key Encapsulation Mechanism (mID-KEM)

In this section, we quickly review the basic framework of an mID-KEM and the formal security model for the same. In the forthcoming sections, we shall be using these as black-boxes while taking a general mID-KEM to a GKM scheme and proving its security.

5.1 General Framework of an mID-KEM

We describe the framework of a non-trivial mID-KEM here. By non-trivial, we mean that we do not consider normal encryption schemes (which may trivially be used to encrypt keys just like messages) as KEMs for the purposes of our discussion. An mID-KEM consists of a Private Key Generator (PKG), who generates, using a master secret key MSK, the private keys SK_{ID_i} of group members with identities ID_i and transmits these keys to them through secure channels. The sender, uses the public key PK and identities of the intended or privileged receivers to generate a ciphertext or header, which can be decrypted only by the privileged receivers to obtain a key. More formally, a multi-receiver ID-based Key Encapsulation Mechanism (mID-KEM) with security parameter k and maximum size N of the set of privileged members, consists of the following four algorithms[11].

Setup$(\mathbf{k}, \mathbf{N})$ — This algorithm takes as input a security parameter k and the maximum size of the set of authorized receivers N, and outputs a master secret key MSK and a public key PK. The PKG is given MSK, and PK is made public.

Extract$(\mathbf{MSK}, \mathbf{ID_i}, \mathbf{PK})$ — This algorithm takes as input the master secret key MSK, a user identity ID_i, and the public key PK, and outputs the private key SK_{ID_i} of the user, which is securely transported to the user.

Encapsulate$(\mathcal{S}, \mathbf{PK})$ — This algorithm takes as input a set of identities of privileged (intended) receivers $\mathcal{S} = \{ID_1, ID_2, \ldots, ID_t\}$, with $t \leq N$ and the public key PK, and outputs a pair (Hdr, DEK). Hdr is called the header and $DEK \in \mathcal{K}$, where $\mathcal{K}$ is the key space.

Decapsulate$(\mathcal{S}, \mathbf{ID_i}, \mathbf{SK_{ID_i}}, \mathbf{Hdr}, \mathbf{PK})$. Takes as input the set $\mathcal{S}$ of identities of the intended receivers, the identity ID_i of one of the intended receivers, and the corresponding private key SK_{ID_i}, a header Hdr, and the public key PK. If $ID_i \in \mathcal{S}$, the algorithm outputs the key K.

5.2 Security Model for mID-KEM

The adversarial game involves a challenger to present the adversary with an interface consisting of oracles that model the algorithms of the real scheme. Below, we describe in functional terms, the oracles to be implemented by a challenger of a generic mID-KEM.

1. $\mathcal{O}_{\mathbf{Extract}}(\mathbf{ID_i})$ — Here, ID_i is the identity of a user. The oracle returns the secret key SK_{ID_i} of the user by using the *Extract* algorithm.
2. $\mathcal{O}_{\mathbf{Decapsulate}}(\mathbf{ID_i}, \mathcal{S}, \mathbf{Hdr})$ — Here, ID_i is the identity of an intended user, $\mathcal{S}$ is the set of identities of the intended (privileged) users, and Hdr is a header to be decrypted. The oracle returns the DEK corresponding to the Hdr by using the *Decapsulate* algorithm.

[11] Our description of an mID-KEM does fall into the generic framework of the underlying mKEM discussed in Section 3; the only difference is that the *Setup* algorithm is split here into two algorithms *Setup* and *Extract*

We define CCA2 security for mID-KEM using the adversarial game $\mathcal{G}^{\mathbf{mID-KEM}}_{\mathbf{CCA2}}$ that is described in Game 5.1.

Definition 5. *A $(k, N) - mID - KEM$ is CCA2 secure against adaptive chosen ciphertext attacks if for all polynomials $N(\cdot)$, the advantage $Adv^{CCA2}_{mID-KEM}$ of any probabilistic polynomial time adversary $\mathcal{A}^{mID-KEM}$ in the game $\mathcal{G}^{mID-KEM}_{CCA2}$ against a challenger $\mathcal{C}^{mID-KEM}$ is negligible in the security parameter k.*

Game 5.1 $\mathcal{G}^{mID-KEM}_{CCA2}$

This game is played between the challenger $\mathcal{C}^{mID-KEM}$ and the adversary $\mathcal{A}^{mID-KEM}$. Both the challenger and the adversary are given the security parameter k and the maximum number of receivers N. The game consists of the following phases that are presented in the order in which they occur.

Setup Phase — The challenger runs $\mathbf{Setup}(\mathbf{k}, \mathbf{N})$ and the public key PK is given to the adversary $\mathcal{A}^{mID-KEM}$.

Query Phase 1 — During this phase the adversary is given access to the oracles as described below.

- Queries of the form $\mathcal{O}_{\mathbf{Extract}}(\mathbf{ID_i})$ — The adversary can use this query to learn the secret keys of any of the members of his choice.
- Queries of the form $\mathcal{O}_{\mathbf{Decapsulate}}(\mathbf{ID_i}, \mathcal{S}, \mathbf{Hdr})$ — The adversary can use this query to learn the DEK corresponding to any Hdr meant for any subset of privileged users.

Challenge Phase — During this phase the adversary issues one challenge query to the challenger, submitting a set $\mathcal{S}^*$ of identities of users of the adversary's choice. The only restriction is that $\mathcal{S}^*$ should not contain an identity of a user whose secret key was queried earlier by the adversary. The challenger then uses the *Encapsulate* algorithm with $\mathcal{S}^*$ as input to obtain a (Hdr^*, DEK^*) pair. He then chooses a bit $b \in \{0, 1\}$ at random and sets K_b to DEK^* and K_{1-b} to a random element from the key space $\mathcal{K}$. He then challenges the adversary with $\langle Hdr^*, K_0, K_1 \rangle$.

Query Phase 2 — During this phase the adversary can continue to query the oracles as before, subject to the following restrictions.

- He should not query the *Extract* oracle for the secret key of any member whose identity belongs to $\mathcal{S}^*$.
- He should not query the *Decapsulate* oracle with $(ID_i, \mathcal{S}^*, Hdr^*)$, for any $ID_i \in \mathcal{S}^*$.

Guess Phase — During this phase, the adversary outputs a guess b' of b from $\{0, 1\}$ and he wins the game if $b' = b$. The adversary's advantage in winning the game is defined as $Adv^{CCA2}_{mID-KEM} = |Pr[b' = b] - \frac{1}{2}|$.

Other Security Notions. We have defined only adaptive CCA2 security for mID-KEM. Now, without going into detailed definitions for other security definitions, which would result in considerable repetition, we explain the intuition behind them. We consider adaptive CCA and adaptive CPA security as well as static versions of these security notions.

- *Adaptive CCA Security* — The adversarial game $\mathcal{G}^{mID-KEM}_{CCA}$ for adaptive CCA security is the same as the game $\mathcal{G}^{mID-KEM}_{CCA2}$, except that in the *Query* phase that follows the *Challenge* phase, the adversary is denied access to the *Decryption* oracle altogether.
- *Adaptive CPA Security* — The adversarial game $\mathcal{G}^{mID-KEM}_{CPA}$ for adaptive CPA security is the same as the game $\mathcal{G}^{mID-KEM}_{CCA}$, except that in all the *Query* phases, the adversary is denied access to the *Decryption* oracle.
- *Static Security* — The adversarial games $\mathcal{G}^{mID-KEM}_{sCCA2}$, $\mathcal{G}^{mID-KEM}_{sCCA}$ and $\mathcal{G}^{mID-KEM}_{sCPA}$ for static security are the same as the respective games for adaptive security, except that the adversary must submit, in the beginning of the *Setup* phase, to the challenger, the set $\mathcal{S}^*$ of identities of users he wishes to be challenged upon.[12]

[12] Consequently, in *Query Phase 1* of $\mathcal{G}^{\mathbf{mID-KEM}}$, the adversary should not query the *Extract* oracle for any identities that are present in $\mathcal{S}^*$.

6 A Generic Conversion to Centralized GKM from mID-KEM

Let $m\mathcal{ID} - KEM$ be the underlying mID-KEM and let $\mathcal{GKM}$ be the centralized GKM scheme that is to be constructed using $m\mathcal{ID} - KEM$. Before we formally describe the constituent algorithms of $\mathcal{GKM}$ as per our construction, we state informally what it does and the intuition behind it.

Consider the following trivial (and hypothetical) construction of $\mathcal{GKM}$. For *Setup*, run the *Setup* algorithm of $m\mathcal{ID} - KEM$, make the public key public, run the *Extract* algorithm of $m\mathcal{ID} - KEM$ for all the group members, and securely transport their secret keys and the initial DEK to them. For *Rekey*, simply execute the *Encapsulate* algorithm of $m\mathcal{ID} - KEM$ and broadcast the new header to the members, who can retrieve the new DEK by running the *Decapsulate* algorithm. For *Join* and *Leave*, just update the set of identities of the current group members accordingly and do a *Rekey* operation. It is not difficult to see that this $\mathcal{GKM}$ will be forward secure, backward secure and collusion resistant if $m\mathcal{ID} - KEM$ is provably secure. But it is not *perfect forward secure* because, a header generated now can be decrypted by the group member (who was part of the group when the ciphertext was generated) at any point in the future. This enables a group member to decrypt past headers and recover past DEKs. We circumvent this problem by introducing time-dependent secret keys for group members, so that a group member cannot use his current secret key to decrypt a header that was generated in the past.

Informally, all that our construction does is to introduce an additional time-varying secret key component g that is common to all group members, with which the header of $m\mathcal{ID} - KEM$ is XORed before being broadcasted to the group. The group members first recover the header because they know the secret g, and then decrypt it to recover the DEK. Both the CA and the members update this secret g during every *Rekey* operation by using a one-way function, the old value of g, and a randomness parameter that is broadcasted by the CA. Since we are using a one-way function to update the secret keys, a group member cannot derive a past secret key from his present secret key. (If he manages to do that, then he can decrypt past headers.) Of course, the group member can store his past secret keys, but we prohibit this in our construction, considering it to be a violation of the protocol.

Formally, $\mathcal{GKM}$ consists of the following algorithms, all of which are run by the CA, who plays the role of the PKG of $m\mathcal{ID} - KEM$ as well.

Setup$(\mathbf{k}, \mathbf{N}, \mathcal{S}_{\mathbf{init}}, \mathbf{m\mathcal{I}D - KEM})$

- **Input.** Take as input the security parameter k, the maximum number of group members N, the set $\mathcal{S}_{init}$ of the identities of initial group members, and $m\mathcal{ID} - KEM$, the underlying multi-receiver key encapsulation mechanism.
- Choose a one-way function $\mathcal{F} : \mathbb{Z}_p^* \to \mathbb{Z}_p^*$, and a random seed $g \in \mathbb{Z}_p^*$, where p is a large prime such that $|p| = k$.
- Run $Setup_{m\mathcal{ID}-KEM}(k, N)$ to obtain $PK^{m\mathcal{ID}-KEM}$ and $MSK^{m\mathcal{ID}-KEM}$. Construct the public key $PK = \langle PK^{m\mathcal{ID}-KEM}, \mathcal{F}, m\mathcal{ID} - KEM\rangle$ and make it public.
- Set $MSK = \langle MSK^{m\mathcal{ID}-KEM}, g\rangle$.
- Choose a data encryption key DEK at random from the key space $\mathcal{K}$.
- Run $Extract_{m\mathcal{ID}-KEM}(ID_i)$ for each identity $ID_i \in \mathcal{S}_{init}$ to obtain the secret keys of all the members $SK_{ID_i}^{m\mathcal{ID}-KEM}$. Compute $SK_{ID_i} = (SK_{ID_i}^{m\mathcal{ID}-KEM}, g)$ for all $ID_i \in \mathcal{S}_{init}$ and securely send these keys to the corresponding members. Also send the initial DEK securely to these members.

Note. The second component of the secret key SK_{ID_i} is a $\mathbb{Z}_p^*$-element and is common to all the group members. We refer to this component of the key as the *dynamic key*. It is "dynamic" because, as we shall see, it is updated regularly during every *Rekey* operation.

Rekey$(\mathcal{S}, \mathbf{PK})$

- **Input.** Take as input the set $\mathcal{S}$ of the identities of current group members, and the public key PK.
- Select $r \in_R \mathbb{Z}_p^*$ and update the *dynamic key* by using the one-way function $\mathcal{F}$ as $g_{t_{now}} \leftarrow r \cdot \mathcal{F}(g_{t_{now}^-})$.
- Run $Encapsulate^{m\mathcal{ID}-KEM}(\mathcal{S}, PK^{m\mathcal{ID}-KEM})$ to obtain a $(Hdr_{m\mathcal{ID}-KEM}, DEK)$ pair.
- Construct $Hdr_{\mathcal{GKM}} = Hdr_{m\mathcal{ID}-KEM} \oplus (g_{t_{now}})$ [13] and broadcast $\langle Hdr_{\mathcal{GKM}}, r \rangle$ to the group.
- Every group member also updates the second component of his secret key (the dynamic key) as $g_{t_{now}} \leftarrow r \cdot \mathcal{F}(g_{t_{now}^-})$ and securely erases the copy of $g_{t_{now}^-}$ values.
- Every group member with identity ID_i will retrieve $Hdr_{m\mathcal{ID}-KEM} = Hdr_{\mathcal{GKM}} \oplus (g_{t_{now}})$ and run $Decapsulate^{m\mathcal{ID}-KEM}(\mathcal{S}, ID_i, SK_{ID_i}^{m\mathcal{ID}-KEM}, Hdr_{m\mathcal{ID}-KEM}, PK^{m\mathcal{ID}-KEM})$ to obtain DEK.

Note. The CA keeps running the *Rekey* algorithm periodically even though the group may remain static without any *Join* or *Leave* operations.

Join$(\mathbf{ID_i}, \mathcal{S}, \mathbf{PK})$

- **Input.** Take as input the identity ID_i of a member who wishes to join the group, the set $\mathcal{S}$ of identities of current group members, and the public key PK.
- The joining member establishes a secure connection with the CA, who may perform some checks before authorizing the member to join the group. If authorized, run Run $Extract_{m\mathcal{ID}-KEM}(ID_i)$ to obtain the secret key $SK_{ID_i}^{m\mathcal{ID}-KEM}$ of the member.
- Compute $SK_{ID_i} = (SK_{ID_i}^{m\mathcal{ID}-KEM}, g)$ and securely send it to the joining member.
- Update the set of identities of current group members as $\mathcal{S} \leftarrow \mathcal{S} \cup \{ID_i\}$.
- Run **Rekey**$(\mathcal{S}, \mathbf{PK})$.

Leave$(\mathcal{L}, \mathcal{S}, \mathbf{PK})$

- **Input.** Take as input the set $\mathcal{L}$ of identities of members who wish to leave the group or are revoked, the set $\mathcal{S}$ of identities of current group members, and the public key PK.
- Update the set of identities of current group members as $\mathcal{S} \leftarrow \mathcal{S} - \mathcal{L}$.
- Run **Rekey**$(\mathcal{S}, \mathbf{PK})$.

7 Formal Security Proof for $\mathcal{GKM}$

We now prove that $\mathcal{GKM}$ is secure against *Chosen Ciphertext Attack* (CCA) with respect to all the four security properties by assuming the CCA security of the underlying mID-KEM and the hardness of inverting one-way functions. For proofs which involve the reduction of an adversary of $m\mathcal{ID} - KEM$ to an adversary of $\mathcal{GKM}$, we will be running the following two adversarial games in parallel.

- $\mathcal{G}_{CCA}^{m\mathcal{ID}-KEM}$ — The CCA game corresponding to $m\mathcal{ID} - KEM$. The challenger for this game is denoted by $\mathcal{C}^{m\mathcal{ID}-KEM}$ and the adversary for this game is denoted by $\mathcal{A}^{m\mathcal{ID}-KEM}$.
- $\mathcal{G}_{CCA}^{(\cdot)-\mathcal{GKM}}$ — The $(\cdot)$-CCA game corresponding to $\mathcal{GKM}$. Here, $(\cdot)$ can refer to fs, bs, pfs or cr depending on the security property that is being proved. The challenger and adversary for this game are denoted by $\mathcal{C}^{(\cdot)-\mathcal{GKM}}$ and $\mathcal{A}^{(\cdot)-\mathcal{GKM}}$ respectively.

[13] The XOR operation is done bitwise. $g_{t_{now}}$ is represented as bits and is padded with additional zeroes if necessary.

For proofs which involve the reduction of the problem of inverting a given one-way function to the problem of breaking the security of $\mathcal{GKM}$, we will just run the game $\mathcal{G}_{CCA}^{(\cdot)-\mathcal{GKM}}$.

Before presenting the formal proofs, we give a short informal overview of the two proof techniques that we employ.

- *Proofs for Forward Secrecy, Backward Secrecy and Collusion Resistance* — For these properties, we shall be reducing $\mathcal{A}^{m\mathcal{ID}-KEM}$ to $\mathcal{A}^{(\cdot)-\mathcal{GKM}}$. That is, we assume the existence of an adversary $\mathcal{A}^{(\cdot)-\mathcal{GKM}}$ who can break a particular security property of $\mathcal{GKM}$ and use him to construct the adversary $\mathcal{A}^{m\mathcal{ID}-KEM}$ who can break the security of $m\mathcal{ID}-KEM$. For this purpose, we let $\mathcal{A}^{m\mathcal{ID}-KEM}$ take on the role of $\mathcal{C}^{(\cdot)-\mathcal{GKM}}$ and interact with $\mathcal{A}^{(\cdot)-\mathcal{GKM}}$ on one side through the game $\mathcal{G}_{CCA}^{(\cdot)-\mathcal{GKM}}$ and simultaneously interact with $\mathcal{C}^{m\mathcal{ID}-KEM}$ through the game $\mathcal{G}_{CCA}^{m\mathcal{ID}-KEM}$. Thus, the task of $\mathcal{A}^{m\mathcal{ID}-KEM}$ is to use its interaction with $\mathcal{A}^{(\cdot)-\mathcal{GKM}}$ to try and win against $\mathcal{C}^{m\mathcal{ID}-KEM}$.
- *Proof for Perfect Forward Secrecy* — For this property, we shall be reducing the problem of inverting a one-way function $\mathcal{F}$ to the problem of breaking perfect forward secrecy of $\mathcal{GKM}$. This reduction is weak in the sense that we do not give an exact algorithm for inverting a given one-way function, but merely show the existence of such an algorithm. This is done by acting as the challenger $\mathcal{C}^{pfs-\mathcal{GKM}}$ of the adversary $\mathcal{A}^{pfs-\mathcal{GKM}}$, and interacting with him through the game $\mathcal{G}_{CCA}^{pfs-\mathcal{GKM}}$. Thus, the task of $\mathcal{C}^{pfs-\mathcal{GKM}}$ is to force $\mathcal{A}^{pfs-\mathcal{GKM}}$ to invert the one-way function $\mathcal{F}$, if at all the adversary is to win $\mathcal{G}_{CCA}^{pfs-\mathcal{GKM}}$.

We now describe the working of $\mathcal{C}^{(\cdot)-\mathcal{GKM}}$, who is an important entity in all our proofs.[14] He maintains five lists $\mathcal{L}_c$, $\mathcal{L}_s$, $\mathcal{L}_g$, $\mathcal{L}_j$ and $\mathcal{L}_\ell$ as described below.

- $\mathcal{L}_c$ contains entries of the form $\langle t, Hdr_{\mathcal{GKM}}\rangle$, where $Hdr_{\mathcal{GKM}}$ is the broadcast ciphertext of the *Rekey* operation performed at time t.
- $\mathcal{L}_s$ contains entries of the form $\langle t, \mathcal{S}_t\rangle$, where $\mathcal{S}_t$ is the set of identities of the group members present at time t.
- $\mathcal{L}_g$ contains entries of the form $\langle t, g_t\rangle$, where g_t is the dynamic key at time t.
- $\mathcal{L}_j$ contains entries of the form $\langle ID, t_{Join}(ID)\rangle$, where $t_{Join}(ID)$ is the most recent time at which the member with identity ID joined the group. For every ID, there will be a unique entry in this list.
- $\mathcal{L}_\ell$ contains entries of the form $\langle ID, t_{Leave}(ID)\rangle$, where $t_{Leave}(ID)$ is the most recent time at which the member with identity ID left the group. For every ID, there will be a unique entry in this list.

$\mathcal{C}^{(\cdot)-\mathcal{GKM}}$, acting as the challenger for $\mathcal{A}^{(\cdot)-\mathcal{GKM}}$, must provide access to all the oracles involved in $\mathcal{G}_{CCA}^{(\cdot)-\mathcal{GKM}}$. In the three proofs where he is also an adversary for $m\mathcal{ID}-KEM$, he has access to the oracles provided by $\mathcal{C}^{m\mathcal{ID}-KEM}$, namely $\mathcal{O}_{Extract}^{m\mathcal{ID}-KEM}$ and $\mathcal{O}_{Decapsulate}^{m\mathcal{ID}-KEM}$. In the proof where there is no access to the oracles of $m\mathcal{ID}-KEM$, he can simulate them himself.[15] So, in any case, we will now describe how $\mathcal{C}^{(\cdot)-\mathcal{GKM}}$ can simulate the oracles of $\mathcal{GKM}$ using the two oracles of $m\mathcal{ID}-KEM$ and a little bookkeeping.

- $\mathcal{O}_{\mathbf{Join}}(\mathbf{ID_i})$ — $\mathcal{C}^{(\cdot)-\mathcal{GKM}}$ does the following.
  1. Retrieve the last entry, $(t', \mathcal{S}_{t'})$, from $\mathcal{L}_s$ and check if $ID_i \in \mathcal{S}_{t'}$. If so, then abort. Else, set $\mathcal{S}_{t_{now}} = \mathcal{S}_{t'} \cup \{ID_i\}$ and append $(t_{now}, \mathcal{S}_{t_{now}})$ to $\mathcal{L}_s$.
  2. Retrieve $g_{t_{now}^-}$ from $\mathcal{L}_g$ ($g_{t_{now}^-} = g_{t"}$, where $(t", g_{t"})$ is the last entry in $\mathcal{L}_g$), pick a random $r \in \mathbb{Z}_p^*$, compute $g_{t_{now}} = r \cdot \mathcal{F}(g_{t_{now}^-})$ and append the entry $(t_{now}, g_{t_{now}})$ to $\mathcal{L}_g$.

[14] In the proofs for forward secrecy, backward secrecy and collusion resistance, $\mathcal{C}^{(\cdot)-\mathcal{GKM}}$ is also $\mathcal{A}^{m\mathcal{ID}-KEM}$

[15] He is able to do so because there is no game $\mathcal{G}_{CCA}^{m\mathcal{ID}-KEM}$ and no corresponding challenger to win against.

3. Run $Encapsulate_{m\mathcal{ID}-KEM}(\mathcal{S}_{t_{now}}, PK^{m\mathcal{ID}-KEM})$ to obtain $Hdr_{m\mathcal{ID}-KEM}$ corresponding to a new DEK, compute $Hdr_{\mathcal{G}KM} = \langle Hdr_{m\mathcal{ID}-KEM} \oplus g_{t_{now}}, r\rangle$ and append the entry $(t_{now}, Hdr_{\mathcal{G}KM})$ to $\mathcal{L}_c$.
4. Record the join by appending the entry (ID_i, t_{now}) to $\mathcal{L}_j$. If there already exists an entry corresponding to ID_i, overwrite it.

– $\mathcal{O}_{\mathbf{Leave}}(\mathbf{ID_i})$ — $\mathcal{C}^{(\cdot)-\mathcal{G}KM}$ does the following.

1. Retrieve the last entry, $(t', \mathcal{S}_{t'})$, from $\mathcal{L}_s$ and check if $ID_i \notin \mathcal{S}_{t'}$. If so, then abort. Else, set $\mathcal{S}_{t_{now}} = \mathcal{S}_{t'} - \{ID_i\}$ and append $(t_{now}, \mathcal{S}_{t_{now}})$ to $\mathcal{L}_s$.
2. Retrieve $g_{t^-_{now}}$ from $\mathcal{L}_g$ ($g_{t^-_{now}} = g_{t"}$, where $(t", g_{t"})$ is the last entry in $\mathcal{L}_g$), pick a random $r \in \mathbb{Z}_p^*$, compute $g_{t_{now}} = r \cdot \mathcal{F}(g_{t^-_{now}})$ and append the entry $(t_{now}, g_{t_{now}})$ to $\mathcal{L}_g$.
3. Run $Encapsulate_{m\mathcal{ID}-KEM}(\mathcal{S}_{t_{now}}, PK^{m\mathcal{ID}-KEM})$ to obtain $Hdr_{m\mathcal{ID}-KEM}$ corresponding to a new DEK, compute $Hdr_{\mathcal{G}KM} = \langle Hdr_{m\mathcal{ID}-KEM} \oplus g_{t_{now}}, r\rangle$ and append the entry $(t_{now}, Hdr_{\mathcal{G}KM})$ to $\mathcal{L}_c$.
4. Record the leave by appending the entry (ID_i, t_{now}) to $\mathcal{L}_\ell$. If there already exists an entry corresponding to ID_i, overwrite it.

– $\mathcal{O}_{\mathbf{Ciphertext}}(\mathbf{t})$ — $\mathcal{C}^{(\cdot)-\mathcal{G}KM}$ aborts if $t > t_{now}$. Otherwise, he retrieves, if present, the entry $(t', Hdr_{\mathcal{G}KM})$ from $\mathcal{L}_c$ such that t' is the most recent (numerically largest) time stamp satisfying $t' \leq t$ and returns $Hdr_{\mathcal{G}KM}$. If no such entry is present, he returns $\perp$.

– $\mathcal{O}_{\mathbf{Decrypt}}(\mathbf{Hdr_{\mathcal{G}KM}}, \mathbf{t})$ — $\mathcal{C}^{(\cdot)-\mathcal{G}KM}$ aborts if $t > t_{now}$. Otherwise, he does the following.

1. Retrieve, if present, the entries $(t', \mathcal{S}_{t'})$ from $\mathcal{L}_s$ and $(t', g_{t'})$ from $\mathcal{L}_g$ such that t' is the most recent (numerically largest) time stamp satisfying $t' \leq t$. If no such entries are present, return $\perp$.
2. Generate the header $Hdr_{m\mathcal{ID}-KEM} = Hdr_{\mathcal{G}KM} \oplus g_{t'}$ corresponding to $m\mathcal{ID}-KEM$ and return the result of $\mathcal{O}^{m\mathcal{ID}-KEM}_{Decapsulate}(ID_i, \mathcal{S}_{t'}, Hdr_{m\mathcal{ID}-KEM})$, where ID_i is chosen at random from $\mathcal{S}_{t'}$.

– $\mathcal{O}_{\mathbf{Corrupt}}(\mathbf{ID_i}, \mathbf{type})$ — $\mathcal{C}^{(\cdot)-\mathcal{G}KM}$ does the following.

1. When $type = \texttt{fs}$, retrieve if present, the entries $(ID_i, t_{Leave}(ID_i))$ and $(t_{Leave}(ID_i), g_{t_{Leave}(ID_i)})$ from $\mathcal{L}_\ell$ and $\mathcal{L}_g$ respectively. If no such entries are present, return $\perp$. Else obtain S_{ID_i} by querying $\mathcal{O}^{m\mathcal{ID}-KEM}_{Extract}(ID_i)$ and return $SK_{ID_i} = (SK^{m\mathcal{ID}-KEM}_{ID_i}, g_{t_{Leave}(ID_i)})$.
2. When $type = \texttt{bs}$, retrieve if present, the entries $(ID_i, t_{Join}(ID_i))$ and $(t_{Join}(ID_i), g_{t_{Join}(ID_i)})$ from $\mathcal{L}_j$ and $\mathcal{L}_g$ respectively. If no such entries are present, return $\perp$. Else obtain S_{ID_i} by querying $\mathcal{O}^{m\mathcal{ID}-KEM}_{Extract}(ID_i)$ and return $SK_{ID_i} = (SK^{m\mathcal{ID}-KEM}_{ID_i}, g_{t_{Join}(ID_i)})$.
3. When $type = \texttt{pfs}$, retrieve the last entry (t, g_t) from $\mathcal{L}_g$, query $\mathcal{O}^{m\mathcal{ID}-KEM}_{Extract}(ID_i)$ to obtain S_{ID_i} and return $SK_{ID_i} = (SK^{m\mathcal{ID}-KEM}_{ID_i}, g_t)$.

We now present the four security theorems and their formal proofs.

Theorem 1. *$\mathcal{G}KM$ is fs-CCA secure if $m\mathcal{ID}-KEM$ is at least CCA secure.*

Proof. Here, we describe how the adversary $\mathcal{A}^{m\mathcal{ID}-KEM}$ on one side acts as the challenger $\mathcal{C}^{fs-\mathcal{G}KM}$ who interacts with $\mathcal{A}^{fs-\mathcal{G}KM}$, while simultaneously interacting with $\mathcal{C}^{m\mathcal{ID}-KEM}$ on the other side, trying to win against him. Since the two games are being run in parallel and we describe the events in chronological order, the description below switches between the phases of the two games. To ensure some clarity, we present the description from the point of view of the game $\mathcal{G}^{fs-\mathcal{G}KM}_{CCA}$.

1. *Setup Phase* — The challenger $\mathcal{C}^{m\mathcal{ID}-KEM}$ runs $Setup_{m\mathcal{ID}-KEM}(k, N)$ to obtain $PK^{m\mathcal{ID}-KEM}$, and gives it to $\mathcal{A}^{m\mathcal{ID}-KEM}$, who constructs $PK = \langle PK^{m\mathcal{ID}-KEM}, \mathcal{F}, m\mathcal{ID}-KEM\rangle$ and gives it to $\mathcal{A}^{fs-\mathcal{GKM}}$. He also picks a random seed g from $\mathbb{Z}_p^*$ and sets the master secret key MSK to $\langle MSK_{m\mathcal{ID}-KEM}, g\rangle$.
2. *Query Phase 1* — $\mathcal{A}^{fs-\mathcal{GKM}}$ is allowed to query the oracles $\mathcal{O}_{Join}$, $\mathcal{O}_{Leave}$, $\mathcal{O}_{Ciphertext}$ and $\mathcal{O}_{Decrypt}$.
3. *Corrupt Phase* — $\mathcal{A}^{fs-\mathcal{GKM}}$ chooses ID_{i_c}, an identity which he wants to corrupt and makes the query $\mathcal{O}_{Corrupt}(ID_{i_c}, \texttt{fs})$ at time $t_{Corrupt}$ (which is the choice of $\mathcal{A}^{pfs-\mathcal{GKM}}$).
4. *Query Phase 2* — $\mathcal{A}^{fs-\mathcal{GKM}}$ can query the oracles as in *Query Phase 1*.
5. *Challenge Phase* — $\mathcal{A}^{fs-\mathcal{GKM}}$ issues one challenge query to its challenger $\mathcal{A}^{m\mathcal{ID}-KEM}$ at time $t_{Challenge}$ (which is the choice of $\mathcal{A}^{fs-\mathcal{GKM}}$), subject to the restriction that $ID_{i_c} \notin \mathcal{S}_{t_{Challenge}}$. Now, $\mathcal{A}^{m\mathcal{ID}-KEM}$ does the following before responding with the challenge.
   - Retrieve the set $\mathcal{S}_{t_{Challenge}}$ from the list $\mathcal{L}_s$.
   - Issue a challenge query, specifying the set $\mathcal{S}_{t_{Challenge}}$, to the challenger $\mathcal{C}^{m\mathcal{ID}-KEM}$.
   - Receive the challenge $(Hdr^*_{m\mathcal{ID}-KEM}, K_0, K_1)$.
   - Compute $Hdr^*_{\mathcal{GKM}}$ as $\langle Hdr^*_{m\mathcal{ID}-KEM} \oplus g_{t_{Challenge}}, r_{t_{Challenge}}\rangle$.[16]

   $\mathcal{A}^{m\mathcal{ID}-KEM}$ returns $(Hdr^*_{\mathcal{GKM}}, K_0, K_1)$ as the challenge to $\mathcal{A}^{fs-\mathcal{GKM}}$.
6. *Guess Phase* — $\mathcal{A}^{fs-\mathcal{GKM}}$ outputs a bit $b' \in \{0, 1\}$ as its guess. $\mathcal{A}^{m\mathcal{ID}-KEM}$ passes on b' as its guess to $\mathcal{C}^{m\mathcal{ID}-KEM}$.

It is easy to see that the advantage of $\mathcal{A}^{fs-\mathcal{GKM}}$ in breaking the forward secrecy of $\mathcal{GKM}$ is the same as that of $\mathcal{A}^{m\mathcal{ID}-KEM}$ in breaking the CCA security of $m\mathcal{ID}-KEM$.

$$Adv^{fs-CCA}_{\mathcal{GKM}} = Adv^{CCA}_{m\mathcal{ID}-KEM} = |Pr[b = b'] - \frac{1}{2}|$$

This means that if there exists no adversary $\mathcal{A}^{m\mathcal{ID}-KEM}$ who can break the CCA security of $m\mathcal{ID}-KEM$ with non-negligible advantage, then there cannot be any adversary $\mathcal{A}^{fs-\mathcal{GKM}}$ who can break the forward secrecy of $\mathcal{GKM}$ with non-negligible advantage. □

Theorem 2. *$\mathcal{GKM}$ is bs-CCA secure if $m\mathcal{ID}-KEM$ is at least CCA secure.*

Proof. Here, we describe how the adversary $\mathcal{A}^{m\mathcal{ID}-KEM}$ on one side acts as the challenger $\mathcal{C}^{bs-\mathcal{GKM}}$ who interacts with $\mathcal{A}^{bs-\mathcal{GKM}}$, while simultaneously interacting with $\mathcal{C}^{m\mathcal{ID}-KEM}$ on the other side, trying to win against him. Since the two games are being run in parallel and we describe the events in chronological order, the description below switches between the phases of the two games. To ensure some clarity, we present the description from the point of view of the game $\mathcal{G}^{bs-\mathcal{GKM}}_{CCA}$.

1. *Setup Phase* — The challenger $\mathcal{C}^{m\mathcal{ID}-KEM}$ runs $Setup_{m\mathcal{ID}-KEM}(k, N)$ to obtain $PK^{m\mathcal{ID}-KEM}$, and gives it to $\mathcal{A}^{m\mathcal{ID}-KEM}$, who constructs $PK = \langle PK^{m\mathcal{ID}-KEM}, \mathcal{F}, m\mathcal{ID}-KEM\rangle$ and gives it to $\mathcal{A}^{bs-\mathcal{GKM}}$. He also picks a random seed g from $\mathbb{Z}_p^*$ and sets the master secret key MSK to $\langle MSK_{m\mathcal{ID}-KEM}, g\rangle$.
2. *Query Phase 1* — $\mathcal{A}^{bs-\mathcal{GKM}}$ is allowed to query the oracles $\mathcal{O}_{Join}$, $\mathcal{O}_{Leave}$, $\mathcal{O}_{Ciphertext}$ and $\mathcal{O}_{Decrypt}$.

[16] $g_{t_{Challenge}}$ is retrieved from the list $\mathcal{L}_g$. Since $g_{t_{Challenge}} = r_{t_{Challenge}} \cdot \mathcal{F}(g_{t^-_{Challenge}})$, it can be seen that $r_{t_{Challenge}}$ can be computed using $g_{t_{Challenge}}$ and $g_{t^-_{Challenge}}$, both of which are available in $\mathcal{L}_g$.

3. *Corrupt Phase* — $\mathcal{A}^{bs-\mathcal{GKM}}$ chooses ID_{i_c}, an identity which he wants to corrupt and makes the query $\mathcal{O}_{Corrupt}(ID_{i_c}, \mathtt{bs})$ at time $t_{Corrupt}$ (which is the choice of $\mathcal{A}^{pfs-\mathcal{GKM}}$).
4. *Query Phase 2* — $\mathcal{A}^{bs-\mathcal{GKM}}$ can query the oracles as in *Query Phase 1*.
5. *Challenge Phase* — $\mathcal{A}^{bs-\mathcal{GKM}}$ issues one challenge query to its challenger $\mathcal{A}^{m\mathcal{ID}-KEM}$, specifying a time $t_{Challenge}$ (which is the choice of $\mathcal{A}^{bs-\mathcal{GKM}}$), subject to the restrictions that $ID_{i_c} \notin \mathcal{S}_{t_{Challenge}}$ and $t_{Challenge} \leq t_{Join}(ID_{i_c})$. Now, $\mathcal{A}^{m\mathcal{ID}-KEM}$ does the following before responding with the challenge.
   - Retrieve the set $\mathcal{S}_{t_{Challenge}}$ from the list $\mathcal{L}_s$.
   - Issue a challenge query, specifying the set $\mathcal{S}_{t_{Challenge}}$, to the challenger $\mathcal{C}^{m\mathcal{ID}-KEM}$.
   - Receive the challenge $(Hdr^*_{m\mathcal{ID}-KEM}, K_0, K_1)$.
   - Compute $Hdr^*_{\mathcal{GKM}}$ as $\langle Hdr^*_{m\mathcal{ID}-KEM} \oplus g_{t_{Challenge}}, r_{t_{Challenge}} \rangle$.[17]

   $\mathcal{A}^{m\mathcal{ID}-KEM}$ returns $(Hdr^*_{\mathcal{GKM}}, K_0, K_1)$ as the challenge to $\mathcal{A}^{bs-\mathcal{GKM}}$.
6. *Guess Phase* — $\mathcal{A}^{bs-\mathcal{GKM}}$ outputs a bit $b' \in \{0, 1\}$ as its guess. $\mathcal{A}^{m\mathcal{ID}-KEM}$ passes on b' as its guess to $\mathcal{C}^{m\mathcal{ID}-KEM}$.

It is easy to see that the advantage of $\mathcal{A}^{bs-\mathcal{GKM}}$ in breaking the backward secrecy of $\mathcal{GKM}$ is the same as that of $\mathcal{A}^{m\mathcal{ID}-KEM}$ in breaking the CCA security of $m\mathcal{ID} - KEM$.

$$Adv^{bs-CCA}_{\mathcal{GKM}} = Adv^{CCA}_{m\mathcal{ID}-KEM} = |Pr[b = b'] - \frac{1}{2}|$$

This means that if there exists no adversary $\mathcal{A}^{m\mathcal{ID}-KEM}$ who can break the CCA security of $m\mathcal{ID} - KEM$ with non-negligible advantage, then there cannot be any adversary $\mathcal{A}^{bs-\mathcal{GKM}}$ who can break the backward secrecy of $\mathcal{GKM}$ with non-negligible advantage. □

Theorem 3. *$\mathcal{GKM}$ is pfs-CCA secure if inverting $\mathcal{F}$ is hard.*

Proof. This proof differs somewhat from the other proofs because we are reducing the security of $\mathcal{GKM}$ to the one-wayness of $\mathcal{F}$. Here, we describe how the challenger $\mathcal{C}^{pfs-\mathcal{GKM}}$ interacts with $\mathcal{A}^{pfs-\mathcal{GKM}}$ and forces him to invert the one-way function $\mathcal{F}$ in order for him to win against $\mathcal{C}^{pfs-\mathcal{GKM}}$. The game that is being described is $\mathcal{G}^{pfs-\mathcal{GKM}}_{CCA}$.

1. *Setup Phase* — $\mathcal{C}^{pfs-\mathcal{GKM}}$ runs $Setup_{m\mathcal{ID}-KEM}(k, N)$ to obtain $PK^{m\mathcal{ID}-KEM}$. He constructs $PK = \langle PK^{m\mathcal{ID}-KEM}, \mathcal{F}, m\mathcal{ID} - KEM \rangle$ and gives it to $\mathcal{A}^{pfs-\mathcal{GKM}}$. He also picks a random seed g from $\mathbb{Z}^*_p$ and sets the master secret key MSK to $\langle MSK_{m\mathcal{ID}-KEM}, g \rangle$.
2. *Query Phase 1* — $\mathcal{A}^{pfs-\mathcal{GKM}}$ is allowed to query the oracles $\mathcal{O}_{Join}$, $\mathcal{O}_{Leave}$, $\mathcal{O}_{Ciphertext}$ and $\mathcal{O}_{Decrypt}$.
3. *Corrupt Phase* — $\mathcal{A}^{pfs-\mathcal{GKM}}$ chooses ID_{i_c}, an identity which he wants to corrupt and makes the query $\mathcal{O}_{Corrupt}(ID_{i_c}, \mathtt{bs})$ at time $t_{Corrupt}$ (which is the choice of $\mathcal{A}^{pfs-\mathcal{GKM}}$).
4. *Query Phase 2* — $\mathcal{A}^{pfs-\mathcal{GKM}}$ can query the oracles as in *Query Phase 1*.
5. *Challenge Phase* — $\mathcal{A}^{pfs-\mathcal{GKM}}$ issues one challenge query to $\mathcal{C}^{pfs-\mathcal{GKM}}$, specifying a time $t_{Challenge}$ (which is the choice of $\mathcal{A}^{pfs-\mathcal{GKM}}$), subject to the restrictions that $ID_{i_c} \in \mathcal{S}_{t_{Challenge}}$ and $t_{Join}(ID_{i_c}) < t_{Challenge} < t_{Corrupt}$. Now, $\mathcal{C}^{pfs-\mathcal{GKM}}$ does the following before responding with the challenge.

[17] $g_{t_{Challenge}}$ is retrieved from the list $\mathcal{L}_g$. Since $g_{t_{Challenge}} = r_{t_{Challenge}} \cdot \mathcal{F}(g_{t^-_{Challenge}})$, it can be seen that $r_{t_{Challenge}}$ can be computed using $g_{t_{Challenge}}$ and $g_{t^-_{Challenge}}$, both of which are available in $\mathcal{L}_g$.

- Retrieve the set $\mathcal{S}_{t_{Challenge}}$ from the list $\mathcal{L}_s$.
- Run $Encapsulate^{m\mathcal{ID}-KEM}(\mathcal{S}_{t_{Challenge}}, PK^{m\mathcal{ID}-KEM})$ and obtain a $(Hdr_{m\mathcal{ID}-KEM}, DEK)$ pair.
- Compute $Hdr^*_{\mathcal{G}KM} \leftarrow \langle Hdr_{m\mathcal{ID}-KEM} \oplus g_{t_{Challenge}}, r_{t_{Challenge}} \rangle$.[18]
- Randomly select a bit $b \in \{0,1\}$ and set $K_b = DEK$ and K_{1-b} to a random element from the key space $\mathcal{K}$.

Now, $\mathcal{C}^{pfs-\mathcal{G}KM}$ returns $(Hdr^*_{\mathcal{G}KM}, K_0, K_1)$ as the challenge to $\mathcal{A}^{pfs-\mathcal{G}KM}$.

6. *Guess Phase* — $\mathcal{A}_{pfs-\mathcal{G}KM}$ outputs a bit $b' \in \{0,1\}$ as its guess.

Note that since $g_{t_{Challenge}} = r_{t_{Challenge}} \cdot \mathcal{F}(g_{t^-_{Challenge}})$ and $r_{t_{Challenge}}$ is random in $\mathbb{Z}^*_p$, $g_{t_{Challenge}}$ is also random. Therefore, the challenge $Hdr^*_{\mathcal{G}KM}$ is also random. So, the only way by which the adversary $\mathcal{A}_{pfs-\mathcal{G}KM}$ can get any information about from $Hdr^*_{\mathcal{G}KM}$ about the DEK corresponding to $Hdr_{m\mathcal{ID}-KEM}$ is by obtaining $Hdr_{m\mathcal{ID}-KEM}$ itself. This implies that, if he is able to obtain $Hdr_{m\mathcal{ID}-KEM}$, then he is also able to obtain $g_{t_{Challenge}}$[19] from $g_{t_{Corrupt}}$. Since $t_{Challenge} < t_{Corrupt}$, this shows the ability of the adversary to invert the one-way function $\mathcal{F}$. Hence the advantage of the adversary $\mathcal{A}^{pfs-\mathcal{G}KM}$ is at most his advantage in inverting the one-way function $\mathcal{F}$.

$$Adv^{pfs-CCA}_{\mathcal{G}KM} < Adv^{inv}_{\mathcal{F}}$$

This means that if there exists no algorithm that can invert a one-way function $\mathcal{F}$ with non-negligible advantage, then there cannot be any adversary $\mathcal{A}^{pfs-\mathcal{G}KM}$ who can break the perfect forward secrecy of $\mathcal{G}KM$ with non-negligible advantage. □

Theorem 4. *$\mathcal{G}KM$ is cr-CCA secure if $m\mathcal{ID}-KEM$ is at least CCA secure.*

Proof. Here, we describe how the adversary $\mathcal{A}^{m\mathcal{ID}-KEM}$ on one side acts as the challenger $\mathcal{C}^{cr-\mathcal{G}KM}$ who interacts with $\mathcal{A}^{cr-\mathcal{G}KM}$, while simultaneously interacting with $\mathcal{C}^{m\mathcal{ID}-KEM}$ on the other side, trying to win against him. Since the two games are being run in parallel and we describe the events in chronological order, the description below switches between the phases of the two games. To ensure some clarity, we present the description from the point of view of the game $\mathcal{G}^{cr-\mathcal{G}KM}_{CCA}$.

1. *Setup Phase* — The challenger $\mathcal{C}^{m\mathcal{ID}-KEM}$ runs $Setup_{m\mathcal{ID}-KEM}(k, N)$ to obtain $PK^{m\mathcal{ID}-KEM}$, and gives it to $\mathcal{A}^{m\mathcal{ID}-KEM}$, who constructs $PK = \langle PK^{m\mathcal{ID}-KEM}, \mathcal{F}, m\mathcal{ID}-KEM \rangle$ and gives it to $\mathcal{A}^{cr-\mathcal{G}KM}$. He also picks a random seed g from $\mathbb{Z}^*_p$ and sets the master secret key MSK to $\langle MSK_{m\mathcal{ID}-KEM}, g \rangle$.
2. *Query Phase* — $\mathcal{A}^{cr-\mathcal{G}KM}$ is allowed to query the oracles $\mathcal{O}_{Join}$, $\mathcal{O}_{Leave}$, $\mathcal{O}_{Ciphertext}$ and $\mathcal{O}_{Decrypt}$.
3. *Challenge Phase* — $\mathcal{A}^{cr-\mathcal{G}KM}$ issues one challenge query to its challenger $\mathcal{A}^{m\mathcal{ID}-KEM}$ at time $t_{Challenge}$ (which is the choice of $\mathcal{A}^{cr-\mathcal{G}KM}$). Now, $\mathcal{A}^{m\mathcal{ID}-KEM}$ does the following before responding with the challenge.
   - Retrieve the set $\mathcal{S}_{t_{Challenge}}$ from the list $\mathcal{L}_s$, and $g_{t_{Leave}(ID_i)}$ from the list $\mathcal{L}_g$, for all $ID_i \notin \mathcal{S}_{t_{Challenge}}$.
   - For each identity $ID_i \notin \mathcal{S}_{t_{Challenge}}$, issue the query $\mathcal{O}^{m\mathcal{ID}-KEM}_{Extract}(ID_i)$ to obtain S_{ID_i} and return $SK_{ID_i} = (S_{ID_i}, g_{t_{Leave}(ID_i)})$.
   - Issue a challenge query, specifying the set $\mathcal{S}_{t_{Challenge}}$, to the challenger $\mathcal{C}^{m\mathcal{ID}-KEM}$.

[18] $g_{t_{Challenge}}$ is retrieved from the list $\mathcal{L}_g$. Since $g_{t_{Challenge}} = r_{t_{Challenge}} \cdot \mathcal{F}(g_{t^-_{Challenge}})$, it can be seen that $r_{t_{Challenge}}$ can be computed using $g_{t_{Challenge}}$ and $g_{t^-_{Challenge}}$, both of which are available in $\mathcal{L}_g$.

[19] Obtaining $g_{t_{Challenge}}$ from $Hdr^*_{\mathcal{G}KM}$ and $Hdr_{m\mathcal{ID}-KEM}$ just involves an XOR operation

– Receive the challenge $(Hdr^*_{m\mathcal{I}D-KEM}, K_0, K_1)$.
– Compute $Hdr^*_{\mathcal{G}KM}$ as $\langle Hdr^*_{m\mathcal{I}D-KEM} \oplus g_{t_{Challenge}}, r_{t_{Challenge}} \rangle$.[20]
$\mathcal{A}^{m\mathcal{I}D-KEM}$ returns $(Hdr^*_{\mathcal{G}KM}, K_0, K_1)$ as the challenge to $\mathcal{A}^{cr-\mathcal{G}KM}$.

4. *Guess Phase* — $\mathcal{A}^{cr-\mathcal{G}KM}$ outputs a bit $b' \in \{0,1\}$ as its guess. $\mathcal{A}^{m\mathcal{I}D-KEM}$ passes on b' as its guess to $\mathcal{C}^{m\mathcal{I}D-KEM}$.

It is easy to see that the advantage of $\mathcal{A}^{cr-\mathcal{G}KM}$ in breaking the collusion resistance of $\mathcal{G}KM$ is the same as that of $\mathcal{A}^{m\mathcal{I}D-KEM}$ in breaking the CCA security of $m\mathcal{I}D-KEM$.

$$Adv^{cr-CCA}_{\mathcal{G}KM} = Adv^{CCA}_{m\mathcal{I}D-KEM} = |Pr[b = b'] - \frac{1}{2}|$$

This means that if there exists no adversary $\mathcal{A}^{m\mathcal{I}D-KEM}$ who can break the CCA security of $m\mathcal{I}D-KEM$ with non-negligible advantage, then there cannot be any adversary $\mathcal{A}^{cr-\mathcal{G}KM}$ who can break the collusion resistance of $\mathcal{G}KM$ with non-negligible probability. □

8 An Illustration of the Generic Conversion to GKM

In this section, we present an example of the generalized transformation to GKM that was presented in Section 6. We construct the most efficient centralized GKM scheme proposed till date using the efficient $m\mathcal{I}D-KEM$ that was proposed by Delerablée [13] in 2007. This is the first efficient and scalable GKM scheme to achieve a constant size rekeying message framework. Before going into the details, we first recall Delerablée's scheme.

8.1 Delerablée's mID-KEM

Setup$(\mathbf{k}, \mathbf{N})$ — Given the security parameter k and the maximum number of receivers N, a bilinear map group system $\mathcal{B} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}(\cdot,\cdot))$ is constructed such that $|p| = k$. Also, two generators $f \in \mathbb{G}_1$ and $h \in \mathbb{G}_2$ and a secret value $\gamma \in \mathbb{Z}^*_p$ are randomly selected. Choose a cryptographic hash function $\mathcal{H} : \{0,1\}^* \rightarrow \mathbb{Z}^*_p$. The master secret key is defined as $MSK = (f, \gamma)$. The public key is $PK = (\omega, v, h, h^\gamma, \ldots, h^{\gamma^N})$ where $\omega = f^\gamma$, and $v = \hat{e}(f, h)$.

Extract$(\mathbf{MSK}, \mathbf{ID_i}, \mathbf{PK})$ — Given $MSK = (f, \gamma)$, the public key PK and the identity ID_i, it outputs $SK_{ID_i} = f^{\frac{1}{\gamma + \mathcal{H}(ID_i)}}$

Encapsulate$(\mathcal{S}, \mathbf{PK})$ — Assume for notational simplicity that $\mathcal{S} = \{ID_j\}_{j=1}^s$, with $s \leq N$. Given PK, it randomly picks $r \in \mathbb{Z}^*_p$ and computes $Hdr = (C_1, C_2)$ and $DEK \in \mathcal{K}$ where

$$C_1 = \omega^{-\alpha}, \qquad C_2 = h^{\alpha \cdot \prod_{i=1}^{s} (\gamma + \mathcal{H}(ID_i))}, \qquad DEK = v^\alpha$$

and outputs (Hdr, DEK).

Decapsulate$(\mathbf{S}, \mathbf{ID_i}, \mathbf{SK_{ID_i}}, \mathbf{Hdr}, \mathbf{PK})$ — In order to retrieve the DEK encapsulated in the header $Hdr = (C_1, C_2)$, the user with identity ID_i and the corresponding private key $SK_{ID_i} = f^{\frac{1}{\gamma + \mathcal{H}(ID_i)}}$ (with $ID_i \in \mathcal{S}$) computes

$$DEK = \left(\hat{e}(C_1, h^{p_{i,\mathcal{S}}(\gamma)}) \cdot \hat{e}(sk_{ID_i}, C_2) \right)^{\frac{1}{\prod_{j=1, j\neq i}^{s} \mathcal{H}(ID_j)}}$$

with

$$p_{i,\mathcal{S}}(\gamma) = \frac{1}{\gamma} \cdot \left(\prod_{j=1, j\neq i}^{s} (\gamma + \mathcal{H}(ID_j)) - \prod_{j=1, j\neq i}^{s} \mathcal{H}(ID_j) \right)$$

[20] $g_{t_{Challenge}}$ is retrieved from the list $\mathcal{L}_g$. Since $g_{t_{Challenge}} = r_{t_{Challenge}} \cdot \mathcal{F}(g_{t^-_{Challenge}})$, it can be seen that $r_{t_{Challenge}}$ can be computed using $g_{t_{Challenge}}$ and $g_{t^-_{Challenge}}$, both of which are available in $\mathcal{L}_g$.

8.2 The Centralized GKM Scheme from Delerablée's mID-KEM

Setup$(\mathbf{k}, \mathbf{N}, \mathcal{S}_{\mathbf{init}})$

- **Input.** Take as input the security parameter k, the maximum number of group members N, the set $\mathcal{S}_{init}$ of the identities of initial group members.
- A bilinear map group system $\mathcal{B} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}(,))$ is constructed such that $|p| = k$.
- Two generators $f \in \mathbb{G}_1$ and $h \in \mathbb{G}_2$ and a secret value $\gamma \in \mathbb{Z}_p^*$ are randomly selected.
- Choose a cryptographic hash function $\mathcal{H} : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$ and a *one-way function* $\mathcal{F} : \mathbb{Z}_p^* \rightarrow \mathbb{Z}_p^*$.
- Pick a random $g \in \mathbb{Z}_p^*$, a seed for the one-way function.
- The master secret key is defined as $MSK = (f, \gamma, g)$ and $PK = (\omega, v, h, h^{\gamma}, \ldots, h^{\gamma^N}, \mathcal{H}, \mathcal{F})$ is the public key where $\omega = f^{\gamma}$, and $v = \hat{e}(f, h)$.
- Choose a data encryption key DEK at random from the key space $\mathcal{K}$.
- Compute $SK_i = (f^{\frac{1}{\gamma + \mathcal{H}(ID_i)}}, g)$ for all $ID_i \in \mathcal{S}_{init}$ and securely send these keys to the corresponding members. Also send the initial DEK securely to these members.

Rekey$(\mathcal{S}, \mathbf{PK})$

- **Input.** Take as input the set $\mathcal{S}$ of the identities of current group members, and the public key PK.
- Pick a random $r \in \mathbb{Z}_p^*$ and update the *dynamic key* by using the *one-way function* $\mathcal{F}$ as $g \leftarrow r \cdot \mathcal{F}(g)$.
- Compute

$$C_1 = \omega^{-\alpha}, \qquad C_2 = h^{\alpha \cdot \prod_{i=1}^{s} (\gamma + \mathcal{H}(ID_i))}, \qquad DEK = v^{\alpha}$$

- Construct $Hdr_{GKM} = \langle Hdr \oplus g, r \rangle$, where $Hdr = (C_1, C_2)$ and broadcast it to the group.
- Every group member parses Hdr_{GKM} as (C_0, r), updates the second component of his secret key (the dynamic key) as $g \leftarrow r \cdot \mathcal{F}(g)$, and securely erases any copies of older g values.
- Every group member with identity ID_i will retrieve $Hdr = C_0 \oplus g$, parse $Hdr = (C_1, C_2)$, and compute

$$DEK = \left(\hat{e}(C_1, h^{p_{i,\mathcal{S}}(\gamma)}) \cdot \hat{e}(sk_{ID_i}, C_2) \right)^{\frac{1}{\prod_{j=1, j \neq i}^{s} \mathcal{H}(ID_j)}}$$

with

$$p_{i,\mathcal{S}}(\gamma) = \frac{1}{\gamma} \cdot \left(\prod_{j=1, j \neq i}^{s} (\gamma + \mathcal{H}(ID_j)) - \prod_{j=1, j \neq i}^{s} \mathcal{H}(ID_j) \right)$$

to obtain DEK.

Join$(\mathbf{ID_i}, \mathcal{S}, \mathbf{PK})$

- **Input.** Take as input the identity ID_i of a member who wishes to join the group, the set $\mathcal{S}$ of identities of current group members, and the public key PK.
- The joining member establishes a secure connection with the CA, who may perform some checks before authorizing the member to join the group. If authorized, compute $SK_i = (f^{\frac{1}{\gamma + \mathcal{H}(ID_i)}}, g)$ and securely send it to the joining member.
- Update the set of identities of current group members as $\mathcal{S} \leftarrow \mathcal{S} \cup \{ID_i\}$.
- Run **Rekey**$(\mathcal{S}, \mathbf{PK})$.

Leave$(\mathcal{L}, \mathcal{S}, \mathbf{PK})$

- **Input.** Take as input the set $\mathcal{L}$ of identities of members who wish to leave the group or are revoked, the set $\mathcal{S}$ of identities of current group members, and the public key PK.
- Update the set of identities of current group members as $\mathcal{S} \leftarrow \mathcal{S} - \mathcal{L}$.
- Run **Rekey**$(\mathcal{S}, \mathbf{PK})$.

Note. This $\mathcal{G}KM$ scheme is only sCPA secure against static adversaries since the underlying $m\mathcal{I}D - KEM$ is sCPA secure. As noted in [13], this scheme can be converted to an sCCA secure scheme by using the result of [8].

9 Conclusion

In this paper, we have identified the lack of formal framework and security model for Group Key Management. To fill this gap, in Sections 3 and 4, we proposed a generic framework for GKM and a fitting formal security model in which we defined the vital security properties that any GKM scheme should satisfy. We have also shown in Sections 6 and 7 how to convert any multi-receiver ID-based key encapsulation mechanism to a centralized GKM scheme and formally prove its security properties, assuming the security of the mID-KEM and the existence of one-way functions. Though simple and efficient, a drawback of our generic conversion is that the GKM inherits the security strength of the underlying mID-KEM only up to CCA. In Section 8, we also gave an illustration of our generic conversion taking the mID-KEM of [13].

Future Work. Some of the open problems in this direction would include construction of mID-KEMs which are efficient and secure against adaptive attacks. The generic conversion from mID-KEM would be complete if the security-inheritance of the resulting GKM goes further to CCA2.

References

1. Michel Abdalla, Eike Kiltz, and Gregory Neven. Generalized key delegation for hierarchical identity-based encryption. In *ESORICS*, pages 139–154, 2007.
2. Joonsang Baek, Reihaneh Safavi-Naini, and Willy Susilo. Efficient multi-receiver identity-based encryption and its application to broadcast encryption. In *Public Key Cryptography*, pages 380–397, 2005.
3. Manuel Barbosa and Pooya Farshim. Efficient identity-based key encapsulation to multiple parties. In *IMA Int. Conf.*, pages 428–441, 2005.
4. Dan Boneh and Matthew K. Franklin. Identity-Based Encryption from the Weil Pairing. In *CRYPTO*, pages 213–229, 2001.
5. Colin A. Boyd and Anish Mathuria. *Protocols for Key Establishment and Authentication.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.
6. B.Quinn. Ip multicast applications: Challenges and solutions. Bob Quinn, IP Multicast Applications: Challenges and Solutions, draft-quinn-multicastapps-00.txt, November 1998.
7. R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. Multicast Security: A Taxonomy and Some Efficient Constructions. In *Proceedings of the IEEE INFOCOM*, volume 2, pages 708–716, 1999.
8. Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-ciphertext security from identity-based encryption. In *EUROCRYPT*, pages 207–222, 2004.
9. Ran Canetti, Tal Malkin, and Kobbi Nissim. Efficient Communication-Storage Tradeoffs for Multicast Encryption. In *EUROCRYPT*, pages 459–474, New York, NY, USA, 1999. Springer-Verlag New York, Inc.
10. Isabella Chang, Robert Engel, Dilip D. Kandlur, Dimitrios E. Pendarakis, and Debanjan Saha. Key management for secure internet multicast using boolean function minimization techniques. In *INFOCOM*, pages 689–698, 1999.
11. Sanjit Chatterjee and Palash Sarkar. Multi-receiver identity-based key encapsulation with shortened ciphertext. In *INDOCRYPT*, pages 394–408, 2006.
12. Ling Cheung, Joseph A. Cooley, Roger Khazan, and Calvin Newport. Collusion-Resistant Group Key Management Using Attribute-Based Encryption. In *1st International Workshop on Group-Oriented Cryptographic Protocols*, 2007.
13. Cécile Delerablée. Identity-based broadcast encryption with constant size ciphertexts and private keys. In *ASIACRYPT*, pages 200–215, 2007.
14. Steven D. Galbraith, Keith Harrison, and David Soldera. Implementing the Tate Pairing. In *ANTS*, pages 324–337. Springer-Verlag, 2002.

15. H. Harney and C. Muckenhirn. *Group Key Management Protocol (GKMP) Specification.* RFC Editor, United States, 1997.
16. Guang huei Chiou and Wen-Tsuen Chen. Secure broadcasting using the secure lock. *IEEE Trans. Softw. Eng.*, 15(8):929–934, 1989.
17. Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.
18. Refik Molva and Alain Pannetrat. Scalable multicast security in dynamic groups. In *ACM Conference on Computer and Communications Security*, pages 101–112, 1999.
19. Sandro Rafaeli and David Hutchison. A Survey of Key Management for Secure Group Communication. *ACM Comput. Surv.*, 35(3):309–329, 2003.
20. Ryuichi Sakai and Jun Furukawa. Identity-based broadcast encryption. Cryptology ePrint Archive, Report 2007/217, 2007. http://eprint.iacr.org/.
21. Alan T. Sherman and David A. McGrew. Key Establishment in Large Dynamic Groups Using One-Way Function Trees. *IEEE Trans. Softw. Eng.*, 29(5):444–458, 2003.
22. Nigel P. Smart. Efficient key encapsulation to multiple parties. In *SCN*, pages 208–219, 2004.
23. Graham Steel. Group protocol attacks, 2006. http://homepages.inf.ed.ac.uk/gsteel/group-protocol-corpus/.
24. Michael Steiner, Gene Tsudik, and Michael Waidner. Cliques: A new approach to group key agreement. In *ICDCS*, pages 380–387, 1998.
25. Marcel Waldvogel, Germano Caronni, Dan Sun, Nathalie Weiler, and Bernhard Plattner. The VersaKey Framework: Versatile Group Key Management. *IEEE Journal on Selected Areas in Communications*, 17(9):1614–1631, sep 1999.
26. Chung Kei Wong, Mohamed G. Gouda, and Simon S. Lam. Secure Group Communications using Key Graphs. *IEEE/ACM Trans. Netw.*, 8(1):16–30, 2000.