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

Naga Naresh Karuturi*, §, Ragavendran Gopalakrishnan*, Rahul Srinivasan† and Pandu Rangan Chandrasekaran*

*Theoretical Computer Science Laboratory
Department of Computer Science and Engineering
Indian Institute of Technology Madras
Chennai, India
nnaresh@cse.iitm.ernet.in, ragav@cse.iitm.ernet.in, prangan@iitm.ac.in

†Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Mumbai, India
rahul.srinivasan@iitb.ac.in

Abstract — Group Key Establishment is fundamental for a variety of security mechanisms in group applications. It allows \( n \geq 2 \) principals to agree upon a common secret key. This can further be classified into Group Key Exchange (or Group Key Agreement), where all the principals participate in the construction of the key, and Group Key Transport (or Group Key Distribution), where the key is chosen by a singe principal and is then securely communicated to the others. Both these techniques can be analyzed in the context of either static or dynamic groups. Dynamic Group Key Establishment is better known as Group Key Management (GKM), as it involves not only the initial key establishment, but also efficient key management when group members join or leave the group. Dynamic Group Key Exchange is also known as decentralized or distributed GKM, while Dynamic Group Key Transport is known as centralized GKM. While there has been a lot of recent work in formal security models for Dynamic Group Key Exchange, little, if any, attention has been directed towards building a concrete framework and formal security model for centralized GKM. Many such schemes that have been proposed so far have been broken, as they cite ambiguous arguments and lack formal proofs. In this paper, we take a first step towards addressing this problem by providing firm foundations for centralized 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 dynamic groups demand. 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 Delerablé.


I. 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 [2] 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 establishing the DEKs among the authorized group members. Furthermore, changes in membership may require that the group key be refreshed. Such a key refreshing procedure prevents a joining (leaving) group member from decoding messages exchanged in the past (future), even if he has recorded earlier messages, in their encrypted form (encrypted with the old (new) keys).

However, establishing and managing the group key among valid members is a complex problem. Although refreshing the DEK before the join of a new member is trivial (in a centralized setting, for example, the central authority can simply 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 establish a new one, because the leaving group member knows the old key. Therefore, some other scalable mechanism to refresh the data encryption key must be provided.

§Work Supported by Project No. CSE/05-06/075/MICO/CPAN on Foundation Research in Cryptography sponsored by Microsoft Research India
**Group Key Establishment** — Group Key Establishment allows $n \geq 2$ principals to agree upon a common secret key. This general definition can further be shaped in two different classes — Group Key Exchange (Agreement) and Group Key Transport (Distribution).

- **Group Key Transport (Distribution).** A Group Key Transport (Distribution) protocol is a Group Key Establishment technique where a single entity (often known as the central authority) creates or otherwise obtains a secret value, and securely transfers it to the other members. This definition leaves open whether the central authority may be a group member. It is also imaginable to have some trusted third party (TTP) as the central authority.

- **Group Key Exchange (Agreement).** A Group Key Exchange (Agreement) protocol is a Group Key Establishment technique where a shared secret is derived by two or more group members as a function of the information contributed by each of them, such that no group member can predetermine the resulting value. Here, the main difference from Group Key Transport techniques is that no group member is allowed to choose the group key on behalf of the whole group.

Both group key establishment techniques can be analyzed in context of either static or dynamic groups. Of course, it is always possible to establish the group key for the modified group by restarting the protocol. However, this may be inefficient if groups are large or the protocol is expensive (in terms of communication or computational costs). Therefore, many Group Key Establishment protocols that are designed for dynamic groups provide more efficient operations for addition and exclusion of group members. Dynamic Group Key Establishment is better known as Group Key Management (GKM).

**Group Key Management** — As defined by Menezes et al. in [2], Group Key Management is the set of techniques and procedures supporting the establishment and maintenance of keying relationships between authorized parties that form a group. It plays an important role enforcing access control on the group key (DEK) (and consequently on the group). It plays an important role enforcing access control on the group key (DEK) (and consequently on the group communication). According to [2], group key management can be classified into the following three categories.

- **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. This category falls into the class of Dynamic Group Key Transport (Distribution) protocols.

- **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 GKM schemes are key independence, keys vs. data, type of communication, etc. This category falls into the class of Dynamic Group Key Exchange (Agreement) protocols.

- **Distributed Group Key Management** — The distributed Group 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. This category can fall under the class of Dynamic Group Key Exchange or Dynamic Group Key Transport, depending on how the group key is generated.

**Security Properties** — Any secure GKM scheme must satisfy the following desired security properties. We will define them formally in Section 2.

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

**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 [2] 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, [2] proposed an mID-KEM that has an efficient trade-off between the ciphertext size and the private key size. Recently, Abdalla et al. [2] proposed an mID-KEM construction where ciphertexts are of constant size, but private keys grow quadratic in the number of receivers. Furukawa [2] and Delerablé [2] 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.

A. **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) [?] is a cooperative generation between two protocol entities. There are several key generation algorithms viable for use in GKMP (i.e., RSA, Diffie-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 [?]).

- **Logical Key Hierarchy (LKH)** [?] — 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)** [?] — 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** [?] — 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** [?] — 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 [?] uses the Chinese Remainder Theorem to construct a secure lock that is used to lock the group decryption key. Because the lock is common among all valid members, the transmission efficiency of the decryption key is \( O(1) \) if the message size is disregarded. However, this method suffers from scalability problems.

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

The flat-table based schemes proposed by Waldvogel et al. [?] uses a table to reduce the number of keys stored at the KDC. When a member leaves, the KDC changes all the keys associated with that member. Chang et al. [?] use boolean function minimization to minimize the number of messages needed for Rekey, but this method is not collusion resistant. Some attribute based encryption schemes, like FT (CP-ABE) by Cheung et al. [?] are collusion resistant 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 [?] presents a somewhat formal proof, it is not clear as to how each security property is satisfied. Waldvogel et al. [?] 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 perfect forward secrecy. Some of the flat-table based schemes are not collusion resistant.

**B. Our Contribution**

Though a lot of work has been done in the development of formal frameworks and security models for Dynamic Group Key Exchange [?], no concrete framework and security model for centralized Group Key Management exists in literature. To the best of our knowledge, we are the first to propose a generic framework for centralized GKM, and more importantly, to present a formal security model that defines each of the security properties (forward secrecy, backward secrecy, perfect forward secrecy and collusion resistance). This is done in Section ??.

None of the existing centralized GKM schemes have been formally proven secure due to the lack of such a formal security model. Numerous attacks [?] have been mounted on various GKM schemes proposed so far. In Section ??, we construct adversarial games for each of the security properties mentioned above, to provide a framework in which one can formally prove a centralized 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 in Section ??, 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, in Section ??, 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. Finally, in Section ??, we illustrate our generalization by extending the efficient mID-KEM proposed in [?], to centralized GKM. This is the first GKM scheme to achieve constant-size rekeying message length.

**II. Preliminaries**

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

**A. 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 $A'$, every polynomial $p(\cdot)$, 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 $B$ in inverting a one-way function $F$ as

$$Adv_B^n = Pr \left[ F(B(F(x))) = F(x) \mid x \sim \{0, 1\}^n \right]$$

B. Bilinear Maps

We present the necessary facts about bilinear maps and bilinear map groups. Let $G$ be an additive cyclic group and $G_1$ be a multiplicative cyclic group, both of prime order $p$. A bilinear map or a bilinear pairing is a map $\hat{e} : G \times G \rightarrow G_1$ with the following properties.

• Bilinearity. For all $P, Q, R \in G$,
  
  - $\hat{e}(P + Q, R) = \hat{e}(P, R) \cdot \hat{e}(Q, R)$
  
  - $\hat{e}(P, Q + R) = \hat{e}(P, Q) \cdot \hat{e}(P, R)$
  
  - $\hat{e}(aP, bQ) = \hat{e}(P, Q)^{ab}$

• Non-Degeneracy. There exist $P, Q \in G$ such that $\hat{e}(P, Q) \neq 1_{G_1}$, where $1_{G_1}$ is the identity in $G_1$.

• Computability. There exists an efficient algorithm to compute $\hat{e}(P, Q)$ for all $P, Q \in G$.

Modified Weil pairing [2] and Tate pairing [1] are examples of cryptographic bilinear maps where $G$ is an elliptic curve group and $G_1$ is a subgroup of a finite field.

C. 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 $\mu(n) < \frac{1}{p(n)\cdot n}$. Negligible functions remain negligible when multiplied by any fixed polynomial.

III. A Formal Framework for Group Key Management

In centralized Group Key Management (centralized GKM) schemes, 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) among 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 transport, 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 [3], 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 [4] 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 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($k, N, S_{init}, E$)

Input. $k$ is a security parameter, $N$ is the maximum number of group members (the capacity)\(^1\), $S_{init}$ is the set of identifiers of initial group members and $E$ 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) Setup$_E(k, N)$. 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) Encapsulate$_E$(DEK, PK, MSK, $S$) — This algorithm takes as input the key $DEK$ to be encrypted\(^2\), the public key $PK$, the master secret key $MSK$ (if used) and the set $S$ 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.

\(^1\)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.

\(^2\)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)).
c) \text{Decapsulate}_\mathcal{E}(\text{Hdr}, \text{PK}, \text{SK}_i, 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, and the set S of authorized receivers. It returns the key DEK corresponding to the header Hdr.

The CA runs \text{Setup}_\mathcal{E}(k, N, \text{S}_{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 S_{init} of current group members is given through secure channels, his secret key SK_i and the initial DEK, which may be chosen randomly from the key space K.

2) \text{Rekey}(S, \text{PK}, \text{MSK}, \mathcal{E})

\text{Input.} 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 violated the protocol, meaning that he has already been compromised. Also, the CA can choose to update the public key as well, if required.

The CA has the pair (Hdr^\mathcal{E}, DEK^\mathcal{E}), after running \text{Encapsulate}_\mathcal{E}(\text{DEK}^\mathcal{E}, \text{PK}^\mathcal{E}, \text{MSK}^\mathcal{E}, S). He computes (Hdr, DEK) for the group and broadcasts Hdr.

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

3) \text{Join}(i, S, \text{PK}, \text{MSK}, \mathcal{E})

\text{Input.} i is the identifier of the member who wishes to join the group, 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 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 \( S \leftarrow S \cup \{ i \} \), and gives \( SK_i \) to the joining member through a secure channel.

The CA then runs \text{Rekey}(S, \text{PK}, \text{MSK}, \mathcal{E}).

4) \text{Leave}(\mathcal{L}, S, \text{PK}, \text{MSK}, \mathcal{E})

\text{Input.} \mathcal{L} is the set of identifiers of the members who wish to leave the group or are being banned (revoked), 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 \( S \leftarrow S - \mathcal{L} \).

The CA then runs \text{Rekey}(S, \text{PK}, \text{MSK}, \mathcal{E}).

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

\text{Note.} The CA may choose to perform the \text{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 \textit{perfect forward secrecy}.

IV. \textbf{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.

\text{A. 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 \( S_t \) 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 IV.1 summarizes the notations dealing with time.

\text{B. 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) \text{O}_\text{Join}(i) — This oracle simulates the \text{Join} algorithm of the GKM, to include the member i in the current group.

\text{Input.} i should be the identifier of a member who is not currently part of the group.

- The oracle aborts if \( i \notin S \).
- The set of identifiers of current group members is updated as \( S_{\text{new}} \leftarrow S_{\text{old}} \cup \{ i \} \).
- The \text{Rekey} algorithm is run and the new ciphertext is recorded.

2There is no ambiguity because, as we shall see, in every adversarial game, the adversary makes at most one corrupt query

3In other words, the group parameters used in generating the challenge ciphertext will be those at time \( t_{\text{challenge}} \)
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t )</td>
<td>An arbitrary instant of time</td>
</tr>
<tr>
<td>( t_{\text{now}} )</td>
<td>The current time instant (the present time)</td>
</tr>
<tr>
<td>( t_{\text{Corrupt}} )</td>
<td>The time at which the corrupt query was issued</td>
</tr>
<tr>
<td>( t_{\text{Challenge}} )</td>
<td>The time for which the challenge ciphertext is to be generated</td>
</tr>
<tr>
<td>( t_{\text{Leave}}(i) )</td>
<td>The time at which the user with identifier ( i ) most recently left the group</td>
</tr>
<tr>
<td>( t_{\text{Join}}(i) )</td>
<td>The time at which the user with identifier ( i ) most recently joined the group</td>
</tr>
<tr>
<td>( t_{\text{now}} )</td>
<td>The time instant just before ( t_{\text{now}} )</td>
</tr>
</tbody>
</table>

2) \( O_{\text{Leave}}(i) \) \(^6\) — 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 \( i \notin S_{\text{now}} \).
- The set of identifiers of current group members is updated as \( S_{\text{now}} \leftarrow S_{\text{now}} - \{i\} \).
- The Rekey algorithm is run and the new ciphertext is recorded.

3) \( O_{\text{Ciphertext}}(t) \) — 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 \( t > t_{\text{now}} \).
- 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) \( O_{\text{Decrypt}}(\text{Hdr}, t) \) — This oracle is used to retrieve the DEK from its encrypted form.

- **Input.** \( \text{Hdr} \) should be a ciphertext and \( t \) should be the present time or a time in the past.
- The oracle aborts if \( t > t_{\text{now}} \).
- The set \( S_t \) of group members at time \( t \) is recalled and the secret key \( SK_i \) corresponding to a user with identifier \( i \in S_t \) at time \( t \) is obtained.
- \( Hdr^{\xi} \) and \( SK_i^{\xi} \) are derived from \( Hdr \) and \( SK_i \) respectively.
- \( \text{Decapsulate}_{\xi}(\text{Hdr}^{\xi}, PK^{\xi}, SK_i^{\xi}, S_t) \) is run, and the resultant DEK is returned.

5) \( O_{\text{Corrupt}}(i, \text{type}) \) — This oracle simulates the compromise of a member.

- **Input.** \( i \) should be the identifier of a member, and \( \text{type} \) should be one of \( fs \) (forward secrecy), \( bs \) (backward secrecy) or \( pfs \) (perfect forward secrecy), indicating the type of security that is being attacked using this compromised member.
- The oracle aborts if \( \text{type} = pfs \) and \( i \notin S_{\text{now}} \) 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 \( \text{type} \) is \( fs, bs \) or \( pfs \), the secret key corresponding to the user with identifier \( i \) at time \( t_{\text{Leave}}(i), t_{\text{Join}}(i) \) or \( t_{\text{now}} \) 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.

C. Formal Definitions of Security

Normal multi-receiver cryptographic schemes which do not involve operations carried out over a time-line, but are just a collection of algorithms that are executed once, have two clearly defined extremes when describing the intensity of attacks — static attacks, while proving the security against which, the adversary is required to submit the identifiers of the entities whom he would attack during the challenge phase of the game, and adaptive attacks, in which case, the adversary is under no such restriction. In Group Key Management, we consider static and adaptive security not only along the dimension of receiver identifiers, but also along the time dimension. While describing adversarial games for time-static security, the adversary would be required to submit beforehand the time at which he would like the challenge to be generated, which would eventually be given to him during the challenge phase. The adversary is not required to do so for time-adaptive security. From now, when we simply say “static” (“adaptive”), we mean static (adaptive) in both dimensions. In contexts where a mixed security is discussed, we will be explicit with respect to the two dimensions.

Before describing the adversarial games involved, we formally define the four security notions that were informally discussed in Section ???. For simplicity, we define only the CCA2 security against adaptive attacks here. We discuss briefly about other notions in a separate paragraph at the end of this section.

\(^6\)For a set \( L \) of leaving members, this oracle can be called repeatedly on each member in \( L \).
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 $N(.)$, the advantage $Adv_{GKM}^{fs}$ of any probabilistic polynomial time adversary $A_{fs}^{GKM}$ in the game $G_{CCA2}$ against a challenger $C_{fs}^{GKM}$ is negligible in $k$, the security parameter.

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 $N(.)$, the advantage $Adv_{GKM}^{bs}$ of any probabilistic polynomial time adversary $A_{bs}^{GKM}$ in the game $G_{CCA2}$ against a challenger $C_{bs}^{GKM}$ is negligible in $k$, the security parameter.

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 $N(.)$, the advantage $Adv_{GKM}^{pfs}$ of any probabilistic polynomial time adversary $A_{pfs}^{GKM}$ in the game $G_{CCA2}$ against a challenger $C_{pfs}^{GKM}$ is negligible in $k$, the security parameter.

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 $N(.)$, the advantage $Adv_{GKM}^{cr}$ of any probabilistic polynomial time adversary $A_{cr}^{GKM}$ in the game $G_{CCA2}$ against a challenger $C_{cr}^{GKM}$ is negligible in $k$, the security parameter.

These definitions are not complete because we have neither described the adversarial games nor defined the advantage of an adversary. In Game $??$, we describe formally a generic adversarial CCA2 game $G_{CCA2}$. Below, we define the games $G_{CCA2}^{fs}$, $G_{CCA2}^{bs}$ and $G_{CCA2}^{pfs}$ as special cases of this generic game. The collusion resistance game $G_{CCA2}^{cr}$ is described in Game $??$.

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

- **Adversarial Game for Forward Secrecy**

  $G_{CCA2}^{fs} = G_{CCA2}(C^{fs}, A^{fs}, f_s)$

  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 past ciphertexts intended to the group when he is no longer part of the group. Since we are talking about a corrupted member who has left the group, during the corrupt phase, we give the adversary the secret key of the corrupted member at the time of his leaving the group. We allow the adversary to enter the challenge phase at any time after the corrupt phase. In particular, he may choose to make the challenge query at the time that 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 in the group.

- **Adversarial Game for Backward Secrecy**

  $G_{CCA2}^{bs} = G_{CCA2}(C^{bs}, A^{bs}, bs)$

  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 was not part of the group. Since we are talking about a corrupted member who has joined the group during the corrupt phase, we give the adversary the secret key of the corrupted member at the time of his joining the group. And, 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 $t_{challenge}$ during which the challenge is to be generated. Of course, since we are dealing with backward secrecy, the corrupted member should not be part of the group during $t_{challenge}$.

- **Adversarial Game for Perfect Forward Secrecy**

  $G_{CCA2}^{pfs} = G_{CCA2}(C^{pfs}, A^{pfs}, pfs)$

  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. Accordingly, we give the adversary the secret key of the corrupted member at the time of corruption. 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 in the group during $t_{challenge}$. Otherwise, it would model backward secrecy.

- **Adversarial Game for Collusion Resistance**

  $G_{CCA2}^{cr} = G_{CCA2}(C^{cr}, A^{cr}, cr)$

  This game is described in Game $??$. 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 of all the users who are currently not part of the group to the adversary.

**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

---

7Since 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.
This generic game is played between a challenger $C^{GKM}$ and an adversary $A^{GKM}$. The variable type signifies the type of security that the adversary claims he can break, and can take on any of three values $fs, bs, pfs$.

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 Setup$(k, N, S_{\text{init}}, \mathcal{E})$, for any choice of $S_{\text{init}}$ by the adversary. The public key $PK$ is given to the adversary $A^{GKM}$. 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 $O_{\text{Join}}(1)$ and $O_{\text{Leave}}(1)$. 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 $O_{\text{Ciphertext}}(t)$. These queries help the adversary to retrieve the $Hdr$ corresponding to the most recent Rekey operation performed at or before a past time $t$ (Note the Join and Leave operations also involve a Rekey operation and such rekeys are also taken into account).
- Queries of the form $O_{\text{Decrypt}}(Hdr, t)$. 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 $u \in S_t$.

**Corrupt Phase** — The adversary, at any time $t_{\text{Corrupt}}$ of his choice, invokes $O_{\text{Corrupt}}(i_c, \text{type})$, 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 $t_{\text{Join}}(i_c)$, $t_{\text{Leave}}(i_c)$, or $t_{\text{now}}$, 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 $O_{\text{Join}}, O_{\text{Leave}}, O_{\text{Ciphertext}}$ and $O_{\text{Decrypt}}$.

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

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

The challenger runs $\text{Encapsulate}_{\mathcal{E}}(DEK^x, PK^x, MSK^x, S_{\text{Challenge}})$, at the end of which he has the $(Hdr^x, DEK^x)$ pair. Using this, he computes $(Hdr^*, DEK^*)$ corresponding to time $t_{\text{Challenge}}$, 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 $K$ and challenges the adversary with $(Hdr^*, K_b, K_{1-b})$.

**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 $(Hdr^*, t_{\text{Challenge}})$ is not given as a query to $O_{\text{Decrypt}}$.

**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 $Adv_{GKM}^{CCA2} = |Pr[b' = b] - \frac{1}{2}|$

**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.
This game is played between the challenger $C^{cr^−GKM}$ and the adversary $A^{cr^−GKM}$. 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 $E$. 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 $G^{GKM}_{CCA2}(C^{cr^−GKM}, A^{cr^−GKM}, \cdot, \cdot)$.

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

**Challenge Phase** — The adversary issues one challenge query to the challenger $C^{cr^−GKM}$ 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 $i \notin S_{Challenge}$. The challenger obtains the $(Hdr^E, DEK^E)$ pair by running $Encapsulate_E(DEK^E, PK^E, MSK^E, S_{Challenge})$. 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 $K$ and challenges the adversary with $(Hdr^*, K_0, K_1)$.

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

**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 $Adv_{G^{GKM}_{CCA2}}^{cr^−GKM} = |Pr[b' = b] - \frac{1}{2}|$

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 $G^{GKM}_{CCA}$ for adaptive CCA security is the same as the game $G^{GKM}_{CCA2}$, except that in the Query phase that follows the Challenge phase, the adversary is denied access to $O_{Decrypt}$ altogether.

- **Adaptive CPA Security** — The adversarial game $G^{GKM}_{CPA}$ for adaptive CPA security is the same as the game $G^{GKM}_{CCA2}$, except that in all the Query phases, the adversary is denied access to $O_{Decrypt}$.

- **Static Security** — The adversarial games $G^{GKM}_{CCA2}$, $G^{GKM}_{CPA}$, and $G^{GKM}_{CPA}$ for static security are the same as the respective games for adaptive security, except that the adversary must submit $S_{Challenge}$ (for identifier-static) and $t_{Challenge}$ (for time-static) to the challenger in the beginning of the Setup phase.

V. 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.

A. 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 $SK_{ID_1}$ of group members with identities $ID_1$, and securely transmits these keys to them. The sender uses the public key $PK$ and identities of the 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 algorithms$^8$.

**Setup($k, N$)** — This algorithm inputs 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($MSK, ID_1, PK$)** — This algorithm inputs the master secret key $MSK$, a user identity $ID_1$, and the public key $PK$, and outputs the private key $SK_{ID_1}$ of the user, which is securely transported to the user.

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

**Decapsulate($S, ID_1, SK_{ID_1}, Hdr, PK$)** — This algorithm inputs the set $S$ of identities of the intended receivers, the identity $ID_1$ of one of the intended receivers, and the corresponding private key $SK_{ID_1}$, a header $Hdr$, and the public key $PK$. If $ID_1 \in S$, the algorithm outputs the key $K$.

$^8$Our description of an mID-KEM does fall into the generic framework of the underlying mKEM discussed in Section 7; the only difference is that the Setup algorithm is split here into two algorithms Setup and Extract
Game V.1  \( \mathcal{G}_{\text{CCA}}^{\text{mID-KEM}} \)

This game is played between the challenger \( \mathcal{C}_{\text{mID-KEM}} \) and the adversary \( \mathcal{A}_{\text{mID-KEM}} \). 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 Setup\((k,N)\) and the public key \( PK \) is given to the adversary \( \mathcal{A}_{\text{mID-KEM}} \).

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

- Queries of the form \( \mathcal{O}_{\text{Extract}}(\text{id}_i) \) — The adversary can use this query to learn the secret keys of any of the members of his choice.
- Queries of the form \( \mathcal{O}_{\text{Decapsulate}}(\text{id}_i,S,\text{Hdr}) \) — 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 \( S^* \) of identities of users of the adversary’s choice. The only restriction is that \( S^* \) 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 \( S^* \) as input to obtain a \((\text{Hdr}^*, \text{DEK}^*)\) pair. He then chooses a bit \( b \in \{0,1\} \) at random and sets \( K_b \) to \( \text{DEK}^* \) and \( K_{1-b} \) to a random element from the key space \( K \). He then challenges the adversary with \((\text{Hdr}^*,K_b,K_{1-b})\).

**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 \( S^* \).
- He should not query the Decapsulate oracle with \((\text{id}_i,S^*,\text{Hdr}^*)\), for any \( \text{id}_i \in S^* \).

**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 \( \text{Adv}_{\text{CCA}}^{\text{mID-KEM}} = |Pr[b' = b] - \frac{1}{2}| \).

**B. Security Model for mID-KEM**

The adversarial game involves a challenger who presents 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) \( \mathcal{O}_{\text{Extract}}(\text{id}_i) \) — Here, \( \text{id}_i \) is the identity of a user. The oracle returns the secret key \( SK_{ID_i} \) of the user by using the Extract algorithm.

2) \( \mathcal{O}_{\text{Decapsulate}}(\text{id}_i,S,\text{Hdr}) \) — Here, \( \text{id}_i \) is the identity of an intended user, \( S \) is the set of identities of the intended (privileged) users, and \( \text{Hdr} \) is a header to be decrypted. The oracle returns the DEK corresponding to \( \text{Hdr} \) by using the Decapsulate algorithm.

We define CCA2 security for mID-KEM using the adversarial game \( \mathcal{G}_{\text{CCA}}^{\text{mID-KEM}} \) that is described in Game ??.

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

**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 \( \mathcal{G}_{\text{CCA}}^{\text{mID-KEM}} \) for adaptive CCA security is the same as the game \( \mathcal{G}_{\text{CCA}}^{\text{mID-KEM}} \), except that in the Query phase that follows the Challenge phase, the adversary is denied access to \( \mathcal{O}_{\text{Decrypt}} \) altogether.

- **Adaptive CPA Security** — The adversarial game \( \mathcal{G}_{\text{CPA}}^{\text{mID-KEM}} \) for adaptive CPA security is the same as the game \( \mathcal{G}_{\text{CCA}}^{\text{mID-KEM}} \), except that in all the Query phases, the adversary is denied access to \( \mathcal{O}_{\text{Decrypt}} \).

- **Static Security** — The adversarial games \( \mathcal{G}_{\text{CCA}}^{\text{mID-KEM}}, \mathcal{G}_{\text{CPA}}^{\text{mID-KEM}} \) and \( \mathcal{G}_{\text{CPA}}^{\text{mID-KEM}} \) 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 \( S^* \) of identities of users he wishes to be challenged upon.\(^9\)

VI. A GENERIC CONVERSION TO CENTRALIZED GKM FROM mID-KEM

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

\(^9\)Consequently, in Query Phase 1 of \( \mathcal{G}_{\text{CCA}}^{\text{mID-KEM}} \), the adversary should not query the Extract oracle for any identities that are present in \( S^* \).
Consider the following trivial (and hypothetical) construction of $GKM$. For Setup, run the Setup algorithm of $mID - KEM$, make the public key public, run the Extract algorithm of $mID - KEM$ 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 $mID - KEM$ 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 $GKM$ of the current group members accordingly and do a Join operation. 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.

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 $mID - KEM$ 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. The XOR operation is done bitwise. $g$ is represented as bits and is padded with additional zeroes if necessary.

Formally, $GKM$ consists of the following algorithms, all of which are run by the CA, who plays the role of the PKG of $mID - KEM$ as well.

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

**Note.** We refer to the second component of the secret key SK$_{ID_i}$, which is common to all the group members, as the dynamic key. It is “dynamic” because, as we shall see, it is updated regularly during every Rekey operation.

**Rekey**($S, PK$)
- **Input.** Take as input the set $S$ of the identities of current group members, and the public key $PK$.
- Select a random $r \in \mathbb{Z}_p^*$ and update the dynamic key by using the one-way function $\mathcal{F}$ as $g \leftarrow r \cdot \mathcal{F}(g)$.
- Run Encapsulate$_{mID - KEM}(S, PK_{mID - KEM})$ to obtain a $(Hdr_{mID - KEM}, DEK)$ pair.
- Construct $Hdr_{GKM} = Hdr_{mID - KEM} \oplus (g)$ and broadcast $(Hdr_{GKM}, r)$ to the group.
- Every group member also updates the second component of his secret key (the dynamic key) as $g \leftarrow r \cdot \mathcal{F}(g)$ and securely erases the old copy of $g$ values.
- Every group member with identity $ID_i$ will retrieve $Hdr_{mID - KEM} = Hdr_{GKM} \oplus g$ and run Decapsulate$_{mID - KEM}(S, ID_i, SK_{ID_i}, Hdr_{mID - KEM}, PK_{mID - KEM})$ 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**($ID_i, S, PK$)
- **Input.** Take as input the identity $ID_i$ of a member who wishes to join the group, the set $S$ 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 Extract$_{mID - KEM}(ID_i)$ to obtain the secret key SK$_{ID_i}$. Compute SK$_{ID_i} = (SK_{mID - KEM}, g)$ and securely send it to the joining member.
- Update the set of identities of current group members as $S \leftarrow S \cup \{ID_i\}$.
- Run Rekey($S, PK$).

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

We now prove that $\mathcal{GKM}$ is secure against adaptive\(^{11}\) Chosen Ciphertext Attacks (CCA) with respect to all the four security properties by assuming the adaptive 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 $mID-KEM$ to an adversary of $\mathcal{GKM}$, we will be running the following two adversarial games in parallel.

- $\mathcal{G}_m^{mID-KEM}$ — The CCA game corresponding to $mID-KEM$. The challenger for this game is denoted by $\mathcal{C}^{mID-KEM}$ and the adversary for this game is denoted by $\mathcal{A}^{mID-KEM}$.

- $\mathcal{G}_m^{(-)GKM}$ — The ( - )-CCA game corresponding to $\mathcal{GKM}$. Here, ( - ) can refer to $f_s$, $b_s$, $p_f$ or $cr$ depending on the security property that is being proved. The challenger and adversary for this game are denoted by $\mathcal{C}^{(-)GKM}$ and $\mathcal{A}^{(-)GKM}$ respectively.

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 $\mathcal{G}_m^{GCA}$. Before presenting the formal proof, we give a short informal overview of the two proof techniques that we employ.

- **Proofs for Forward Secrecy, Backward Secrecy and Collision Resistance** — For these properties, we shall be reducing $A^{mID-KEM}$ to $A^{(-)GKM}$. That is, we assume the existence of an adversary $A^{(-)GKM}$ who can break a particular security property of $\mathcal{GKM}$ and use him to construct the adversary $A^{mID-KEM}$ who can break the security of $mID-KEM$. We let $A^{mID-KEM}$ take on the role of $\mathcal{C}^{(-)GKM}$ and interact with $A^{(-)GKM}$ on one side through the game $G_{mID-KEM}$ and simultaneously interact with $\mathcal{C}^{GCA}$ through the game $G_{GCA}$. Thus, the task of $A^{mID-KEM}$ is to use its interaction with $A^{(-)GKM}$ to try and win against $\mathcal{C}^{mID-KEM}$.

- **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 somewhat 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. The algorithm acts as the challenger $\mathcal{C}^{f_s-GKM}$ of the adversary $A^{f_s-GKM}$, and interacts with him through the game $G_{GCA}$. Thus, the task of the algorithm is to force $A^{f_s-GKM}$ to invert the one-way function $\mathcal{F}$, if at all he is to win $G_{GCA}$.

We now describe the working of $\mathcal{C}^{(-)GKM}$, who is an important entity in all our proofs.\(^{12}\) He maintains five lists $\mathcal{L}_c$, $\mathcal{L}_s$, $\mathcal{L}_g$, $\mathcal{L}_j$ and $\mathcal{L}_t$ as described below.

- $\mathcal{L}_c$ contains entries of the form $\langle t, H_{dr}^{GKM} \rangle$, where $H_{dr}^{GKM}$ is the broadcast ciphertext of the Rekey operation performed at time $t$.
- $\mathcal{L}_s$ contains entries of the form $\langle t, S_t \rangle$, where $S_t$ 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 $\langle ID, t_{join}(ID) \rangle$. Recall that $t_{join}(ID)$ 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.
- $\mathcal{L}_t$ contains entries of the form $\langle ID, t_{leave}(ID) \rangle$. Recall that $t_{leave}(ID)$ 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.

$\mathcal{C}^{(-)GKM}$, acting as the challenger for $A^{(-)GKM}$, must provide access to all the oracles involved in $\mathcal{G}_{GCA}$. In those three proofs in which he is also an adversary for $mID-KEM$, he has access to the oracles provided by $\mathcal{C}^{mID-KEM}$, namely $\mathcal{C}^{mID-KEM}_{Extract}$ and $\mathcal{C}^{mID-KEM}_{Decapsulate}$. In the proof in which there is no access to these oracles, he can simulate them himself.\(^{13}\) In any case, we describe how $\mathcal{C}^{(-)GKM}$ simulates the oracles of $\mathcal{GKM}$ using those of $mID-KEM$ and a little bookkeeping.

- $\mathcal{C}^{Join(ID)}$ — $\mathcal{C}^{(-)GKM}$ does the following.
  1) Retrieve the last entry, $(t', S_{t'})$, from $\mathcal{L}_s$ and check if $ID_t \in S_{t'}$. If so, then abort. Else, set $S_{t_{new}} = S_{t'} \cup \{ID_t\}$ and append $(t_{new}, S_{t_{new}})$ to $\mathcal{L}_s$.
  2) Retrieve $g_{t_{new}}$ from $\mathcal{L}_g$ ($g_{t_{new}} = g_{t''}$, where $(t'', g_{t''})$ is the last entry in $\mathcal{L}_g$), pick a random $r \in \mathbb{Z}_p^*$, compute $g_{t_{new}} = g^r$ and append the entry $(t_{new}, g_{t_{new}})$ to $\mathcal{L}_g$.
  3) Run $Encapsulate_{mID-KEM}(S_{t_{new}}, PK_m^{mID-KEM})$ to obtain $H_{dr}^{mID-KEM}$, compute $H_{dr}^{GKM} = \langle H_{dr}^{mID-KEM} \oplus g_{t_{new}}, r \rangle$ and append the entry $(t_{new}, H_{dr}^{GKM})$ to $\mathcal{L}_c$.
  4) Record the join by appending the entry $(ID_t, t_{new})$ to $\mathcal{L}_j$. If there already exists an entry corresponding to $ID_t$, overwrite it.

- $\mathcal{C}^{Leave(ID)}$ — $\mathcal{C}^{(-)GKM}$ does the following.
  1) Retrieve the last entry, $(t', S_{t'})$, from $\mathcal{L}_s$ and check if $ID_t \notin S_{t'}$. If so, then abort. Else, set $S_{t_{new}} = S_{t'} - \{ID_t\}$ and append $(t_{new}, S_{t_{new}})$ to $\mathcal{L}_s$.
  2) Retrieve $g_{t_{new}}$ from $\mathcal{L}_g$ ($g_{t_{new}} = g_{t''}$, where $(t'', g_{t''})$ is the last entry in $\mathcal{L}_g$), pick a random $r \in \mathbb{Z}_p^*$, compute $g_{t_{new}} = g^r$ and append the entry $(t_{new}, g_{t_{new}})$ to $\mathcal{L}_g$.
  3) Run $Encapsulate_{mID-KEM}(S_{t_{new}}, PK_m^{mID-KEM})$ to obtain $H_{dr}^{mID-KEM}$, compute $H_{dr}^{GKM} = \langle H_{dr}^{mID-KEM} \oplus g_{t_{new}}, r \rangle$ and append the entry $(t_{new}, H_{dr}^{GKM})$ to $\mathcal{L}_c$.

\(^{11}\)Both time-adaptive and identity-adaptive

\(^{12}\)It must be kept in mind that in the proofs for forward secrecy, backward secrecy and collusion resistance, $\mathcal{C}^{(-)GKM}$ is also $\mathcal{A}^{mID-KEM}$.

\(^{13}\)He is able to do so because there is no game $\mathcal{G}_{GCA}$ and no corresponding challenger to win against.
We now present the four security theorems and their formal acts with.

1) When \( C \) issues a challenge query, specifying the set \( S_{\text{Challenge}} \) of \( t_{\text{Challenge}} \) oracles, \( O_{\text{Join}}, O_{\text{Leave}}, O_{\text{Ciphertext}} \) and \( O_{\text{Decrypt}} \), both \( A_{fs-GKM} \) and \( A_{mID-KEM} \) proceeds as follows:

- **O_{Ciphertext}(t) → C(-)^{\cdot-GKM} aborts if \( t > t_{\text{now}} \). Otherwise, he retrieves, if present, the entry \( (t', Hdr_{GKM}) \) from \( L_{c} \) such that \( t' \) is the most recent (numerically largest) time stamp satisfying \( t' \leq t \) and returns \( Hdr_{GKM} \). If no such entry is present, he returns \( \perp \).**

- **O_{Decrypt}(Hdr_{GKM}, t) → C(-)^{\cdot-GKM} aborts if \( t > t_{\text{now}} \). Otherwise, he does the following.**
  1. Retrieve, if present, the entries \( (t', S_{t}) \) from \( L_{s} \) and \( (t', g_{t}) \) from \( L_{g} \) 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 \( Hdr_{mID-KEM} = Hdr_{GKM} \oplus g_{t} \) corresponding to \( mID-KEM \) and return the result of \( O_{mID-KEM}(ID_{s}, S_{t}, Hdr_{mID-KEM}) \), where \( ID_{s} \) is chosen at random from the set \( S_{t} \).

- **O_{Corrupt}(ID_{t}, type) → C(-)^{\cdot-GKM} does the following.**
  1. When \( type = fs \), retrieve if present, the entries \( (ID_{t}, t_{\text{Leave}}(ID_{t})) \) and \( (t_{\text{Join}}(ID_{t}), g_{t_{\text{Join}}}(ID_{t})) \) from \( L_{t} \) and \( L_{g} \) respectively. If no such entries are present, return \( \perp \). Obtain \( S_{ID_{t}} \) by querying \( O_{mID-KEM}(ID_{t}) \) and return \( SK_{ID_{t}} = (SK_{ID_{t}}^{\text{act}}, g_{t_{\text{Join}}}(ID_{t})) \).
  2. When \( type = bs \), retrieve if present, the entries \( (ID_{t}, t_{\text{Join}}(ID_{t})) \) and \( (t_{\text{Join}}(ID_{t}), g_{t_{\text{Join}}}(ID_{t})) \) from \( L_{t} \) and \( L_{g} \) respectively. If no such entries are present, return \( \perp \). Obtain \( S_{ID_{t}} \) by querying \( O_{mID-KEM}(ID_{t}) \) and return \( SK_{ID_{t}} = (SK_{ID_{t}}^{\text{act}}, g_{t_{\text{Join}}}(ID_{t})) \).
  3. When \( type = ps \), retrieve the last entry \( (t, g_{t}) \) from \( L_{g} \) query \( O_{Extract}(ID_{t}) \) to obtain \( S_{ID_{t}} \) and return \( SK_{ID_{t}} = (SK_{ID_{t}}^{\text{act}}, g_{t}) \).

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

**Theorem 1.** \( GKM \) is \( fs \)-CCA secure if \( mID-KEM \) is at least CCA secure.

**Proof.** Here, we describe how the adversary \( A_{mID-KEM} \) on one side acts as the challenger \( C_{fs-GKM} \) who interacts with \( A_{fs-GKM} \), while simultaneously interacting with \( C_{mID-KEM} \) 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 \( G_{CCA} \).

1) **Setup Phase** — The challenger \( C_{mID-KEM} \) runs \( Setup_{mID-KEM}(k, N) \) to obtain \( PK_{mID-KEM} \), and gives it to \( A_{mID-KEM} \), who constructs \( PK = (PK_{mID-KEM}, F, mID-KEM) \) and gives it to \( A_{fs-GKM} \). He also picks a random seed \( q \) from \( \mathbb{Z}_{p}^{*} \) and sets the master secret key \( MSK \) to \( (MSK_{mID-KEM}, g) \).

2) **Query Phase 1** — \( A_{fs-GKM} \) is allowed to query the oracles \( O_{Join}, O_{Leave}, O_{Ciphertext} \) and \( O_{Decrypt} \).

3) **Corrupt Phase** — \( A_{fs-GKM} \) chooses \( ID_{t} \), an identity which he wants to corrupt and makes the query \( O_{Corrupt}(ID_{t}, fs) \) at time \( t_{Corrupt} \) (which is the choice of \( A_{fs-GKM} \)).

4) **Query Phase 2** — \( A_{fs-GKM} \) can query the oracles as in Query Phase 1.

5) **Challenge Phase** — \( A_{fs-GKM} \) issues one challenge query to its challenger \( A_{mID-KEM} \), at time \( t_{\text{Challenge}} \), which is the choice of \( A_{fs-GKM} \), subject to the restriction that \( ID_{t} \notin S_{\text{Challenge}} \). Now, \( A_{mID-KEM} \) does the following before responding with the challenge:
   - Retrieve the set \( S_{\text{Challenge}} \) from the list \( L_{s} \).
   - Issue a challenge query, specifying the set \( S_{\text{Challenge}} \), to the challenger \( C_{mID-KEM} \).
   - Receive the challenge \( (Hdr_{mID-KEM}^{*}, K_{0}, K_{1}) \).
   - Compute \( Hdr_{mID-KEM} = (Hdr_{mID-KEM} \oplus g_{t_{\text{Challenge}}}, t_{\text{Challenge}}) \).
   - Compute \( A_{mID-KEM} \) returns \( (Hdr_{mID-KEM}^{*}, K_{0}, K_{1}) \) as the challenge to \( A_{fs-GKM} \).

6) **Guess Phase** — \( A_{fs-GKM} \) outputs a bit \( b' \in \{0, 1\} \) as its guess. \( A_{mID-KEM} \) passes on \( b' \) as its guess to \( C_{mID-KEM} \).

It is easy to see that the advantage of \( A_{fs-GKM} \) in breaking the forward secrecy of \( GKM \) is the same as that of \( A_{mID-KEM} \) in breaking the CCA security of \( mID-KEM \).

\[
Adv_{GKM}^{fs-CCA} = Adv_{mID-KEM}^{CCA} = \left| Pr[b = b'] - \frac{1}{2} \right|
\]

This means that if there exists no adversary \( A_{mID-KEM} \) who can break the CCA security of \( mID-KEM \) with non-negligible advantage, then there cannot be any adversary \( A_{fs-GKM} \) who can break the forward secrecy of \( GKM \) with non-negligible advantage.

**Theorem 2.** \( GKM \) is \( bs \)-CCA secure if \( mID-KEM \) is at least CCA secure.

**Proof.** Here, we describe how the adversary \( A_{mID-KEM} \) on one side acts as the challenger \( C_{bs-GKM} \) who interacts with \( A_{bs-GKM} \), while simultaneously interacting with \( C_{mID-KEM} \) on the other side, trying to win against him. Since the two games are being run in parallel and we describe the events in

\( ^{14}g_{t_{\text{Challenge}}} \) is retrieved from the list \( L_{g} \). Since \( g_{t_{\text{Challenge}}} = r_{t_{\text{Challenge}}} \cdot F(g_{t_{\text{Challenge}}}) \), it can be seen that \( r_{t_{\text{Challenge}}} \) can be computed using \( g_{t_{\text{Challenge}}} \) and \( g_{t_{\text{Challenge}}} \), both of which are available in \( L_{g} \).
chronological order, the description below switches between the phases of the two games. Again, we present the description from the point of view of the game $G_{CCA}^{t-\text{GKM}}$.

1) Setup Phase — The challenger $C^{\text{mID}^{-\text{KEM}}}$ runs $Setup_{\text{mID}^{-\text{KEM}}}(k, N)$ to obtain $PK^{\text{mID}^{-\text{KEM}}}$, and gives it to $A^{\text{mID}^{-\text{KEM}}}$, who constructs $PK = \langle PK^{\text{mID}^{-\text{KEM}}}, F, \text{mID} - \text{KEM} \rangle$ and gives it to $A^{\text{bs}^{-\text{GKM}}}$. He also picks a random seed $g$ from $\mathbb{Z}_p^*$ and sets the master secret key $MSK$ to $(MSK_{\text{mID}^{-\text{KEM}}}, g)$.

2) Query Phase 1 — $A^{\text{bs}^{-\text{GKM}}}$ is allowed to query the oracles $O_{\text{Join}}, O_{\text{Leave}}, O_{\text{Ciphertext}}$ and $O_{\text{Decrypt}}$.

3) Corrupt Phase — $A^{\text{bs}^{-\text{GKM}}}$ chooses $ID_{i, s}$, an identity which he wants to corrupt and makes the query $O_{\text{Corrupt}}(ID_{i, s})$ at time $t_{\text{Corrupt}}$ (which is the choice of $A^{\text{bs}^{-\text{GKM}}}$).

4) Query Phase 2 — $A^{\text{bs}^{-\text{GKM}}}$ can query the oracles as in Query Phase 1.

5) Challenge Phase — $A^{\text{bs}^{-\text{GKM}}}$ issues one challenge query to its challenger $A^{\text{mID}^{-\text{KEM}}}$, specifying a time $t_{\text{Challenge}}$ (which is the choice of $A^{\text{bs}^{-\text{GKM}}}$), subject to the restrictions that $ID_{i, s} \notin S_{\text{Challenge}}$ and $t_{\text{Challenge}} \leq t_{\text{Join}}(ID_{i, s})$. Now, $A^{\text{mID}^{-\text{KEM}}}$ does the following before responding with the challenge.

- Retrieve the set $S_{\text{Challenge}}$ from the list $L_s$.
- Issue a challenge query, specifying the set $S_{\text{Challenge}}$, to the challenger $C^{\text{mID}^{-\text{KEM}}}$.
- Receive the challenge $(\text{Hdr}_{\text{mID}^{-\text{KEM}}, \text{b}}^s, K_0, K_1, K_r)$.
- Compute $\text{Hdr}_{\text{GKM}}^s$ as $\langle \text{Hdr}_{\text{mID}^{-\text{KEM}}, \text{b}}^s \oplus g_{\text{Challenge}}, r_{\text{Challenge}} \rangle$. $A^{\text{mID}^{-\text{KEM}}}$ returns $(\text{Hdr}_{\text{GKM}, \text{b}}^s, K_0, K_1)$ as the challenge to $A^{\text{bs}^{-\text{GKM}}}$.

6) Guess Phase — $A^{\text{bs}^{-\text{GKM}}}$ outputs a bit $b' \in \{0, 1\}$ as its guess. $A^{\text{mID}^{-\text{KEM}}}$ passes on $b'$ as its guess to $C^{\text{mID}^{-\text{KEM}}}$.

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

$$Adv_{\text{bs}^{-\text{GKM}}}^{\text{CCA}} = Adv_{\text{mID}^{-\text{KEM}}}^{\text{CCA}} = \Pr[b = b'] - \frac{1}{2}$$

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

**Theorem 3.** $G_{\text{KM}}$ is pfs-CCA secure if inverting $F$ is hard.

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

1) Setup Phase — $C^{pfs^{-\text{GKM}}}$ runs $Setup_{\text{mID}^{-\text{KEM}}}(k, N)$ to obtain $PK^{\text{mID}^{-\text{KEM}}}$. He constructs $PK = \langle PK^{\text{mID}^{-\text{KEM}}}, F, \text{mID} - \text{KEM} \rangle$ and gives it to $A^{pfs^{-\text{GKM}}}$. He also picks a random seed $g$ from $\mathbb{Z}_p^*$ and sets the master secret key $MSK$ to $(MSK_{\text{mID}^{-\text{KEM}}}, g)$.

2) Query Phase 1 — $A^{pfs^{-\text{GKM}}}$ is allowed to query the oracles $O_{\text{Join}}, O_{\text{Leave}}, O_{\text{Ciphertext}}$ and $O_{\text{Decrypt}}$.

3) Corrupt Phase — $A^{pfs^{-\text{GKM}}}$ chooses $ID_{i, s}$, an identity which he wants to corrupt and makes the query $O_{\text{Corrupt}}(ID_{i, s})$ at time $t_{\text{Corrupt}}$ (which is the choice of $A^{pfs^{-\text{GKM}}}$).

4) Query Phase 2 — $A^{pfs^{-\text{GKM}}}$ can query the oracles as in Query Phase 1.

5) Challenge Phase — $A^{pfs^{-\text{GKM}}}$ issues one challenge query to its challenger $A^{\text{pfs}^{-\text{GKM}}}$, specifying a time $t_{\text{Challenge}}$ (which is the choice of $A^{pfs^{-\text{GKM}}}$), subject to the restrictions that $ID_{i, s} \in S_{\text{Challenge}}$ and $t_{\text{Challenge}} \leq t_{\text{Join}}(ID_{i, s})$. $A^{\text{pfs}^{-\text{GKM}}}$ does the following before responding with the challenge.

- Retrieve the set $S_{\text{Challenge}}$ from the list $L_s$.
- Obtain a $(\text{Hdr}_{\text{mID}^{-\text{KEM}}, \text{DEK}})$ pair by running $Encapsulate_{\text{mID}^{-\text{KEM}}}(S_{\text{Challenge}}, PK^{\text{mID}^{-\text{KEM}}})$.
- Compute $\text{Hdr}_{\text{GKM}}^s \leftarrow \langle \text{Hdr}_{\text{GKM}, \text{b}}^s \oplus g_{\text{Challenge}}, r_{\text{Challenge}} \rangle$.15
- Randomly select a bit $b \in \{0, 1\}$ and set $K_b = \text{DEK}$ and $K_{1-b}$ to a random element from the key space $K$.

Now, $C^{pfs^{-\text{GKM}}}$ returns $(\text{Hdr}_{\text{GKM}, \text{b}}^s, K_0, K_1)$ as the challenge to $A^{pfs^{-\text{GKM}}}$.

6) Guess Phase — $A^{pfs^{-\text{GKM}}}$ outputs a bit $b' \in \{0, 1\}$ as its guess.

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

$$Adv_{GKM}^{pfs-CCA} < Adv_{F}^{inv}$$

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

**Theorem 4.** $GKM$ is cr-CCA secure if $mID - KEM$ is at least CCA secure.

**Proof.** Here, we describe how the adversary $A^{mID - KEM}$ on one side acts as the challenger $C^{cr-GKM}$ who interacts with $A^{cr-GKM}$, while simultaneously interacting with $A^{mID - KEM}$ 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. As usual, we present the description from the point of view of the game $G^{cr-GKM}$.

1) **Setup Phase** — The challenger $C^{mID - KEM}$ runs $Setup_{mID - KEM}(k, N)$ to obtain $PK^{mID - KEM}$ and gives it to $A^{mID - KEM}$, who constructs $PK = \langle PK^{mID - KEM}, F, mID - KEM \rangle$ and gives it to $A^{cr-GKM}$. He also picks a random seed $g$ from $Z_p^*$ and sets the master secret key $MSK$ to $\langle MSK_{mID - KEM}, g \rangle$.

2) **Query Phase** — $A^{cr-GKM}$ is allowed to query the oracles $O_{Join}, O_{Leave}, O_{Ciphertext}$ and $O_{Decrypt}$.

3) **Challenge Phase** — $A^{cr-GKM}$ issues one challenge query to its challenger $A^{mID - KEM}$ at time $t_{Challenge}$ (which is the choice of $A^{cr-GKM}$). Now, $A^{mID - KEM}$ does the following before responding with the challenge.

- Retrieve the set $S_{t_{Challenge}}$ from the list $L_s$, and $g_{t_{Leave}(ID)}$ from the list $L_g$, for all $ID \notin S_{t_{Challenge}}$.

- For each identity $ID_i \notin S_{t_{Challenge}}$, issue the query $O_{Extract}(ID_i)$ to obtain $S_{ID_i}$, and return $SK_{ID} = \langle S_{ID_i}, g_{t_{Leave}(ID_i)} \rangle$.

- Issue a challenge query, specifying the set $S_{t_{Challenge}}$, to the challenger $C^{mID - KEM}$.

- Receive the challenge $(H_{dr}^{mID - KEM}, K_0, K_1)$.

- Compute $H_{dr}^{mID - KEM} = \langle H_{dr}^{mID - KEM}, S_{t_{Challenge}} \rangle$.

- $A^{mID - KEM}$ returns $(H_{dr}^{mID - KEM}, K_0, K_1)$ as the challenge to $A^{cr-GKM}$.

4) **Guess Phase** — $A^{cr-GKM}$ outputs a bit $b' \in \{0, 1\}$ as its guess. $A^{mID - KEM}$ passes on $b'$ as its guess to $C^{mID - KEM}$.

It is easy to see that the advantage of $A^{cr-GKM}$ in breaking the collusion resistance of $GKM$ is the same as that of $A^{mID - KEM}$ in breaking the CCA security of $mID - KEM$.

$$Adv_{GKM}^{cr-CCA} = Adv_{mID - KEM}^{CCA} = \frac{1}{2}$$

This means that if there exists no adversary $A^{mID - KEM}$ who can break the CCA security of $mID - KEM$ with non-negligible advantage, then there cannot be any adversary $A^{cr-GKM}$ who can break the collusion resistance of $GKM$ with non-negligible probability.

VIII. An Illustration of the Generic Conversion to Centralized GKM

In this section, we present an example of the generalized transformation to centralized GKM that was presented in Section 7. We first recall the efficient mID-KEM that was proposed by Delerable [?] in 2007, and then construct the most efficient centralized GKM scheme proposed till date using this mID-KEM. This is the first efficient and scalable GKM scheme to achieve a constant size rekeying message.

### A. Delerable’s mID-KEM

**Setup**$(k, N)$ — Given the security parameter $k$ and the maximum number of receivers $N$, a bilinear map group system $B = \langle g, G_1, G_2, G_T, \hat{e}(-, -) \rangle$ is constructed such that $|p| = k$. Also, two generators $f \in G_1$ and $h \in G_2$ and a secret value $\gamma \in Z_p^*$ are randomly selected. Choose a cryptographic hash function $H : \{0, 1\}^* \rightarrow Z_p^*$. The master key is defined as $MSK = (f, \gamma)$. The public key is $PK = (w, v, h, \gamma, \ldots, \gamma^N)$ where $\omega = f^\gamma$, and $v = \hat{e}(f, h)$.

**Extract**$(MSK, ID_i, PK)$ — Given $MSK = (f, \gamma)$, the public key $PK$ and the identity $ID_i$, this algorithm outputs $SK_{ID_i} = f^\gamma$.

**Encapsulate**$(S, PK)$ — Assume for notational simplicity that $S = \{ID_i\}_{i=1}^s$, with $s \leq N$. Given $PK$, this algorithm randomly picks $r \in Z_p^*$ and computes $H_{dr} = (C_1, C_2)$ and $DEK \in K$ where

$$C_1 = \omega^{-a}, \quad C_2 = h^\alpha \prod_{i=1}^{s} (\gamma + H(ID_i)), \quad DEK = \nu^a$$

and outputs $(H_{dr}, DEK)$.

**Decapsulate**$(S, ID_i, SK_{ID_i}, H_{dr}, PK)$ — In order to retrieve the DEK encapsulated in the header $H_{dr} = (C_1, C_2)$, the user with identity $ID_i$ and the corresponding private key $SK_{ID_i} = f^\gamma$ (with $ID_i \in S$) computes the data encryption key as follows.

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

with

$$p_i, s(\gamma) = \frac{1}{\gamma} \cdot \left(\prod_{j=1, j \neq i}^{\hat{\gamma}} (\gamma + H(ID_j)) - \prod_{j=1, j \neq i}^{s} H(ID_j) \right)$$
Delerablée has shown this scheme to be secure against static chosen plaintext attacks. Because of this, the centralized GKM scheme that we derive from this mID-KEM will also enjoy only identity-static CPA security. However, our GKM scheme will be secure against time-adaptive attacks. As noted in [2], her mID-KEM can be converted to one that is secure against chosen ciphertext attacks by using the result of [2], on using which the resultant GKM scheme would also be CCA secure.

B. The Centralized GKM Scheme from Delerablée

Now, we present, without much ado, the identity-static, time-adaptive CPA secure centralized GKM scheme that is constructed out of Delerablée’s mID-KEM. While describing this GKM scheme, we follow the general framework that we presented in Section ??.

Setup(k, N, Sinit)

- **Input.** Take as input the security parameter k, the maximum number of group members N, the set Sinit of the identities of initial group members.
- A bilinear map group system $\mathcal{B} = (p, G_1, G_2, G_T, e(\cdot, \cdot))$ is constructed such that $|p| = k$.
- Two generators $f \in G_1$ and $h \in G_2$ and a secret value $\gamma \in \mathbb{Z}_p^*$ are randomly selected.
- Choose a cryptographic hash function $H : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$ and a one-way function $F : \mathbb{Z}_p^* \rightarrow \mathbb{Z}_p^*$.
- 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 $PK = (\omega, v, h, h^\gamma, \ldots, h^{\gamma N}, H, F)$ is the public key where $\omega = f^\gamma$, and $v = e(f, h)$.
- Choose a data encryption key $DEK$ at random from the key space $\mathcal{K}$.
- Compute $SK_i = (f \mapsto f^\gamma, g)$ for all $ID_i \in Sinit$ and securely send these keys to the corresponding members. Also send the initial $DEK$ securely to these members.

Rekey(S, PK)

- **Input.** Take as input the set $S$ 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 $F$ as $g \leftarrow r \cdot F(g)$.
- Compute
  
  $$C_1 = \omega^{-\alpha}, \quad C_2 = h^{\alpha \prod_{j=i}^{N} (\gamma + H(ID_j))}, \quad DEK = v^\alpha$$

- 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 $g \leftarrow r \cdot F(g)$, and securely erases any copies of older $g$ values.
- Every group member with identity $ID_i$ will retrieve $Hdr = C_0 \oplus g$, and compute
  
  $$DEK = (\hat{e}(C_1, h^{P_i,S(\gamma)}) \cdot \hat{e}(sk_{ID_i}, C_2))^{\prod_{j=1, j \neq i}^{N} H(ID_j)}$$

with

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

to obtain $DEK$.

Join(IDi, S, PK)

- **Input.** Take as input the identity $ID_i$ of a member who wishes to join the group, the set $S$ 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 $SK_i = (f \mapsto f^\gamma, g)$ and securely send it to the joining member.
- Update the set of identities of current group members as $S \leftarrow S \cup \{ID_i\}$.
- Run Rekey($S$, $PK$).

IX. Conclusion

In this paper, we have identified the lack of a formal framework and security model for Group Key Management. To fill this gap, 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 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. Future work can now concentrate on the relatively simpler problem of constructing mID-KEMs which are efficient and secure against adaptive attacks. 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. The generic conversion from mID-KEM to GKM would be complete if the security-inheritance of the resulting GKM goes further to CCA2. Another open problem is to investigate if mKEMs (that are not ID-based) can also be converted to GKM schemes. Decentralized schemes come in handy when the system becomes huge and there is pressure on the central authority who manages the entire group. While some form of formal framework and security models do exist for decentralized GKM in the form of security models for Dynamic Group Key Exchange (which is a larger class of protocols) in [2], it is nevertheless worthwhile to investigate if a simpler, more personalized security model for decentralized GKM can be derived by extending that of centralized GKM.
REFERENCES


