Multi-Stage Group Key Distribution and PAKEs: Securing Zoom Groups against Malicious Servers without New Security Elements

***Abstract*—Video conferencing apps like Zoom have hundreds of millions of daily users, making them a high-value target for surveillance and subversion. While such apps claim to achieve some forms of end-to-end encryption, they usually assume an incorruptible server that is able to identify and authenticate all the parties in a meeting. Concretely this means that, e.g., even when using the "end-to-end encrypted" setting, malicious Zoom servers could eavesdrop or impersonate in arbitrary groups.**

In this work, we show how security against malicious servers can be improved by changing the way in which such protocols use passwords (known as *passcodes* in Zoom) and integrating a password-authenticated key exchange (PAKE) protocol.

To formally prove that our approach achieves its goals, we formalize a class of cryptographic protocols suitable for this setting, and define a basic security notion for them, in which group security can be achieved assuming the server is trusted to correctly authorize the group members. We prove that Zoom indeed meets this notion. We then propose a stronger security notion that can provide security against malicious servers, and propose a transformation that can achieve this notion. We show how we can apply our transformation to Zoom to provably achieve stronger security against malicious servers, notably without introducing new security elements.

1. Introduction

Video conferencing apps such as Zoom are globally used by hundreds of millions of users on a daily basis [1], and aim to use cryptographic protocols to achieve some forms of end-to-end encryption. While there have been many recent advancements in highly-secure messaging protocols such as Signal, their core protocols are typically not suitable for real-time group applications, such as video conferencing, which have have fundamentally different requirements involving real-time constraints, robustness, and usability.

In practice, real-time group protocols used in real-world widely deployed applications such as Zoom incorporate design choices based on real-time requirements that include assigning a group leader role to some participants, relying on key distribution instead of key agreement, and using simplified key evolution mechanisms. These design choices improve features like robustness and usability, and enable real-time communication, at the cost of lower security guarantees compared to some state-of-the-art secure messaging protocols.

Nevertheless, many real-time group protocols explicitly claim that they can provide a form of end-to-end security. For example, in Zoom's case, the 'end-to-end security' option in the app's settings is explained as "Encryption key stored on your local device. No one else can obtain your encryption key, not even Zoom." However, it was shown that a malicious Zoom server can eavesdrop or impersonate in groups [2], [3]. The underlying reason is that in practice, a Zoom server acts as the *sole root of trust* for the authenticity of users' public keys and messages, and implicitly to those that are used to distribute group-specific public keys, which in turn are used by the leader to distribute the group key. Thus, if the Zoom server replaces some of these public keys, it can in fact learn your encryption key.

In this work, we propose a transformation to improve the security of a class of protocols against malicious servers, without introducing new security elements or even new message flows. We achieve this by reworking the way in which such protocols use passwords (known as *passcodes* in Zoom). In the Zoom protocol, the server inherently needs to know the group password and uses it to enforce access control for the group. We propose a modification in which the server no longer knows the password, which is distributed only to the group members and is used by them directly for access control. These passwords can be distributed as one would have done currently (e-mail, messaging app, phone, calendar appointment).

In our new threat model, we can no longer rely on the server providing a priori secure channels between group members. Instead, we employ password-authenticated key exchange to prevent offline guessing attacks on the protocol. We then formally prove that the transformed Zoom protocol achieves a strong form of security even in the presence of malicious servers.

At the technical level, we develop syntax and security notions for multi-stage group key distribution protocols, of which Zoom can be seen as an instance. We show under which assumptions the Zoom protocol (version 4.0) can be proven secure in our multi-stage group key distribution model, and how the known attacks fit into this picture. We design a generic transformation that can turn any protocol in our class into a more secure protocol, for which we define a stronger security notion. We show that our transformation preserves the original security property, and provides security against malicious servers. We apply our transformation to the Zoom protocol, resulting in the ZoomPAKE protocol. We also show how the ZoomPAKE protocol prevents the attacks possible on the Zoom protocol.

Our main contributions are:

- We develop a solution to improve the security of Zoom-like apps against malicious servers, without introducing new security elements. The core observation is that Zoom already uses group-specific passwords, but they are by design known to the server. By leveraging techniques from password-authenticated key exchange, we can get rid of the reliance on the server for trusted channels.
- To formally prove the security of our solution, we need to develop substantial machinery. We propose a formal model and syntax of multi-stage group key distribution protocols, called mGKD, of which Zoom can be seen as an instance. For such protocols, we develop a basic security notion Sec-mGKD-pki, which assumes the server did not interfere with the public keys of a group's participants, and prove that Zoom meets this notion. We show how real-world attacks manifest in this basic notion and notably how malicious zoom servers can manipulate groups.
- We formally prove that our transformation turns a protocol that is Sec-mGKD-pki secure into one that is also secure in a model that makes no assumptions on the server but only on the password, which we call Sec-mGKD-pw.
- We show how to efficiently apply our transformation to the Zoom version 4.0 protocol to obtain the ZoomPAKE protocol, in which the server no longer knows the password, and groups are protected against malicious servers.

Outline We discuss related work in Section 2 and notation in Section 3. In Section 4 we present our syntax for multi-stage group key distribution (mGKD) protocols and three security notions: basic Sec-mGKD-pki security, full end-to-end Sec-mGKD-pw security, and the combined Sec-mGKD-pw+ security. We show in Section 5 that the Zoom library can be modeled as a mGKD protocol and provably satisfies the basic Sec-mGKD-pki security, and show how impersonation attacks prevent it from satisfying the stronger Sec-mGKD-pw notion. In Section 6, we develop a generic transformation on any Sec-mGKD-pki secure mGKD protocol to achieve Sec-mGKD-pw and Sec-mGKD-pw+ security and apply it to Zoom.

We provide our customized notions w-PAKE security for PAKE and frob security for AEAD and proof sketches of all our theorems in appendix. We provide all details and full proofs in the long version [4].

2. Related Work

While there is a lot of adjacent related work that we will mention below, it turns out that there is surprisingly little directly related work in analysis of real-time group protocols. A number of surveys [5]–[10] examine numerous historical designs for secure group key establishment in different application scenarios. We identify three categories:

- centralized group key management/distribution protocols, where each group has a single trusted authority for group key generation and distribution.
- (continuous) group key agreement and distributed key management protocols, where every party in a group contributes to the group key generation and distribution.
- multi-factor key agreement and password-authenticated key exchange protocols, where the group key generation and distribution relies only on a secret that is used for authorization to a group.

We review each of these categories below.

2.1. Centralized Group Key Management Protocols

A centralized group key management (CGKM) protocol starts every group with a trusted authority, often referred to as the "Key Distribution Center" (KDC). The KDC is responsible for controlling for the whole group, e.g., member authentication, access control, and group key generation and distribution.

One of the first CGKM schemes is [11], [12]. In this approach, the KDC creates a "Group Key Packet" (GKP) for encrypting the communication payload with the help from the first group participant. The KDC sends the GKP to every party that wants to join the group and encrypts the new GKP to all group participants using the old one. To achieve forward secrecy, the KDC has to recreate the group whenever a participant leaves the group. After that, numerous tree-like CGKM constructions [13]–[18] were proposed to reduce computation cost. In these approaches, all trust resides in the KDC, which forms a single point of failure for compromise.

2.2. (Continuous) Group Key Agreement

Two important canonical group key agreement protocols are [19], [20]. Their constructions have a binary tree-like hierarchy, where the keying material of each party is a leaf node at the bottom of the tree and the shared group key is the top node of the tree. Every party can compute the key material on the path from its associated leaf node to the top using Diffie-Hellman Exchange (DHE). However, these designs are inherently synchronous: the initialization of the tree requires all parties to be online.

In the asynchronous secure messaging context, where participants might be offline and group keys need to be evolved, this approach does not work without modification. These drawbacks were lifted by the design in [21], leading to a line of papers in the continuous GKA (CGKA) domain, including [22]–[24] that focus on continuously evolving (ratcheting) group keys after the group establishment.

In general, ratcheting-like protocols such as CGKAs are impractical for real-time group applications, as they have to tolerate high amounts of packet loss and still being able to continue immediately when some packets arrive on time.

2.3. Multi-factor Key Agreement and Password-Authenticated Key Exchange

Multi-factor key agreement protocols often rely on three classes of human authentication factors: (1) something you

know, e.g., passwords, (2) something you have, e.g., secure devices, and (3) something you are, e.g., biometric date. Among them, the password is possibly one of the most convenient means for sharing in practice, as it can be easily sent out-of-band, e.g., via email, in letters, or even in-person.

The human-chosen passwords are often low-entropy rather than uniformly at random. The Password-Authenticated Key Exchange (PAKE) protocols are designed to allow some parties to establish a high-entropy session key with authentication based on a low-entropy shared password without being subject to offline guessing attacks. There are numerous modern and efficient 2-party PAKE constructions in the literature, such as CPace [25], [26] and SPAKE2 [27]–[29]. The are also several known group PAKE protocols [30]–[33]. However, the existing group PAKE protocols always require multiple rounds for the key agreement and is restricted to static groups. Thus, these group PAKE protocols are impractical for the real-time group applications, where the participants can freely join and leave the groups.

2.4. Existing Security Analysis for Zoom

In [2], [3], the authors describe several specific classes of impersonation attacks on end-to-end Zoom (version 2.3.1). First, a malicious meeting participant can impersonate any other participant inside this group, since there is no entity authentication in a group meeting. Second, the Zoom server can replay some messages and impersonate a legitimate user for a meeting. Third, if multiple users share a device, the Zoom server colluding with any user can impersonate any other users on the same device. Moreover, the authors also present a tampering attack based on potential implementation flaws and a Denial-of-Service attack. [2], [3] provide feasible countermeasures for each of above specific attacks. However, more general impersonation attacks by a malicious server are not considered in [2], [3].

The "accepted papers" page of Eurocrypt 2023 lists [34]. According to the title and abstract, [34] formally analyzes the security of the end-to-end Zoom protocol, focuses on various liveness properties, including dynamic group joining and leaving and the update of group roster and keys, but does not address assumptions on the server or PKI. However, at the time of this submission, [34] is not yet publicly available (neither on IACR eprint, arxiv, nor the author's webpages), and thus we cannot comment on its details.

3. Preliminaries

We use the following notational conventions. We write ϵ to denote the empty string, and write $x \parallel y$ to denote the concatenation of strings x and y. We assume the existence of a reserved error symbol $\perp$. We write $y \leftarrow x$ for deterministic assignment or computation. We write $y \xleftarrow{\$} D$ for randomized sampling and $y \xleftarrow{\$} f(x)$ for non-deterministic computation. We write $[\![\cdot]\!]$ for a boolean statement that is either true (denoted by 1) or false (denoted by 0).

Within algorithms, we write "**require** C" to denote that C is a requirement: if the condition C is not met, the algorithm undoes the execution and outputs $\perp$.

Moreover, we use several abbreviations. To avoid duplicating some longer variable names, we use some update-based variants of common operators: For a number x, we write x++ as a shorthand for $x \leftarrow x + 1$. For a set S and an element x, we write $S \xleftarrow{+} x$ for $S \leftarrow S \cup \{x\}$, and $S \xleftarrow{-} x$ for $S \leftarrow S \setminus \{x\}$. We write PPT as an abbreviation of Probabilistic Polynomial Time.

4. multi-stage Group Key Distribution Protocols

In this section, we define our syntax for multi-stage Group Key Distribution mGKD protocols. Our class of mGKD protocols covers behaviors of a party for using real-time group services, such as long-term identity information generation, joining groups, group key rotation, and leaving groups. Then, we propose three security models that capture distinct security guarantees for real-time group services.

4.1. mGKD Definition

mGKD protocols are stateful interactive group communication protocols executed by a set of parties $\mathcal{P}$. Each party P must be uniquely identified by an identifier id_P. Each group is uniquely identified by an identifier gid. In each group, one party performs the leader role; all other group members perform the participant role. The role of each party in different groups might be distinct: a party can be leader in one group and participant in others. In practice, the leader is typically the so-called *host* that initiated the group. In this paper, we assume that each group gid is associated with a unique leader and that the group's leader stays in the group for its entire duration. While one can in theory implement changing leaders by starting a new group, we leave the modeling and efficient implementation of multiple and changing leaders to future work.

Definition 1. *A multi-stage group key distribution protocol* $\mathsf{mGKD} = (\mathsf{SignUp}, \mathsf{Schedule}, \mathsf{Register}, \mathsf{Join}, \mathsf{Leave}, \mathsf{KeyRotat})$ *consists of the following algorithms:*

Sign Up: $m_{\mathsf{SignUp}} \xleftarrow{\$} \mathsf{SignUp}(P)$ *allows a (stateful) party P to initialize a long-term identity information for signing up. The private portion is locally stored. The public portion is output as an outgoing sign-up message* m_{SignUp}.

Group Schedule: $m^{\mathsf{gid}}_{\mathsf{GSch}} \xleftarrow{\$} \mathsf{Schedule}(P, \mathsf{gid}, gs)$ *allows a (stateful) party P to take the role of the leader for scheduling a group* gid *using a group secret gs. The output is an outgoing group schedule message $m^{\mathsf{gid}}_{\mathsf{GSch}}$ for the server. The group secret gs is expected to be sent to authorized participants over secure out-of-band channels.*

Register: $\mathsf{Register} = (\mathsf{Register\text{-}L}, \mathsf{Register\text{-}P})$ *consists of two sub-algorithms depending on the role of the caller:*

- $m' \xleftarrow{\$} \mathsf{Register\text{-}L}(P, \mathsf{gid}, gs, m)$ *(resp.* $m' \xleftarrow{\$} \mathsf{Register\text{-}P}(P, \mathsf{gid}, gs, m)$*) allows a (stateful) party P to register for a group* gid *as leader (resp. participant)*

using a group secret gs and an incoming message m followed by group initialization. The output is an outgoing message m'.

Participant Join: $\mathsf{Join} = (\mathsf{Join\text{-}L}, \mathsf{Join\text{-}P})$ *allows a participant to interact with a leader for joining a group. This interactive phase consists of two sub-algorithms depending on the role of the caller:*

- $m' \xleftarrow{\$} \mathsf{Join\text{-}L}(P, \mathsf{id}_{P'}, \mathsf{gid}, gs, m)$ *(resp.* $m' \xleftarrow{\$} \mathsf{Join\text{-}P}(P, \mathsf{id}_{P'}, \mathsf{gid}, gs, m)$*) allows a leader P (resp. a participant P) of the group* gid *to input a group secret gs and an incoming message m and to output an outgoing message m' for the participant (resp. the leader) P'.*

Member Leave: $\mathsf{Leave} = (\mathsf{Leave\text{-}L}, \mathsf{Leave\text{-}P})$ *consists of two sub-algorithms depending on the role of the caller:*

- $\mathsf{Leave\text{-}L}(P, \mathsf{gid}, \mathsf{id}_{P'})$ *(resp.* $\mathsf{Leave\text{-}P}(P, \mathsf{gid}, \mathsf{id}_{P'})$*) allows the leader (resp. a participant) P of the group* gid *to react to a party P''s leaving. If $\mathsf{id}_P = \mathsf{id}_{P'}$, the leader (resp. the participant) P leaves the group* gid *and erases the corresponding state information.*

Key Rotation: $\mathsf{KeyRotat} = (\mathsf{KeyRotat\text{-}L}, \mathsf{KeyRotat\text{-}P})$ *consists of two sub-algorithms depending on the role of the caller:*

- $m_{\mathsf{KRot}} \xleftarrow{\$} \mathsf{KeyRotat\text{-}L}(P, \mathsf{gid}, m)$ *allows the leader P of the group* gid *to input an incoming message m and to locally update the group key. The output is an outgoing message m_{KRot} that enables all participants of the same group* gid *to update group keys correspondingly.*
- $m_{\mathsf{KRot}} \xleftarrow{\$} \mathsf{KeyRotat\text{-}P}(P, \mathsf{gid}, m)$ *allows a participant P of the group* gid *to input an incoming message m and to locally update the group key. The output is an (optionally empty) outgoing message m_{KRot}.*

We assume all incoming and outgoing messages of an mGKD protocol are publicly accessible; we leave this implicit as the concrete mechanisms can differ substantially between protocols, but could for example be a PKI or a "bulletin board" on the server. In contrast, the input group secret gs of the Schedule algorithm is expected to be chosen by the leader and be sent to authorized parties for joining the group over secure out-of-band channels. Before a party P joins a group gid, P has to register for this group, no matter whether P has previously joined the group gid and left. The Member Leave phase enables every party P to react to a participant P' leaving the group, notably, without any additional incoming message. This captures the scenario where the server might notify group members that a participant has left the group without sending any leave request due to unexpected network disconnection. The KeyRotat algorithm enables every party to update their group key, the execution frequency of which can be decided in advance, in a regular schedule and/or when a party joins or leaves the group.

To model concurrent or sequential groups of a party P, let π_P^{gid} denote party P's session with respect to the short-term group gid. In addition, each party P has an associated long-term state st_P that is shared by all of P's sessions.

Definition 2. *In a* mGKD *protocol, each party P has the following state variables. The long-term state variables are initialized during the Sign Up phase:*

- $\mathsf{st}_P.\mathsf{id}$*: the associated and unique identifier of the party P. In this paper, we assume it equal to* id_P.
- $\mathsf{st}_P.isk$*: the identity secret key of the party P.*
- $\mathsf{st}_P.ipk$*: the identity public key of the party P.*

The short-term per-group state variables are initialized during the Register phase:

- $\pi_P^{\mathsf{gid}}.sk$*: the group-specific secret key of the party P for joining the group* gid.
- $\pi_P^{\mathsf{gid}}.pk$*: the group-specific public key of the party P for joining the group* gid.
- $\pi_P^{\mathsf{gid}}.gk$*: the current group key used by party P, which is supposed to be shared by all parties in the group* gid*. This variable is initialized with $\perp$.*
- $\pi_P^{\mathsf{gid}}.\mathsf{gkid}$*: the index of the current group key of the party P in the group* gid*. This variable is initialized with* 0.
- $\pi_P^{\mathsf{gid}}.GP$*: The set of all parties in the group* gid*. This variable is initialized with the empty set $\emptyset$.*
- $\pi_P^{\mathsf{gid}}.\mathsf{status} \in \{\perp, \mathsf{registered}, \mathsf{joined}\}$*: the status that indicates whether the party P has initialized the state for (but not yet registered for), or registered for (but not yet joined), or joined the group* gid*. This variable is $\perp$ by default.*

Definition 3 (Correctness)**.** *Consider any group* gid *with an associated leader P, any sequence of parties $\{P^i\}_i$, and a sequence of executions that includes the following (perhaps not consecutive) algorithms: i) a Sign Up of the leader P or a participant P^i for any i, ii) a Group Schedule of the group* gid *and the leader P, iii) a Register of the leader P or a participant P^i for any i to the group* gid*, iv) a successful Participant Join between the leader P and a participant P^i for any i, v) a Leaving of participant P^i for any i to all other parties in the group* gid*, vi) a successful Key Rotation for the leader P and every participant P^i in the group* gid*, i.e.,* $\pi_{P^i}^{\mathsf{gid}}.\mathsf{status} = \mathsf{joined}$*. Correctness requires that* $\pi_P^{\mathsf{gid}}.\mathsf{gkid} = \pi_{P^i}^{\mathsf{gid}}.\mathsf{gkid}$ *and* $\pi_P^{\mathsf{gid}}.gk = \pi_{P^i}^{\mathsf{gid}}.gk$ *for any P^i with* $\pi_{P^i}^{\mathsf{gid}}.\mathsf{status} = \mathsf{joined}$ *at any time.*

4.2. A Generic Security Model

We next define a generic Sec-mGKD-X security model. By presenting different instantiations of the freshness conditions that the attacker must obey and of the winning conditions that the attacker must pursue, we then introduce three distinct concrete models for $\mathsf{X} \in \{\mathsf{pki}, \mathsf{pw}, \mathsf{pw+}\}$ in Section 4.3, Section 4.4, and Section 4.5.

Trust Model. We assume that all parties' sampled randomness is independent, uniform, and unpredictable. For simplicity, we assume every leader samples group secret from a same distribution $\mathcal{D}$ (according to the underlying protocol). We assume that every party joins every group at most once. We assume that every leader stays in the corresponding group for the entire duration and leaves only when all other participants have left. This means, once the leader leaves the corresponding group, this group is immediately marked as "invalid" and no party is allowed to register for or join this group anymore. We assume that every party can send

register request for every group at most once. Our model assumes a single shared group key for communication. Thus, impersonation attacks between parties inside the group is out of scope of this work.

Threat Model. We allow the attacker to have full control over the network and can eavesdrop, drop, and insert messages during all phases. We allow the attacker to corrupt the long-term state st_P for any participant P to capture the real-world scenarios where the hardware devices might be stolen. Moreover, we allow attackers to compromise the short-term per-group state π_P^{gid} for any participant P after joining any group gid to capture attacks during the ongoing communications. We also allow attackers to leak arbitrary group keys (to analyze the impact on the security of previous and future group keys). We allow attackers to reveal the group secret for any group to capture the real-world scenarios where the the out-of-band channels might be vulnerable.

Security Experiment. The security experiment is conducted between a challenger $\mathcal{C}$ and an attacker $\mathcal{A}$ against a mGKD protocol Π. At the beginning of the experiment, the challenger $\mathcal{C}$ samples a random challenge bit $\mathsf{b} \in \{0,1\}$. During the experiment, $\mathcal{C}$ produces two sequences of variables $\{GP^{(\mathsf{gid},\mathsf{gkid})}\}_{\mathsf{gid},\mathsf{gkid}}$ and $\{gk_P^{(\mathsf{gid},\mathsf{gkid})}\}_{\mathsf{gid},\mathsf{gkid},P}$. The variable $GP^{(\mathsf{gid},\mathsf{gkid})}$ aims to record the identifier of every party that is expected to know the gkid-th group key in the group gid from the leader's view. The challenger $\mathcal{C}$ monitors the states of the leader P of any group gid. Whenever $\pi_P^{\mathsf{gid}}.GP$ changes, $\mathcal{C}$ records $GP^{(\mathsf{gid},\pi_P^{\mathsf{gid}}.\mathsf{gkid})} \leftarrow GP^{(\mathsf{gid},\pi_P^{\mathsf{gid}}.\mathsf{gkid})} \cup \pi_P^{\mathsf{gid}}.GP$. The variable $gk_P^{(\mathsf{gid},\mathsf{gkid})}$ records the gkid-th group key derived by any party P in the group gid. Whenever $\pi_P^{\mathsf{gid}}.\mathsf{gkid}$ changes for any party P and group gid during the experiment, $\mathcal{C}$ stores the new group key $gk_P^{(\mathsf{gid},\pi_P^{\mathsf{gid}}.\mathsf{gkid})} \leftarrow \pi_P^{\mathsf{gid}}.gk$. The attacker $\mathcal{A}$ can interact with $\mathcal{C}$ by querying the following oracles, where $\mathcal{C}$ responds according to Π. To simplify the explanation, we partition the oracles into categorizes.

Oracle Category 1: Setup of groups and parties. This category includes a NEWPARTY oracle that simulates the Sign Up phase of a party, a NEWGROUP that simulates the Group Schedule phase of a group, and a AUTHORIZE oracle that simulates the group secret transmission from the leader to the authorized participants over out-of-band channels.

- NEWPARTY(id_P): This oracle can be queried at most once on each input. The challenger $\mathcal{C}$ initializes a state st_P by setting $\mathsf{st}_P.\mathsf{id} \leftarrow \mathsf{id}_P$. Then, $\mathcal{C}$ runs $m_{\mathsf{SignUp}}^P \xleftarrow{\$} \mathsf{SignUp}(\mathsf{st}_P)$ and followed by forwarding the sign-up message m_{SignUp}^P to $\mathcal{A}$. The party P is marked as "created".
- NEWGROUP($\mathsf{id}_P, \mathsf{gid}$): This oracle can be invoked at most once for each gid. The input party P must be marked as "created". The challenger $\mathcal{C}$ samples the secret of the group gid by $gs^{\mathsf{gid}} \xleftarrow{\$} \mathcal{D}$ and runs $m_{\mathsf{GSch}}^{\mathsf{gid}} \xleftarrow{\$} \mathsf{Schedule}(P, \mathsf{gid}, gs^{\mathsf{gid}})$ for an associated outgoing message $m_{\mathsf{GSch}}^{\mathsf{gid}}$. Then, $\mathcal{C}$ marks the group gid as "created" and "valid" and marks P as the leader of the group gid and "authorized". Finally, $\mathcal{C}$ returns $m_{\mathsf{GSch}}^{\mathsf{gid}}$ to $\mathcal{A}$.
- AUTHORIZE($\mathsf{gid}, \mathsf{id}_P$): The group gid must be marked as both created and valid and the party P must be marked as created and have not registered for the group gid, i.e., $\pi_P^{\mathsf{gid}}.\mathsf{status} = \perp$. The challenger $\mathcal{C}$ marks P as authorized for the group gid.

Oracle Category 2: Register phase. This category includes a REGISTERAUTH oracle, which simulates that an authorized party registers for a group using honest group secret, and a REGISTERINJECT oracle, which simulates that an unauthorized (malicious) party registers for a group with an input using some chosen group secret.

- REGISTERAUTH($\mathsf{id}_P, \mathsf{gid}, m$): This oracle can be queried at most once for each tuple $(\mathsf{id}_P, \mathsf{gid})$. The group gid must be marked as both created and valid and the party P must be authorized for the group gid. The challenger $\mathcal{C}$ runs Register-L$(P, \mathsf{gid}, gs^{\mathsf{gid}}, m)$ if P is the leader of the group gid and Register-P$(P, \mathsf{gid}, gs^{\mathsf{gid}}, m)$ otherwise. In both cases, $\mathcal{C}$ forwards the output message m' to $\mathcal{A}$.
- REGISTERINJECT($\mathsf{id}_P, \mathsf{gid}, gs, m$): This oracle can be queried at most once for each tuple $(\mathsf{id}_P, \mathsf{gid})$. The group gid must be marked as both created and valid and the party P must be unauthorized for the group gid. The challenger $\mathcal{C}$ runs Register-P(P, gid, gs, m) and forwards the output message m' to $\mathcal{A}$.

Oracle Category 3: Participant Join phase. This category includes a SENDJOINAUTH oracle, which simulates that an authorized party (as either leader or participant) sends messages to another party (either authorized or unauthorized) during the Participant Join phase in the group, and a SENDJOININJECT oracle, which simulates that an unauthorized (malicious) participant sends messages (to the leader) during the Participants Join phase in the group.

- SENDJOINAUTH($\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, m$): The challenger $\mathcal{C}$ first checks
  - whether gid is marked as created and valid,
  - whether P is authorized for this group gid,
  - whether both parties P and P' have been created and registered for this group,
  - whether either P or P' is the leader of the group gid,
  - whether the leader of the group, either P or P', has joined the group, and that the other party hasn't joined the group yet.

  If any of the check fails, $\mathcal{C}$ directly returns $\perp$ to $\mathcal{A}$. Otherwise, $\mathcal{C}$ runs Join-L$(P, \mathsf{id}_{P'}, \mathsf{gid}, gs^{\mathsf{gid}}, m)$ if P is the leader of the group gid or Join-P$(P, \mathsf{id}_{P'}, \mathsf{gid}, gs^{\mathsf{gid}}, m)$ if P is a participant. Then, the output message m' of either Join-L or Join-P is returned to $\mathcal{A}$.
- SENDJOININJECT($\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, gs, m$): The challenger $\mathcal{C}$ first checks
  - whether gid is marked as created and valid,
  - whether P is unauthorized for the group gid,
  - whether P' is the leader of the group gid,
  - whether the participant P has been created and registered for this group gid but has not joined the group yet, and
  - whether the leader P' has joined the group gid.

  If any of the check fails, $\mathcal{C}$ directly returns $\perp$ to $\mathcal{A}$. Otherwise, $\mathcal{C}$ runs Join-P$(P, \mathsf{id}_{P'}, \mathsf{gid}, gs, m)$ and returns

the output message m' of Join-P to $\mathcal{A}$.

Oracle Category 4: Member Leave phase. This category includes a SENDLEAVE oracle, which simulates that a party notices another party leaving the group gid, and an ENDGROUP oracle, which simulates that a leader leaves and ends the group.

- SENDLEAVE($\mathsf{id}_P, \mathsf{gid}, \mathsf{id}_{P'}$): The challenger $\mathcal{C}$ aborts if
  - the gid is not marked as both created and valid, or
  - the leaving party P' is the leader of the group gid, or
  - the party P has not joined the group.

  Otherwise, $\mathcal{C}$ runs Leave-L($P, \mathsf{gid}, \mathsf{id}_{P'}$) if P is the leader of the group gid and Leave-P($P, \mathsf{gid}, \mathsf{id}_{P'}$) otherwise.
- ENDGROUP(gid): The challenger $\mathcal{C}$ first checks
  - whether the group gid is created and valid, and
  - whether the leader P of the group gid is the unique party in his local party list, i.e., $\pi_P^{\mathsf{gid}}.GP = \{\mathsf{id}_P\}$.

  If either check fails, $\mathcal{C}$ aborts. Otherwise, $\mathcal{C}$ runs Leave-L($P, \mathsf{gid}, \mathsf{id}_P$) and marks the group gid as "invalid".

Oracle Category 5: Key Rotation phase. This category includes only one SENDKEYROTAT oracle that simulates the process where a party updates their local group key.

- SENDKEYROTAT($\mathsf{id}_P, \mathsf{gid}, m$): The party P must have joined the group gid. The challenger $\mathcal{C}$ runs KeyRotat-L(P, gid, m) if P is the leader of the group gid and KeyRotat-P(P, gid, m) otherwise. The output message of either KeyRotat-L or KeyRotat-P is returned to $\mathcal{A}$.

Oracle Category 6: Secret information leakage. This category includes four oracles CORRUPT, COMPROMISE, LEAK, and REVEAL, that respectively simulates that the attacker $\mathcal{A}$ knows the long-term state of a party, the short-term per-group of a party for a group, a group key of a party in a group, and the group secret of a group.

- CORRUPT(id_P): The challenger $\mathcal{C}$ first checks whether the party P is created. If the check fails, $\mathcal{C}$ simply returns $\perp$. Otherwise, $\mathcal{C}$ returns st_P to the attacker $\mathcal{A}$ and marks st_P as "corrupted".
- COMPROMISE($\mathsf{id}_P, \mathsf{gid}$): The challenger $\mathcal{C}$ checks whether the party P has joined the group gid, i.e, $\pi_P^{\mathsf{gid}}.\mathsf{status} = \mathsf{joined}$. If the check fails, $\mathcal{C}$ simply returns $\perp$. Otherwise, $\mathcal{C}$ returns π_P^{gid} to the attacker $\mathcal{A}$, followed by marking π_P^{gid} as "compromised" and $gk_P^{(\mathsf{gid},\pi_P^{\mathsf{gid}}.\mathsf{gkid})}$ as "leaked".
- LEAK($\mathsf{id}_P, \mathsf{gid}, \mathsf{gkid}$): The challenger $\mathcal{C}$ checks whether $gk_P^{(\mathsf{gid},\mathsf{gkid})}$ has been set. If $gk_P^{(\mathsf{gid},\mathsf{gkid})} = \perp$, then $\mathcal{C}$ simply returns $\perp$. Otherwise, $\mathcal{C}$ marks $gk_P^{(\mathsf{gid},\mathsf{gkid})}$ as "leaked" and returns $gk_P^{(\mathsf{gid},\mathsf{gkid})}$ to $\mathcal{A}$.
- REVEAL(gid): If the group gid is not created, then the challenger $\mathcal{C}$ simply returns $\perp$. Otherwise, $\mathcal{C}$ marks gid as "revealed" and returns gs^{gid} to $\mathcal{A}$.

Oracle Category 7: Test challenge bit. This category includes only one TEST oracle that returns either a real group key if the challenge bit $\mathsf{b} = 0$, or a random key if $\mathsf{b} = 1$.

- TEST($\mathsf{id}_P, \mathsf{gid}, \mathsf{gkid}$): This oracle can be queried at most once. If the party P is authorized for the group gid and the party P has produced gkid-th group key, i.e., $gk_P^{(\mathsf{gid},\mathsf{gkid})} \neq \perp$, then the challenger $\mathcal{C}$ returns $gk_P^{(\mathsf{gid},\mathsf{gkid})}$ to $\mathcal{A}$ if the challenge bit $\mathsf{b} = 0$, or a random key from the same space if $\mathsf{b} = 1$. Then, $\mathcal{C}$ further marks the party P, the group identifier gid, and the group key index gkid as "tested". Otherwise, $\mathcal{C}$ immediately returns $\perp$.

Advantage Measures. In the end, the attacker $\mathcal{A}$ outputs a bit $\mathsf{b}' \in \{0, 1\}$. Under two freshness conditions $\mathsf{fresh}_{\mathsf{KAuth}}^{\mathsf{Sec\text{-}mGKD\text{-}X}}$ and $\mathsf{fresh}_{\mathsf{KPriv}}^{\mathsf{Sec\text{-}mGKD\text{-}X}}$ that prevent the attacker $\mathcal{A}$ from trivially winning the experiment, we say the attacker $\mathcal{A}$ wins the experiment Sec-mGKD-X against a mGKD protocol Π, if either of the following events is triggered:

1) [Event E_{KAuth}] there exists any group gid with the leader P^{gid}, any party P' that is authorized for the group gid, and any group key index gkid, such that $gk_{P'}^{(\mathsf{gid},\mathsf{gkid})} \neq \perp$ but $gk_{P^{\mathsf{gid}}}^{(\mathsf{gid},\mathsf{gkid})} \neq gk_{P'}^{(\mathsf{gid},\mathsf{gkid})}$, without violating the freshness condition $\mathsf{fresh}_{\mathsf{KAuth}}^{\mathsf{Sec\text{-}mGKD\text{-}X}}(\mathsf{id}_{P'}, \mathsf{gid}, \mathsf{gkid})$.
2) [Event E_{KPriv}] $\mathsf{b} = \mathsf{b}'$ without violating the freshness condition $\mathsf{fresh}_{\mathsf{KPriv}}^{\mathsf{Sec\text{-}mGKD\text{-}X}}(\mathsf{id}_{P'}, \mathsf{gid}, \mathsf{gkid})$, where P', gid, and gkid are respectively the tested party, group identifier, and group key index.

We define $\mathsf{Adv}_{\Pi}^{\mathsf{Sec\text{-}mGKD\text{-}X}}(\mathcal{A})$ as the advantage that $\mathcal{A}$ can win the Sec-mGKD-X experiment against a mGKD protocol Π, namely,

$$\mathsf{Adv}_{\Pi}^{\mathsf{Sec\text{-}mGKD\text{-}X}}(\mathcal{A}) := \max\left(\left|\Pr[E_{\mathsf{KPriv}}] - \frac{1}{2}\right|, \Pr[E_{\mathsf{KAuth}}]\right).$$

Definition 4 (Sec-mGKD-X)**.** *We say that a* mGKD *protocol* Π *is* Sec-mGKD-X*-secure, if the advantage* $\mathsf{Adv}_{\Pi}^{\mathsf{Sec\text{-}mGKD\text{-}X}}(\mathcal{A})$ *is negligible for any* PPT *adversary* $\mathcal{A}$*.*

4.3. The Sec-mGKD-pki Security Model

Our basic security model Sec-mGKD-pki captures the following security guarantees for an authorized party in a group assuming the honest distribution of long-term sign-up message of all parties within this group. Note that this basic model guarantees that only group members can learn the key, and where group membership is determined by the *server*. In reality, and in our model, the server may insert group members that are not authorized by the leader and do not know the passcode.

1) **(Implicit) Group Key Authentication**: If a group member accepts a group key, then the leader must have produced the same group with the same group key index.
2) **Group Key Secrecy**: If a group member accepts a group key, then an attacker cannot derive this key, even if it knows other group keys.
3) **Perfect Forward Secrecy**: An attacker that compromises a party's long-term keys, can not learn the group keys of any group the party was previously in.

Definition 5 (Sec-mGKD-pki Freshness Conditions)**.** *We say the freshness condition* $\mathsf{fresh}_{\mathsf{KAuth}}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}(\mathsf{id}_{P'}, \mathsf{gid}, \mathsf{gkid})$*, where* gid *has a unique leader* P^{gid}*, holds if and only if*

1) the per-group states $\pi_{P^{\mathsf{gid}}}^{\mathsf{gid}}$ *and* $\pi_{P'}^{\mathsf{gid}}$ *are not compromised,*

2) *the long-term states* $\mathsf{st}_{P^{\mathsf{gid}}}$ *and* $\mathsf{st}_{P'}$ *are not corrupted before* P^{gid} *and* P' *joined the group* gid*, and*
3) *the sign-up messages* $m^{P^{\mathsf{gid}}}_{\mathsf{SignUp}}$ *and* $m^{P'}_{\mathsf{SignUp}}$ *of* P^{gid} *and* P' *are honestly delivered to the other.*

We say the freshness condition $\mathsf{fresh}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}_{\mathsf{KPriv}}(\mathsf{id}_{P'}, \mathsf{gid}, \mathsf{gkid})$ *holds if and only if*

1) *the group key* $gk_P^{(\mathsf{gid},\mathsf{gkid})}$ *is not leaked for all parties* P *in the group* gid *with* $\mathsf{id}_P \in GP^{(\mathsf{gid},\mathsf{gkid})}$*,*
2) *the short-term state* π_P^{gid} *is not compromised for all parties* P *in the group* gid *with* $\mathsf{id}_P \in GP^{(\mathsf{gid},\mathsf{gkid})}$*,*
3) *the long-term state* $\mathsf{st}_{P^{\mathsf{gid}}}$ *of the leader* P^{gid} *in the group* gid *is not corrupted before all parties* P *with* $\mathsf{id}_P \in GP^{(\mathsf{gid},\mathsf{gkid})}$ *joined the group* gid*,*
4) *the long-term state* st_P *of all participants* P *in the group* gid *with* $\mathsf{id}_P \in GP^{(\mathsf{gid},\mathsf{gkid})}$ *is not corrupted before* P *joined the group* gid*, and*
5) *the sign-up messages* m^{P}_{SignUp} *of all parties* P *with* $\mathsf{id}_P \in GP^{(\mathsf{gid},\mathsf{gkid})}$ *are honestly distributed within the group* gid*.*

Conclusion. Our Sec-mGKD-pki security model captures all guarantees listed at the beginning of this subsection:

1) **(Implicit) Group Key Authentication**: If authentication does not hold, the attacker $\mathcal{A}$ can win via E_{KAuth}.
2) **Group Key Secrecy**: The attacker is allowed to leak arbitrarily many group keys except for the tested one. If group key secrecy does not hold, $\mathcal{A}$ can win via E_{KPriv}.
3) **Perfect Forward Secrecy**: The attacker is allowed to corrupt the long-term state of the tested party. If perfect forward secrecy does not hold, $\mathcal{A}$ can win via E_{KPriv}.

4.4. The Sec-mGKD-pw Security Model

The basic Sec-mGKD-pki model has two restrictions:

- First, both freshness conditions $\mathsf{fresh}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}_{\mathsf{KPriv}}$ and $\mathsf{fresh}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}_{\mathsf{KAuth}}$ require the honest distribution of the sign-up messages. Since the sign-up messages are distributed by servers or by PKI in practice, this restriction is also known as "trusted PKI" assumption in the related literature. In the full end-to-end setting, i.e., no trusted PKI or server exists, a Sec-mGKD-pki secure mGKD protocol might still suffers from machine-in-the-middle attacks such that the attacker can easily impersonate any participant towards the group leader and impersonate any group leader towards any participant.
- Second, both freshness conditions $\mathsf{fresh}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}_{\mathsf{KPriv}}$ and $\mathsf{fresh}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}_{\mathsf{KAuth}}$ allow the attacker to reveal all group secrets but do not prevent unauthorized parties from knowing the group keys. Thus, this Sec-mGKD-pki model does not capture the security benefit provided by the group secret transmitted over secure out-of-band channels.

The goal of the Sec-mGKD-pw security model is to preserve the guarantees achieved in Sec-mGKD-pki model and to capture the following additional security guarantee, while getting rid of the "trusted PKI" assumption.

4) **(Implicit) Group Member Authentication**: If any party produces a same group key as the leader of a group, then this party must be authorized for this group, as long as the group secret is not revealed.

We replace the "trusted PKI" assumption with the arguably simpler assumption of a shared "secure (short) group secret". We assume that (short) group secrets (such as passwords or pin codes) can be distributed over out-of-band secure channels, e.g., by email, encrypted messaging application (e.g., Signal), or even in person. In fact, a similar passcode mechanism has been widely deployed in real life by many service providers such as Zoom for access control management (see Section 5.1). The only difference is that the current Zoom passcode mechanism hands over passcode and the rule of verifying it to the (possibly) untrusted server, while in our model the group secret is known only to the participants.

Definition 6 (Sec-mGKD-pw Freshness Conditions)**.** *We say the freshness condition* $\mathsf{fresh}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}_{\mathsf{KAuth}}(\mathsf{id}_{P'}, \mathsf{gid}, \mathsf{gkid})$*, where* gid *has a unique leader* P^{gid}*, holds if and only if*

1) *the per-group states* $\pi^{\mathsf{gid}}_{P^{\mathsf{gid}}}$ *and* $\pi^{\mathsf{gid}}_{P'}$ *are not compromised,*
2) *the long-term states* $\mathsf{st}_{P^{\mathsf{gid}}}$ *and* $\mathsf{st}_{P'}$ *are not corrupted before* P^{gid} *and* P' *joined the group* gid*, and*
3) *the group* gid *is not revealed.*

We say the freshness condition $\mathsf{fresh}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}_{\mathsf{KPriv}}(\mathsf{id}_{P'}, \mathsf{gid}, \mathsf{gkid})$ *holds if and only if*

1) *the group key* $gk_P^{(\mathsf{gid},\mathsf{gkid})}$ *is not leaked for all authorized parties* P *in the group* gid *with* $\mathsf{id}_P \in GP^{(\mathsf{gid},\mathsf{gkid})}$*,*
2) *the short-term state* π_P^{gid} *is not compromised for all authorized parties* P *in the group* gid *with* $\mathsf{id}_P \in GP^{(\mathsf{gid},\mathsf{gkid})}$*,*
3) *the long-term state* $\mathsf{st}_{P^{\mathsf{gid}}}$ *of the leader* P^{gid} *in the group* gid *is not corrupted before all authorized parties* P *with* $\mathsf{id}_P \in GP^{(\mathsf{gid},\mathsf{gkid})}$ *joined the group* gid*,*
4) *the long-term state* st_P *of all authorized participants* P *in the group* gid *with* $\mathsf{id}_P \in GP^{(\mathsf{gid},\mathsf{gkid})}$ *is not corrupted before* P *joined the group* gid*, and*
5) *the group* gid *is not revealed.*

Conclusion. Similar to the Sec-mGKD-pki model, it is straightforward that our Sec-mGKD-pw model also satisfies all guarantees listed in Section 4.3. However, there are two main differences between this Sec-mGKD-pw model and the previous Sec-mGKD-pki model:

1) Both $\mathsf{fresh}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}_{\mathsf{KAuth}}$ and $\mathsf{fresh}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}_{\mathsf{KPriv}}$ allow attackers to manipulate the transmission of all messages, including the sign-up messages of all parties in any group. This indicates that our Sec-mGKD-pw security model captures the full end-to-end security, i.e., against a malicious server. Consequently, this new Sec-mGKD-pw model solves the first restriction of Sec-mGKD-pki model.
2) While the $\mathsf{fresh}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}_{\mathsf{KPriv}}(\mathsf{id}_{P'}, \mathsf{gid}, \mathsf{gkid})$ condition requires no group key leakage, no short-term per-group state compromise, and no long-term state corruption for all parties P in the group with $\mathsf{id}_P \in GP^{(\mathsf{gid},\mathsf{gkid})}$, our new $\mathsf{fresh}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}_{\mathsf{KPriv}}(\mathsf{id}_{P'}, \mathsf{gid}, \mathsf{gkid})$ condition has the same requirement but only for the authorized parties. By this, our Sec-mGKD-pw model captures the following property:

- **(Implicit) Group Member Authentication**: The attacker can leak arbitrarily many group keys of any unauthorized party in the tested group. If an unauthorized party can successfully produce a same group key as a leader, then the attacker can test this leader, leak the group key of this unauthorized party, and win via the event E_{KPriv}.

4.5. The Sec-mGKD-pw+ Security Model

Note that the Sec-mGKD-pki and Sec-mGKD-pw models rely on different assumptions: trusted PKI and secure group secret. We then define a third Sec-mGKD-pw+ model that incorporates the above two models. The goal of the Sec-mGKD-pw+ model is to capture the security of a mGKD protocol if either the "trusted PKI" or the "secure group secret" assumption holds.

Definition 7 (Sec-mGKD-pw+ Freshness Conditions)**.** *We say the freshness condition* $\mathsf{fresh}^{\mathsf{Sec\text{-}mGKD\text{-}pw+}}_{\mathsf{KAuth}}(\mathsf{id}_{P'}, \mathsf{gid}, \mathsf{gkid})$ *holds if and only if* $\mathsf{fresh}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}_{\mathsf{KAuth}}(\mathsf{id}_{P'}, \mathsf{gid}, \mathsf{gkid})$ *or* $\mathsf{fresh}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}_{\mathsf{KAuth}}(\mathsf{id}_{P'}, \mathsf{gid}, \mathsf{gkid})$ *holds.*

We say the freshness condition $\mathsf{fresh}^{\mathsf{Sec\text{-}mGKD\text{-}pw+}}_{\mathsf{KPriv}}(\mathsf{id}_{P'}, \mathsf{gid}, \mathsf{gkid})$ *holds if and only if* $\mathsf{fresh}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}_{\mathsf{KPriv}}(\mathsf{id}_{P'}, \mathsf{gid}, \mathsf{gkid})$ *or* $\mathsf{fresh}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}_{\mathsf{KPriv}}(\mathsf{id}_{P'}, \mathsf{gid}, \mathsf{gkid})$ *holds.*

The following corollary is straightforward by definition:

Corollary 1. *Let* Π *denote a* mGKD *protocol. If* Π *is* Sec-mGKD-pki *and* Sec-mGKD-pw *secure, then* Π *is also* Sec-mGKD-pw+ *secure, and vice versa.*

5. Zoom's protocol is a mGKD protocol

The Zoom library allows parties to establish end-to-end group meeting communication. In this section, we first introduce the Zoom overview [35] (version 4.0) in Section 5.1. Then, we show that Zoom library is Sec-mGKD-pki secure in Section 5.2 but not Sec-mGKD-pw secure in Section 5.3.

5.1. The Zoom End-to-End Connection Overview

For end-to-end encrypted meetings, Zoom only supports connecting from installed clients: Browser-based connections are not supported. Zoom distinguishes between users and devices by non-cryptographic user identifiers uid and hardware identifiers hid. We model the identifier of each party P by a pair of user and hardware identifiers, i.e., $\mathsf{id}_P = (\mathsf{uid}_P, \mathsf{hid}_P)$ and assume that party identifiers are unique[1].

Zoom deploys two infrastructures for transmitting cryptographic primitives: an identity management system and a multimedia router (MMR). While the identity management system distributes cryptographic public keys generated by individual clients, the MMR distributes cryptographic messages between parties in a meeting. The connection between parties and servers are established on TLS-tunnels over TCP and are encrypted with AES in GCM mode. In this paper, we assume the existence of Zoom servers but do not explicitly model them, because our goal is to consider them adversary-controlled. Zoom allows every party to set up a group meeting. Groups are uniquely identified by their group identifiers gid. Each group meeting is equipped with a specific "bulletin board", where all parties can post (their own) and retrieve (others') cryptographic messages. The server is able to control and tamper with the bulletin boards.

The end-to-end secure Zoom library consists of six phases following the mGKD protocol syntax, of which we give an overview in Figure 1. We recall the cryptographic algorithms ZSign and ZBox underlying the Zoom library in Appendix B. For domain separation, Zoom uses hardcoded context strings (ctxt_1,ctxt_2,ctxt_3).

Sign Up $\mathsf{SignUp}(P)$***:*** During the Sign Up phase, every party P samples an identity public-private ZSign key pair and stores them into the long-term state st_P. The party P outputs the identity public key to the server as the sign-up message. This algorithm is executed only once for each party, i.e., each user on each hardware device.[2]

Group Schedule $\mathsf{Schedule}(P, \mathsf{gid}, gs)$***:*** During the Group Schedule phase, the leader P parses a passcode pc^{gid} from the input gs. The leader P sends pc^{gid} to not only the server as the group schedule message $m^{\mathsf{gid}}_{\mathsf{GSch}}$ for the access control management, but also to the authorized participants for joining the group over out-of-band channels, e.g., email.

Register $\mathsf{Register} = (\mathsf{Register\text{-}L}, \mathsf{Register\text{-}P})$***:*** The Register phase enables every party P to register for joining the meeting gid. We separate the description for $\mathsf{Register\text{-}P}(P', \mathsf{gid}, gs, m)$, where the P' is a participant of the group gid, and for $\mathsf{Register\text{-}L}(P, \mathsf{gid}, gs, m)$, where P the leader of the group gid.

- $\mathsf{Register\text{-}P}(P', \mathsf{gid}, gs, m)$: The input message m is given by the server and should be correctly parsed as a special mUUID string. The mUUID string is a server-generated per-group-instance random string that the individual parties cannot control. Moreover, the participant P' also inputs a group secret gs that can be correctly parsed as a passcode pc^{gid}. This algorithm first samples a public-private per-group ZBox key pair and stores them into the state $\pi^{\mathsf{gid}}_{P'}$. Next, it computes $\mathsf{Binding}^{\mathsf{gid}}_{P'}$, which is the concatenation of the group identifier gid, server-generated random string mUUID, as well as the party P''s identifier $\mathsf{id}_{P'} = (\mathsf{uid}, \mathsf{hid})$, identity public key $\mathsf{st}_{P'}.ipk$, and per-group public key $\pi^{\mathsf{gid}}_{P'}.pk$. Then, it signs the binding information $\mathsf{Binding}^{\mathsf{gid}}_{P'}$ for a signature $\sigma^{\mathsf{gid}}_{P'}$ using ZSign.Sign algorithm, the identity secret key $\mathsf{st}_{P'}.isk$, and a context string ctxt_1. The passcode pc^{gid} and the output register message $m^{(P',\mathsf{gid})}_{\mathsf{Reg}}$ consisting of the per-group public key

[1]The Zoom white-paper [35] states that the user identifiers uid are assigned by servers and the hardware identifiers hid are randomly sampled. Based on this, we assume that they are unique in practice.

[2]The Zoom library also supports anonymous log-in: people without a Zoom account can also join a group meeting as a "guest participant" (note that the guest cannot play the role of leader). Before a guest joins a group, the Sign Up algorithm is always executed. This prevents other parties from tracing them across meetings by noticing when a long-term key is reused [35].

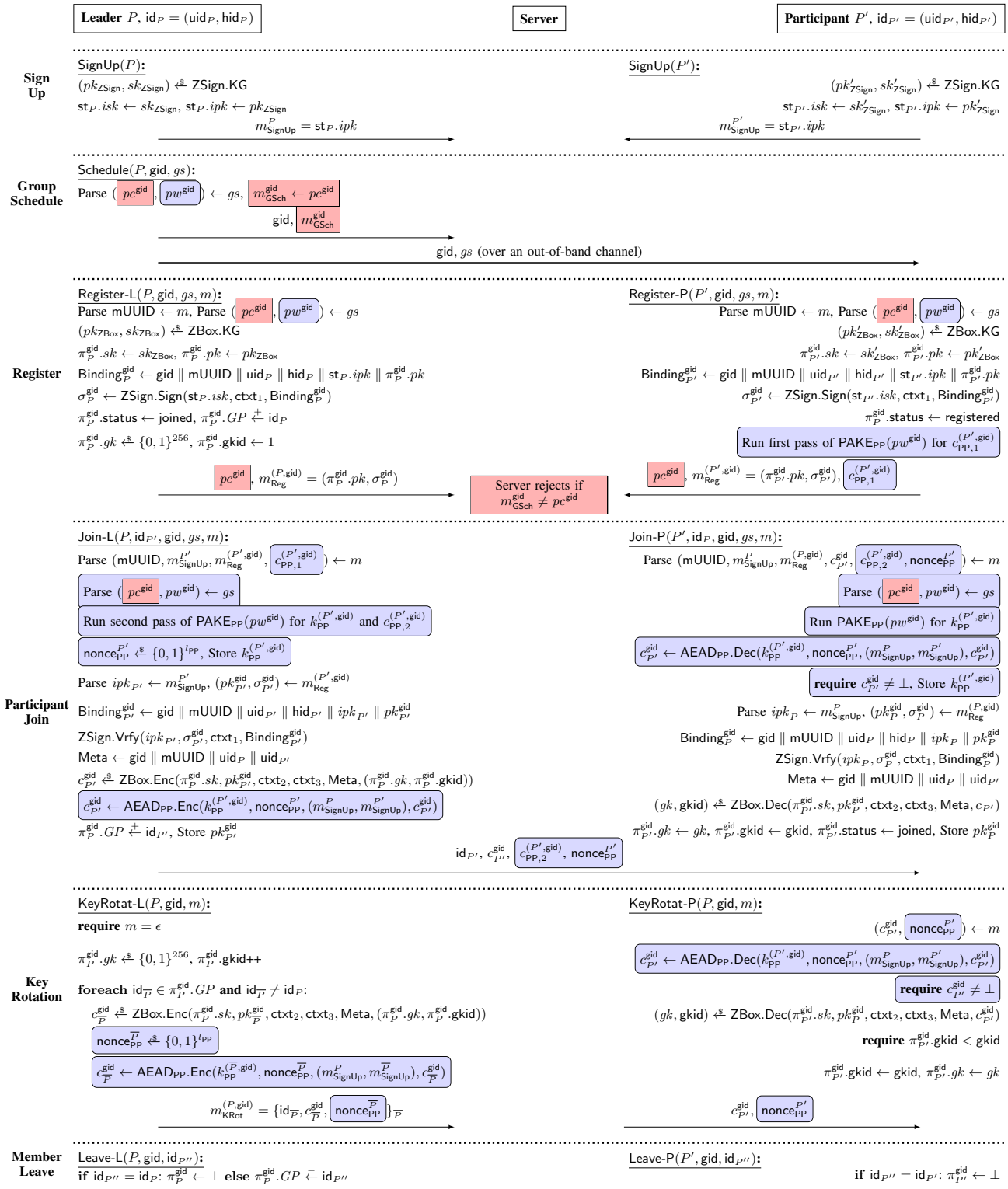

Figure 1. Overview of the **Zoom** protocol and our modified **ZoomPAKE** protocol. The boxes of the form added denote the additions from our transformation for ZoomPAKE, i.e., which are not in current Zoom. Note we do require any additional message flows. The boxes of the form redundant denote the elements that become redundant in our PAKE-based design: thus, for ZoomPAKE, we can essentially set pc^{gid} to the empty string, and still obtain the same guarantees, effectively replacing the old passcode by the new PAKE password. We recall the cryptographic algorithms ZSign and ZBox from the Zoom library in Appendix B. All communications from and to the server are performed over unilaterally secured TLS connections, authenticating the server. Possible out-of-band channels in the Group Schedule include, e.g., e-mail, messaging app, phone, calendar invite, or in-person.

$\pi^{\mathsf{gid}}_{P'}.pk$ and the signature $\sigma^{\mathsf{gid}}_{P'}$ is sent to the server. The server adds $m^{(P',\mathsf{gid})}_{\mathsf{Reg}}$ to the "bulletin board" of the group gid, if the passcode pc^{gid} matches the group schedule message $m^{\mathsf{gid}}_{\mathsf{GSch}}$ received from the group leader, and rejects it otherwise. The status $\pi^{\mathsf{gid}}_{P'}.\mathsf{status}$ of the participant P' in the group gid is set to registered.

- $\mathsf{Register\text{-}L}(P, \mathsf{gid}, gs, m)$: If the party P is the leader of the group gid, P runs the same execution as a participant except for setting the status $\pi^{\mathsf{gid}}_{P}.\mathsf{status}$ to joined rather than registered. Moreover, the leader P initializes the group by sampling the first group key $\pi^{\mathsf{gid}}_{P}.gk$ of bit length 256 and sets the group key index $\pi^{\mathsf{gid}}_{P}.\mathsf{gkid}$ to 1. The identifier id_P is added into the local party set $\pi^{\mathsf{gid}}_{P}.GP$.

Participants Join $\mathsf{Join} = (\mathsf{Join\text{-}L}, \mathsf{Join\text{-}P})$***:*** The Zoom library executes this interactive sub-protocol between the leader P and a participant P' only one-pass:

- $\mathsf{Join\text{-}L}(P, \mathsf{id}_{P'}, \mathsf{gid}, gs, m)$: When the leader P notices the joining request of a new participant P', P retrieves an incoming message m from the server and the group gid's "bulletin board" followed by parsing it into: (1) a server-generated randomness mUUID, (2) the participant P''s sign-up message $m^{P'}_{\mathsf{SignUp}}$, and (3) the participant P''s register message $m^{(P',\mathsf{gid})}_{\mathsf{Reg}}$. Next, the leader P parses the participant P''s identity public key $ipk_{P'}$, per-group public key $pk^{\mathsf{gid}}_{P'}$, and per-group signature $\sigma^{\mathsf{gid}}_{P'}$ from the incoming message, followed by using them to produce the participant's binding information $\mathsf{Binding}^{\mathsf{gid}}_{P'}$. If the binding information cannot pass the verification $\mathsf{ZSign.Vrfy}$ upon the participant's identity public key $ipk_{P'}$, signature $\sigma^{\mathsf{gid}}_{P'}$, and the context ctxt_1, then the leader aborts and undoes the previous executions. Otherwise, the leader creates a meta data Meta by concatenating the group identifier gid, server-generated randomness mUUID, the leader's user identifier uid_P, and the participant's user identifier $\mathsf{uid}_{P'}$. Finally, the leader P encrypts the current group key $\pi^{\mathsf{gid}}_{P}.gk$ as well as the index $\pi^{\mathsf{gid}}_{P}.\mathsf{gkid}$ using the $\mathsf{ZBox.Enc}$ encryption algorithm and the leader P's per-group secret key $\pi^{\mathsf{gid}}_{P}.sk$, the participant P''s per-group public key $\pi^{\mathsf{gid}}_{P'}.pk$, and auxiliary information ctxt_2, ctxt_3, and Meta. The identifier of the participant P' and the ZBox ciphertext $c^{\mathsf{gid}}_{P'}$ are send to P' via the server. The leader P stores the identifier $\mathsf{id}_{P'}$ of the participant P' into the local party set $\pi^{\mathsf{gid}}_{P}.GP$ and stores the participant's per-group public key $pk^{\mathsf{gid}}_{P'}$.
- $\mathsf{Join\text{-}P}(P', \mathsf{id}_P, \mathsf{gid}, gs, m)$: When a participant P', who registered for a group gid, receives an incoming message m that includes (1) a server-generated randomness mUUID, (2) the leader P's sign-up message m^{P}_{SignUp}, (3) the leader P's register message $m^{(P,\mathsf{gid})}_{\mathsf{Reg}}$, and (4) the leader's reply $c^{\mathsf{gid}}_{P'}$, P' first parses the incoming message, creates the leader's binding information, and verifies it using $\mathsf{ZSign.Vrfy}$ algorithm, similar to the leader's execution. Then, P' also generates the meta data Meta and uses it together with its own per-group secret key $\pi^{\mathsf{gid}}_{P'}.sk$, the leader's per-group public key pk^{gid}_{P}, and the contexts ctxt_2 and ctxt_3, to decrypt the ciphertext $c^{\mathsf{gid}}_{P'}$. If any error occurs during above steps, then the participant P' aborts and undoes the previous executions. Otherwise, the participant P' stores the decrypted group key as well as the associated index into the per-group state $\pi^{\mathsf{gid}}_{P'}.gk$ and $\pi^{\mathsf{gid}}_{P'}.\mathsf{gkid}$, followed by setting the status $\pi^{\mathsf{gid}}_{P'}.\mathsf{status}$ to joined. Moreover, P' stores the leader P's per-group public key into the state. [3]

Key Rotation $\mathsf{KeyRotat} = (\mathsf{KeyRotat\text{-}L}, \mathsf{KeyRotat\text{-}P})$***:*** The execution of this algorithm is distinct according to the caller P's role: a leader or a participant. We separate the description for $\mathsf{KeyRotat\text{-}L}(P, \mathsf{gid}, m)$, where P is the leader of the group gid, and for $\mathsf{KeyRotat\text{-}P}(P', \mathsf{gid}, m)$, where P' is a participant of the group gid.

- $\mathsf{KeyRotat\text{-}L}(P, \mathsf{gid}, m)$: The leader P of the group gid executes this algorithm without any auxiliary incoming input, i.e., $m = \epsilon$. The leader P samples a new group key $\pi^{\mathsf{gid}}_{P}.gk$ of bit length 256 and increments the corresponding index $\pi^{\mathsf{gid}}_{P}.\mathsf{gkid}$ by 1. Similar to the encryption during the Participant Join phase, the leader encrypts the new group key and index for each party in its local party set $\pi^{\mathsf{gid}}_{P}.GP$ except for himself. The output is a ciphertext bundle that includes the identifiers of each participant and the customized ciphertexts.

The server is expected to split the ciphertext bundles and to send each ciphertext to the specified participant.

- $\mathsf{KeyRotat\text{-}P}(P', \mathsf{gid}, m)$: The participant P' of the group gid first parses the incoming message m from the server to an ZBox ciphertext $ct^{\mathsf{gid}}_{P'}$. Then, P' decrypts the new group key gk and index gkid as during the Participant Join phase. If any error occurs in the above steps or the decrypted group key index is smaller than or equal to the local one, then the participant P' aborts and undoes the previous executions. Otherwise, P' simply overwrites the local group key as well as the index by the new ones.

Member Leave $\mathsf{Leave} = (\mathsf{Leave\text{-}L}, \mathsf{Leave\text{-}P})$***:*** The execution of this algorithm is distinct according to the caller P's role: a leader or a participant. We separate the description for $\mathsf{Leave\text{-}L}(P, \mathsf{gid}, \mathsf{id}_{P''})$, where the P is the leader of the group gid, and for $\mathsf{Leave\text{-}P}(P', \mathsf{gid}, \mathsf{id}_{P''})$, where P' is a participant of the group gid.

- $\mathsf{Leave\text{-}L}(P, \mathsf{gid}, \mathsf{id}_{P''})$: If $\mathsf{id}_P = \mathsf{id}_{P''}$, i.e., the leader P wants to leave the group gid, then P erases the per-group instance π^{gid}_{P}. Otherwise, the leader P notices a party P'' leaving the group gid, P simply removes the identifier of the participant $\mathsf{id}_{P''}$ from the local party set $\pi^{\mathsf{gid}}_{P}.GP$.
- $\mathsf{Leave\text{-}P}(P', \mathsf{gid}, \mathsf{id}_{P''})$: If $\mathsf{id}_{P'} = \mathsf{id}_{P''}$, i.e., the participant P' wants to leave the group gid, then P' erases the per-group instance $\pi^{\mathsf{gid}}_{P'}$. Otherwise, P' performs no action.

Instantiations. Underlying the ZSign and ZBox algorithms, the key derivation function H_1 is SHA256 and H_2 is HKDF algorithm (using an empty salt parameter). The length l

[3] In practice, Zoom has an independent mechanism for leader P to synchronize the party set $\pi^{\mathsf{gid}}_{P}.GP$ with every participant P'. We omit it here since, this does not impact Zoom security analysis in our models.

underlying ZBox algorithm is 192. The elliptic curve ECDH underlying Diffie-Hellman key exchange is Curve25519. The ZSign algorithm relies DS on EdDSA over Ed25519. The AEAD algorithm is xchacha20poly1305.

5.2. Zoom is Sec-mGKD-pki secure

We omit the correctness analysis of the Zoom library. Below, we investigate the provable security of the Zoom library. The proof of the following theorem is give in Appendix C.

Theorem 1. *Let* Π *denote the end-to-end Zoom protocol in Section 5.1. Assume the* $\epsilon_{\mathsf{H}_1}^{\text{coll-res}}$*-collision resistance of the underlying* H_1*, the* $\epsilon_{\mathsf{DS}}^{\text{euf-cma}}$-euf-cma *security of* DS*, the* $\epsilon_{\mathsf{AEAD}}^{(\mathsf{n},\mathsf{m})\text{-frob}}$-$(\mathsf{n},\mathsf{m})$-FROB *security,* $\epsilon_{\mathsf{AEAD}}^{\text{ind\$-cca}}$-IND$-CCA *security, and* $\epsilon_{\mathsf{AEAD}}^{\text{cti-cpa}}$-cti-cpa *security of the* AEAD*. Assume the* $\epsilon_{\mathsf{ECDH},\mathsf{H}_2}^{\text{mn-prf-ODH}}$ *hardness of the* mn-prf-ODH *problem over* ECDH *and function* H_2*. The advantage of any* PPT *attacker* $\mathcal{A}$ *that breaks the* Sec-mGKD-pki *security of* Π *is bounded by,*

$$\begin{aligned}\mathsf{Adv}_{\Pi}^{\text{Sec-mGKD-pki}}(\mathcal{A}) \le{}& \epsilon_{\mathsf{H}_1}^{\text{coll-res}} + q_{\text{NEWPARTY}}\epsilon_{\mathsf{DS}}^{\text{euf-cma}} \\ &+ c_{\mathsf{maxReg}} q_{\text{NEWGROUP}}\Big(\epsilon_{\mathsf{AEAD}}^{\text{cti-cpa}} + \epsilon_{\mathsf{AEAD}}^{(\mathsf{n},\mathsf{m})\text{-frob}} \\ &+ c_{\mathsf{maxReg}}^{(n_{\mathsf{party}}-1)}(n_{\mathsf{party}}-1)(\epsilon_{\mathsf{ECDH},\mathsf{H}_2}^{\text{mn-prf-ODH}} + \epsilon_{\mathsf{AEAD}}^{\text{ind\$-cca}})\Big)\end{aligned}$$

where c_{maxParty} *denotes the maximal number of parties in every group,* c_{maxReg} *denotes the maximal number of register requests for every group,* $n_{\mathsf{party}} \le c_{\mathsf{maxParty}}$ *denotes the number of parties in the set* $GP^{(\mathsf{gid},\mathsf{gkid})}$ *for tested group identifier* gid *and group key index* gkid*, and* $q_{\mathcal{O}}$ *denote the maximal number of the queries to any oracle* $\mathcal{O}$*.*

Note that H_1 = SHA256 provides an expected collision resistance of 128 bits [36]. The EUF-CMA security of Ed25519 was proven in [37]. The security of xchacha20poly1305 was discussed in [38]. The maximal number of parties per meeting is $c_{\mathsf{maxParty}} = 1000$ [35]. The above theorem shows that the end-to-end Zoom library provably provides Sec-mGKD-pki security and satisfies all properties listed in Section 4.3.

Remark 1. *Theorem 1 shows that Zoom achieves* Sec-mGKD-pki *independent of the passcode: the passcode is only used implicitly for access control by honest servers.*

5.3. Zoom is not Sec-mGKD-pw secure

Recall that the Sec-mGKD-pki model has two restrictions in Section 4.3. A natural question arises whether these restrictions apply to the Sec-mGKD-pki secure Zoom library.

Does Zoom Provide Trusted PKI? The PKI is expected to "*enable users of an insecure public network such as the Internet to securely and privately exchange data through the use of a public and a private cryptographic key pair that is obtained and shared through a trusted authority*" [39, Charpter 1]. As we mentioned in Section 5.1, all public keys of all parties in the Zoom library are uploaded to an infrastructure, called "identity management system", that is fully controlled by Zoom. The identity management system distributes the identity public keys. While Zoom claims the end-to-end security, the goal of which is to protect the secrecy and integrity of the exchanged content between every two parties against all third parties including the service providers, assuming Zoom-controlled PKI trusted is controversial and doubtful. Considering a malicious server, the server can easily perform the "machine-in-the-middle" attack by forging the sign-up messages and impersonate any party towards others.

Although there do exist other group meeting providers, such as Cisco WebEx and Skype, that employ a third-party PKI, such as Microsoft Certificate Authority (CA), we stress that the reliability of PKI is still imperfect. Eckersley and Burns [40] revealed that 14 CAs had been compromised. Moreover, a number of attacks that successfully break several CAs, including DigiNotar, Comodo, GlobalSign, StartSSL, and TurkTrust, have been publicly noticed [41]. It is prudent to consider the potential PKI compromise.

Does Zoom Provide (Implicit) Group Member Authentication? Unfortunately, this does not hold for Zoom. Although the end-to-end Zoom library asks every leader to create every new group together with an associated passcode, the leaders hand over the power of passcode verification to the untrusted server. By colluding with the untrusted server, an unauthorized (and malicious) party can join every group without the knowledge of any passcode. The consequence is that nobody in the group (including the leader) can distinguish authorized participants from the others, in particular, in the end-to-end setting.

6. A Generic Approach to Sec-mGKD-pw Security: Password-Protected Transformation

If we could assume that each group gid has a unique high-entropy group secret gs^{gid} only shared by the leader and authorized participants of the group gid, we could design a trivial construction that meets the Sec-mGKD-pw security. We can simply use a message authentication code MAC with the group secret gs^{gid} as key to sign and verify all outgoing and incoming messages.

However, in practice we use low-entropy passwords for usability, allowing the passwords to be shared over various out-of-band channels. For instance, real-world service providers often support only short passwords[4]. This restricts the upper bound of the password entropy and enables attackers to perform dictionary attacks on the password, e.g., by brute force guessing.

In this section, we introduce a generic Password-Protected (PP) transformation that provably transforms any Sec-mGKD-pki secure mGKD protocol Π to another Sec-mGKD-pw secure $\Pi' = \mathsf{PP}[\Pi, \mathsf{PAKE}_{\mathsf{PP}}, \mathsf{AEAD}_{\mathsf{PP}}]$ protocol by using a password-authenticated key exchange $\mathsf{PAKE}_{\mathsf{PP}}$ and an authenticated encryption with associated

[4]For instance, meeting passcodes in Zoom are 1-16 digit numeric lock codes; the default meeting password in Cisco WebEx has ≥ 11 characters.

data $\mathsf{AEAD}_{\mathsf{PP}}$. We also prove that the PP transformation preserves Sec-mGKD-pki security, i.e., if Π is Sec-mGKD-pki secure, so is Π'. In this sense, Π' satisfies stronger security Sec-mGKD-pw+, due to Corollary 1. Finally, we illustrate how to apply our PP transformation to the Zoom library, and provide efficient instantiations for $\mathsf{PAKE}_{\mathsf{PP}}$ and $\mathsf{AEAD}_{\mathsf{PP}}$, without causing additional message flows.

6.1. The Generic Transformation

The goal of our PP transformation is to ensure that only the authorized parties that know the group secret can recover any group key, even if the server is malicious. The high-level overview of our PP transformation is to (1) let the leader and every participant run a $\mathsf{PAKE}_{\mathsf{PP}}$ protocol upon a new password (included in the group secret) to produce a symmetric key k_{PP} during the Participant Join phase, and (2) use the key k_{PP} and an $\mathsf{AEAD}_{\mathsf{PP}}$ scheme to encrypt/decrypt the original transcript of the mGKD protocol Π during the Participant Join and Key Rotation phases. Note that every participant has to first register for a group before joining it. To avoid introducing additional message flows, we design our PP transformation to shift the first pass of $\mathsf{PAKE}_{\mathsf{PP}}$ to the participant's Register phase. We give the formal definition of our PP transformation below.

Definition 8. *Let* $\Pi = (\mathsf{SignUp}, \mathsf{Schedule}, \mathsf{Register}, \mathsf{Join}, \mathsf{Leave}, \mathsf{KeyRotat})$ *denote a multi-stage group key distribution protocol. Let* $\mathsf{PAKE}_{\mathsf{PP}}$ *denote a password-authenticated key exchange scheme. Let* $\mathsf{AEAD}_{\mathsf{PP}}$ *denote an authenticated encryption with associated data. We define the password-protected (*PP*) transformation* $\mathsf{PP}[\Pi, \mathsf{PAKE}_{\mathsf{PP}}, \mathsf{AEAD}_{\mathsf{PP}}]$ *that outputs* $\Pi' = (\mathsf{SignUp}', \mathsf{Schedule}', \mathsf{Register}', \mathsf{Join}', \mathsf{Leave}', \mathsf{KeyRotat}')$ *as follows:*

Sign Up $\mathsf{SignUp}'(P)$**:** *Run* $m^P_{\mathsf{SignUp}} \xleftarrow{\$} \mathsf{SignUp}(P)$ *and stores* m^P_{SignUp} *locally into the long-term state* st_P.

Group Schedule $\mathsf{Schedule}'(P, \mathsf{gid}, gs)$**:** *Parse* $(gs^{\mathsf{gid}}_{\Pi}, pw^{\mathsf{gid}}) \leftarrow gs$*, run* $m^{\mathsf{gid}}_{\mathsf{GSch}} \xleftarrow{\$} \mathsf{Schedule}(P, \mathsf{gid}, gs^{\mathsf{gid}}_{\Pi})$*, and output* $m^{\mathsf{gid}}_{\mathsf{GSch}}$*. The full group secret* gs *is sent to authorized parties over out-of-band channels.*

Register $\mathsf{Register}' = (\mathsf{Register\text{-}L}', \mathsf{Register\text{-}P}')$**:** *We define the sub-algorithms as follows:*

- $\mathsf{Register\text{-}L}'(P, \mathsf{gid}, gs, m)$*: Parse* $(gs^{\mathsf{gid}}_{\Pi}, pw^{\mathsf{gid}}) \leftarrow gs$*, run* $m' \xleftarrow{\$} \mathsf{Register\text{-}L}(P, \mathsf{gid}, gs^{\mathsf{gid}}_{\Pi}, m)$*, and output* m'.
- $\mathsf{Register\text{-}P}'(P, \mathsf{gid}, gs, m)$*: First, parse* $(gs^{\mathsf{gid}}_{\Pi}, pw^{\mathsf{gid}}) \leftarrow gs$*. Next, run* $m' \xleftarrow{\$} \mathsf{Register\text{-}P}(P, \mathsf{gid}, gs^{\mathsf{gid}}_{\Pi}, m)$*. Then, run the first pass of* $\mathsf{PAKE}_{\mathsf{PP}}$ *upon the password* pw^{gid} *for a ciphertext* $c^{(P,\mathsf{gid})}_{\mathsf{PP}}$*. Finally, output* $(m', c^{(P,\mathsf{gid})}_{\mathsf{PP}})$.

Participant Join $\mathsf{Join}' = (\mathsf{Join\text{-}L}', \mathsf{Join\text{-}P}')$**:** *This phase consists of two steps. In either step, if any error occurs during this algorithm, the caller* P *aborts and undoes the executions. In the first step, the leader and the participant run* $\mathsf{PAKE}_{\mathsf{PP}}$ *until* $\mathsf{PAKE}_{\mathsf{PP}}$ *outputs a key* k_{PP}.

- $\mathsf{Join\text{-}L}'(P, \mathsf{id}_{P'}, \mathsf{gid}, gs, m)$ *or* $\mathsf{Join\text{-}P}'(P, \mathsf{id}_{P'}, \mathsf{gid}, gs, m)$*: The caller* P *first parses* $(gs^{\mathsf{gid}}_{\Pi}, pw^{\mathsf{gid}}) \leftarrow gs$ *from the group secret and other necessary information for running* $\mathsf{PAKE}_{\mathsf{PP}}$ *from the input message* m*. Then,* P *runs the next pass of* $\mathsf{PAKE}_{\mathsf{PP}}$ *on* pw^{gid}*. If the key* k_{PP} *is still unavailable,* P *directly outputs the outgoing message of* $\mathsf{PAKE}_{\mathsf{PP}}$*. Otherwise, the key* k_{PP} *is stored into the per-group state* π^{gid}_P.

If the leader and the participant have computed the key k_{PP} *before this algorithm invocation or in the first step in this invocation, they execute the following second step.*

- $\mathsf{Join\text{-}L}'(P, \mathsf{id}_{P'}, \mathsf{gid}, gs, m)$*: The leader* P *first parses* $(gs^{\mathsf{gid}}_{\Pi}, pw^{\mathsf{gid}}) \leftarrow gs$ *from the group secret. If the original* Join-L *algorithm needs any incoming information from the participant* P'*, then the leader* P *extracts an* $\mathsf{AEAD}_{\mathsf{PP}}$ *ciphertext and an* $\mathsf{AEAD}_{\mathsf{PP}}$ *nonce from the input* m*. Next, the leader* P *decrypts the* $\mathsf{AEAD}_{\mathsf{PP}}$ *ciphertext using the key* k_{PP}*, the* $\mathsf{AEAD}_{\mathsf{PP}}$ *nonce, and an associated data consisting of both parties' sign-up messages, and obtains a message* m_1*. Then, the leader* P *extracts other necessary information* m_2 *from the input* m *for running* $m' \xleftarrow{\$} \mathsf{Join\text{-}L}(P, \mathsf{id}_{P'}, \mathsf{gid}, gs^{\mathsf{gid}}_{\Pi}, m_1 \parallel m_2)$*. After that,* P *encrypts* m' *using the* $\mathsf{AEAD}_{\mathsf{PP}}$ *key* k_{PP}*, a random nonce, and an associated data consisting of both parties' sign-up messages. Finally, the leader* P *outputs the* $\mathsf{AEAD}_{\mathsf{PP}}$ *ciphertext and nonce.*
- $\mathsf{Join\text{-}P}'(P, \mathsf{id}_{P'}, \mathsf{gid}, gs, m)$*: The participant* P *first parses* $(gs^{\mathsf{gid}}_{\Pi}, pw^{\mathsf{gid}}) \leftarrow gs$ *from the group secret. If the original* Join-P *algorithm needs any incoming information from the leader* P'*, then the participant* P *extracts an* $\mathsf{AEAD}_{\mathsf{PP}}$ *ciphertext and an* $\mathsf{AEAD}_{\mathsf{PP}}$ *nonce from the input* m*. Next, the participant* P *decrypts the* $\mathsf{AEAD}_{\mathsf{PP}}$ *ciphertext using the key* k_{PP}*, the* $\mathsf{AEAD}_{\mathsf{PP}}$ *nonce, and an associated data consisting of both parties' sign-up messages, and obtains a message* m_1*. Then, the participant* P *extracts other necessary information* m_2 *from the input* m *for running* $m' \xleftarrow{\$} \mathsf{Join\text{-}P}(P, \mathsf{id}_{P'}, \mathsf{gid}, gs^{\mathsf{gid}}_{\Pi}, m_1 \parallel m_2)$*. After that,* P *encrypts* m' *using the* $\mathsf{AEAD}_{\mathsf{PP}}$ *key* k_{PP}*, a random nonce, and an associated data consisting of both parties' sign-up messages. Finally, the participant* P *outputs the* $\mathsf{AEAD}_{\mathsf{PP}}$ *ciphertext and nonce.*

Member Leave $\mathsf{Leave}' = (\mathsf{Leave\text{-}L}', \mathsf{Leave\text{-}P}')$**:** *These algorithms are identical to the original* $\mathsf{Leave} = (\mathsf{Leave\text{-}L}, \mathsf{Leave\text{-}P})$*. Note that if a per-group state is erased, then the stored key* k_{PP} *must also be erased.*

Key Rotation $\mathsf{KeyRotat}' = (\mathsf{KeyRotat\text{-}L}', \mathsf{KeyRotat\text{-}P}')$**:** *We define the sub-algorithms as follows. If any error occurs during the above execution, then the caller aborts and undoes the executions in this invocation.*

- $\mathsf{KeyRotat\text{-}L}'(P, \mathsf{gid}, m)$*: The leader* P *first runs the original* $m_{\mathsf{KRot}} \xleftarrow{\$} \mathsf{KeyRotat\text{-}L}(P, \mathsf{gid}, m)$*. Then, the leader* P *extracts the portion* $c_{\overline{P}}$ *in* m_{KRot} *that is specific to every participant* $\overline{P}$ *in the group* gid*, followed by encrypting it using the stored corresponding* $\mathsf{AEAD}_{\mathsf{PP}}$ *key* k_{PP}*, a random nonce, and an associated data consisting of both parties' sign-up messages as in the Participant Join phase. Finally, the leader* P *outputs the* $\mathsf{AEAD}_{\mathsf{PP}}$ *ciphertext and nonce for every participant* $\overline{P}$ *in the group* gid.

- $\mathsf{KeyRotat\text{-}P}'(P, \mathsf{gid}, m)$*: The participant P first extracts an* $\mathsf{AEAD_{PP}}$ *ciphertext and an* $\mathsf{AEAD_{PP}}$ *nonce from the input m. Next, P recovers a message m_1 from the* $\mathsf{AEAD_{PP}}$ *ciphertext using the stored corresponding* $\mathsf{AEAD_{PP}}$ *key k_{PP}, the* $\mathsf{AEAD_{PP}}$ *nonce, and an associated data consisting of both parties' sign-up messages as in the Participant Join phase. Then, the participant P extracts other necessary information m_2 from the input m for running $m_{\mathsf{KRot}} \xleftarrow{\$} \mathsf{KeyRotat\text{-}P}(P, \mathsf{gid}, m_1 \parallel m_2)$ and outputting m_{KRot}.*

For brevity we omit the correctness analysis. The theorem below shows that our PP transformation provably turns a Sec-mGKD-pki secure Π into a Sec-mGKD-pw secure $\Pi' = \mathsf{PP}[\Pi, \mathsf{PAKE_{PP}}, \mathsf{AEAD_{PP}}]$ protocol. We give the theorem's proof in Appendix D.

Theorem 2. *Let Π denote a* mGKD *protocol. Let* $\mathsf{PAKE_{PP}}$ *denote a password-authenticated key exchange scheme. Let* $\mathsf{AEAD_{PP}}$ *denote an authenticated encryption with associated data scheme. Let $\Pi' = \mathsf{PP}[\Pi, \mathsf{PAKE_{PP}}, \mathsf{AEAD_{PP}}]$. Let $\mathcal{D} = \mathcal{D}_\Pi \times \mathcal{D}_{pw}$ denote the distribution of the group secrets. Assume the $\epsilon^{\text{w-PAKE}}_{\mathsf{PAKE_{PP}},\mathcal{D}_{pw}}$-w-PAKE security of the underlying* $\mathsf{PAKE_{PP}}$*, the $\epsilon^{\text{d-frob}}_{\mathsf{AEAD_{PP}}}$-d-FROB security, $\epsilon^{\text{ind\$-cca}}_{\mathsf{AEAD_{PP}}}$-IND\$-CCA security, and $\epsilon^{\text{cti-cpa}}_{\mathsf{AEAD_{PP}}}$-cti-cpa security of* $\mathsf{AEAD_{PP}}$*. If there exists any* PPT *attacker $\mathcal{A}$ that breaks the* Sec-mGKD-pw *security of Π', then there must exist a* PPT *attacker $\mathcal{B}$ that breaks the* Sec-mGKD-pki *security of Π such that*

$$\mathsf{Adv}^{\text{Sec-mGKD-pw}}_{\Pi'}(\mathcal{A}) \leq q_{\textsc{NewGroup}}\Big(\epsilon^{\text{w-PAKE}}_{\mathsf{PAKE_{PP}},\mathcal{D}_{pw}} + \epsilon^{\text{d-frob}}_{\mathsf{AEAD_{PP}}} + c_{\mathsf{maxReg}}\big(\epsilon^{\text{cti-cpa}}_{\mathsf{AEAD_{PP}}} + \epsilon^{\text{ind\$-cca}}_{\mathsf{AEAD_{PP}}}\big) + \mathsf{Adv}^{\text{Sec-mGKD-pki}}_{\Pi}(\mathcal{B})\Big)$$

where c_{maxReg} denotes the maximal number of register requests for every group and $q_{\mathcal{O}}$ denotes the maximal number of queries to any oracle $\mathcal{O}$.

Below, we further show that our PP transformation preserves the Sec-mGKD-pki security of the original mGKD protocol Π. We give the theorem's proof in Appendix E.

Theorem 3. *Let Π denote a* mGKD *protocol. Let* $\mathsf{PAKE_{PP}}$ *denote a password-authenticated key exchange scheme. Let* $\mathsf{AEAD_{PP}}$ *denote an authenticated encryption with associated data. Let $\Pi' = \mathsf{PP}[\Pi, \mathsf{PAKE_{PP}}, \mathsf{AEAD_{PP}}]$. If there exists any* PPT *attacker $\mathcal{A}$ that breaks the* Sec-mGKD-pki *security of Π', then there exists another* PPT *attacker $\mathcal{B}$ that breaks the* Sec-mGKD-pki *security of Π such that*

$$\mathsf{Adv}^{\text{Sec-mGKD-pki}}_{\Pi'}(\mathcal{A}) \leq \mathsf{Adv}^{\text{Sec-mGKD-pki}}_{\Pi}(\mathcal{B})$$

Combining these two theorems, our PP transformation provably endows a Sec-mGKD-pki secure mGKD protocol Π with Sec-mGKD-pw security while preserving the original one, i.e., Sec-mGKD-pw+ security due to Corollary 1.

6.2. Application to the Zoom Library

Then, we illustrate how to apply our PP transformation to the Zoom library in Section 5.1 by using a 2-pass $\mathsf{PAKE_{PP}}$ scheme and an $\mathsf{AEAD_{PP}}$ scheme. We call the transformed version ZoomPAKE and show the resulting protocol in Figure 1, where we use boxes to indicate the modifications. Note that Zoom achieves Sec-mGKD-pki security without relying on passcodes, as stated in Remark 1. The passcodes for the server's access control underlying Zoom are redundant in the stronger threat model, and can therefore be set to the empty string without impacting Sec-mGKD-pw security. In the following we assume that the passcode is set to the empty string, which amounts to replacing the passcode with the new PAKE password computations.

The Sign Up and Member Leave phases are unchanged.

ZoomPAKE Group Schedule Phase: This algorithm is nearly identical to the original one except that the leader sends the new password to authorized parties over an out-of-band channel instead of sending the passcode to the server.

ZoomPAKE Register Phase: Similar to the previous, parties no longer need to send the passcode to the server. Then, each participant P' runs the first pass of $\mathsf{PAKE_{PP}}$ on the password pw^{gid} for a ciphertext $c^{(P',\mathsf{gid})}_{\mathsf{PP},1}$ and outputs both the original outgoing messages and $c^{(P',\mathsf{gid})}_{\mathsf{PP},1}$.

ZoomPAKE Participant Join Phase: Our PP transformation modifies both leaders' and participants' execution.

From the leader P's side, P firsts parses an additional $\mathsf{PAKE_{PP}}$ ciphertext $c^{(P',\mathsf{gid})}_{\mathsf{PP},1}$ from the input, and uses the group secret as the password pw^{gid}. Next, the leader P runs the second pass of $\mathsf{PAKE_{PP}}$ with necessary input for a key $k^{(P',\mathsf{gid})}_{\mathsf{PP}}$ and a ciphertext $c^{(P',\mathsf{gid})}_{\mathsf{PP},2}$. Then, P stores $k^{(P',\mathsf{gid})}_{\mathsf{PP}}$ and samples a random nonce of length l_{PP} uniformly at random. After that, P executes the original computation. When the ZBox encryption $c^{\mathsf{gid}}_{P'}$ is derived, the leader P further re-encrypts it using the key $k^{(P',\mathsf{gid})}_{\mathsf{PP}}$, the random nonce $\mathsf{nonce}^{P'}_{\mathsf{PP}}$, and the associated data consisting both parties' sign-up messages. The $\mathsf{PAKE_{PP}}$ ciphertext $c^{(P',\mathsf{gid})}_{\mathsf{PP},2}$ and the $\mathsf{AEAD_{PP}}$ ciphertext are output.

From the participant P''s side, P' first parses two more components from the input messages: a $\mathsf{PAKE_{PP}}$ ciphertext $c^{(P',\mathsf{gid})}_{\mathsf{PP},2}$ and a nonce $\mathsf{nonce}^{P'}_{\mathsf{PP}}$. The participant P' also uses the group secret gs as the password pw^{gid}. Note that the original ciphertext $c^{\mathsf{gid}}_{P'}$ is not the one of ZBox anymore but the one of $\mathsf{AEAD_{PP}}$. Next, P' runs $\mathsf{PAKE_{PP}}$ for a key $k^{(P',\mathsf{gid})}_{\mathsf{PP}}$ and decrypts the $\mathsf{AEAD_{PP}}$ ciphertext $c^{\mathsf{gid}}_{P'}$ using the key $k^{(P',\mathsf{gid})}_{\mathsf{PP}}$, the nonce $\mathsf{nonce}^{P'}_{\mathsf{PP}}$, and the associated data consisting of the leader P's and the participant P''s sign-up messages, for the original ZBox ciphertext. If any error occurs during this step, the participant P' simply aborts. Otherwise, the key $k^{(P',\mathsf{gid})}_{\mathsf{PP}}$ is stored locally. The remaining computation of P' remains the same.

ZoomPAKE Key Rotation Join Phase: The key rotation phase is very similar to the original one. The only difference from the leader P's side is that P has to encrypt the ZBox ciphertext using $\mathsf{AEAD_{PP}}$ for every participant $\overline{P}$ in his local party set, i.e., $\mathsf{id}_{\overline{P}} \in \pi^{\mathsf{gid}}_P.GP$ and $\mathsf{id}_{\overline{P}} \neq \mathsf{id}_P$, using

the stored key $k_{\mathsf{PP}}^{(\overline{P},\mathsf{gid})}$, output by $\mathsf{PAKE}_{\mathsf{PP}}$, a independently random nonce $\mathsf{nonce}_{\mathsf{PP}}^{\overline{P}}$, the associated data consisting of the sign-up messages of P and $\overline{P}$. The output is a ciphertext bundle that includes $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertexts rather than the original ZBox ciphertexts.

When receiving the $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext, a participant P' first decrypts it by using the stored key $k^{(P',\mathsf{gid})}$ for a ZBox ciphertext $c_{P'}^{\mathsf{gid}}$. Then, P' simply runs the original KeyRotat-P algorithm using the new ciphertext $c_{P'}^{\mathsf{gid}}$.

Instantiation Suggestions. We suggest to instantiate the underlying $\mathsf{PAKE}_{\mathsf{PP}}$ with CPace [25] or SPAKE2 [27] for the w-PAKE security, see Section A.5. The $\mathsf{AEAD}_{\mathsf{PP}}$ can be instantiated with CAU-C4 or CAU-SIV-C4 [42] for the d-FROB security, see Section A.2.

References

[1] "How many people use Zoom?" 2022, https://www.zippia.com/advice/zoom-meeting-statistics/, (Accessed Feb 2023).

[2] T. Isobe and R. Ito, "Security analysis of end-to-end encryption for Zoom meetings," *IEEE Access*, vol. 9, pp. 90 677–90 689, 2021.

[3] ——, "Security Analysis of End-to-End Encryption for Zoom Meetings," Cryptology ePrint Archive, Paper 2021/486, 2021, https://eprint.iacr.org/2021/486.

[4] Multi-Stage Group Key Distribution and PAKEs: Securing Zoom Groups against Malicious Servers without New Security Elements (Full version with detailed proofs). https://anonymous.4open.science/r/mGKD-BAB1.

[5] M. J. Moyer, J. R. Rao, and P. Rohatgi, "A survey of security issues in multicast communications," *IEEE network*, vol. 13, no. 6, pp. 12–23, 1999.

[6] S. Rafaeli and D. Hutchison, "A Survey of Key Management for Secure Group Communication," *ACM Comput. Surv.*, vol. 35, no. 3, p. 309–329, sep 2003, https://doi.org/10.1145/937503.937506.

[7] P. S. Kruus, "A survey of multicast security issues and architectures," NAVAL RESEARCH LAB WASHINGTON DC, Tech. Rep., 1998.

[8] C. Boyd, A. Mathuria, and D. Stebila, *Protocols for Authentication and Key Establishment, Second Edition*, ser. Information Security and Cryptography. Springer, 2020.

[9] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*. CRC press, 2018.

[10] B. Poettering, P. Rösler, J. Schwenk, and D. Stebila, "SoK: Game-based Security Models for Group Key Exchange," Cryptology ePrint Archive, Paper 2021/305, 2021, https://eprint.iacr.org/2021/305.

[11] H. Harney and C. Muckenhirn. Group Key Management Protocol (GKMP) Specification. https://www.rfc-editor.org/rfc/rfc2093.

[12] ——. Group Key Management Protocol (GKMP) Architecture. https://www.rfc-editor.org/rfc/rfc2094.

[13] C. K. Wong, M. Gouda, and S. S. Lam, "Secure group communications using key graphs," *IEEE/ACM transactions on networking*, vol. 8, no. 1, pp. 16–30, 2000.

[14] A. T. Sherman and D. A. McGrew, "Key establishment in large dynamic groups using one-way function trees," *IEEE transactions on Software Engineering*, vol. 29, no. 5, pp. 444–458, 2003.

[15] J. Goshi and R. E. Ladner, "Algorithms for dynamic multicast key distribution trees," in *Proceedings of the twenty-second annual symposium on Principles of distributed computing*, 2003, pp. 243–251.

[16] L. Xu and C. Huang, "Computation-efficient multicast key distribution," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 5, pp. 577–587, 2008.

[17] H. Lu, "A novel high-order tree for secure multicast key management," *IEEE Transactions on Computers*, vol. 54, no. 2, pp. 214–224, 2005.

[18] Z. Liu, Y. Lai, X. Ren, and S. Bu, "An efficient LKH tree balancing algorithm for group key management," in *2012 International Conference on Control Engineering and Communication Technology*. IEEE, 2012, pp. 1003–1005.

[19] A. Perrig, "Efficient collaborative key management protocols for secure autonomous group communication," in *International Workshop on Cryptographic Techniques and E-Commerce (CrypTEC'99)*, 1999, pp. 192–202.

[20] Y. Kim, A. Perrig, and G. Tsudik, "Simple and fault-tolerant key agreement for dynamic collaborative groups," in *Proceedings of the 7th ACM Conference on Computer and Communications Security*, 2000, pp. 235–244.

[21] K. Cohn-Gordon, C. Cremers, L. Garratt, J. Millican, and K. Milner, "On Ends-to-Ends Encryption: Asynchronous group messaging with strong security guarantees," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 1802–1819.

[22] K. Klein, G. Pascual-Perez, M. Walter, C. Kamath, M. Capretto, M. Cueto, I. Markov, M. Yeo, J. Alwen, and K. Pietrzak, "Keep the dirt: tainted treekem, adaptively and actively secure continuous group key agreement," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 268–284.

[23] J. Alwen, S. Coretti, Y. Dodis, and Y. Tselekounis, "Security analysis and improvements for the IETF MLS standard for group messaging," in *Advances in Cryptology–CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part I*. Springer, 2020, pp. 248–277.

[24] R. Barnes, B. Beurdouche, R. Robert, J. Millican, E. Omara, and K. Cohn-Gordon. The Messaging Layer Security (MLS) Protocol. https://datatracker.ietf.org/doc/html/draft-ietf-mls-protocol-20.

[25] B. Haase and B. Labrique, "AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2019, no. 2, p. 1–48, Feb. 2019, https://tches.iacr.org/index.php/TCHES/article/view/7384.

[26] M. Abdalla, B. Haase, and J. Hesse, "Security analysis of CPace," in *Advances in Cryptology–ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6–10, 2021, Proceedings, Part IV*. Springer, 2021, pp. 711–741.

[27] M. Abdalla and D. Pointcheval, "Simple password-based encrypted key exchange protocols," in *Topics in Cryptology–CT-RSA 2005: The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005. Proceedings*. Springer, 2005, pp. 191–208.

[28] M. Abdalla and M. Barbosa, "Perfect Forward Security of SPAKE2," Cryptology ePrint Archive, Paper 2019/1194, 2019, https://eprint.iacr.org/2019/1194.

[29] M. Abdalla, M. Barbosa, T. Bradley, S. Jarecki, J. Katz, and J. Xu, "Universally Composable Relaxed Password Authenticated Key Exchange," Cryptology ePrint Archive, Paper 2020/320, 2020, https://eprint.iacr.org/2020/320.

[30] E. Bresson, O. Chevassut, and D. Pointcheval, "Group Diffie-Hellman key exchange secure against dictionary attacks," in *Advances in Cryptology—ASIACRYPT 2002: 8th International Conference on the Theory and Application of Cryptology and Information Security Queenstown, New Zealand, December 1–5, 2002 Proceedings 8*. Springer, 2002, pp. 497–514.

[31] M. Abdalla, E. Bresson, O. Chevassut, and D. Pointcheval, "Password-based group key exchange in a constant number of rounds," in *Public Key Cryptography-PKC 2006: 9th International Conference on Theory and Practice in Public-Key Cryptography, New York, NY, USA, April 24-26, 2006. Proceedings 9*. Springer, 2006, pp. 427–442.

[32] M. Abdalla and D. Pointcheval, "A scalable password-based group key exchange protocol in the standard model," in *ASIACRYPT*, vol. 4284. Springer, 2006, pp. 332–347.

[33] M. Abdalla, J.-M. Bohli, M. I. G. Vasco, and R. Steinwandt, "(Password) authenticated key establishment: From 2-party to group," in *Theory of Cryptography: 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007. Proceedings 4*. Springer, 2007, pp. 499–514.

[34] Y. Dodis, D. Jost, B. Kesavan, and A. Marcedone, "End-to-End Encrypted Zoom Meetings: Proving Security and Strengthening Liveness," in *Advances in Cryptology–EUROCRYPT 2023: 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, Proceedings*. Springer-Verlag, 2023.

[35] J. Blum, S. Booth, B. Chen, O. Gal, M. Krohn, J. Len, K. Lyons, A. Marcedone, M. Maxim, M. E. Mou, A. Namavari, J. O'Connor, S. Rien, M. Steele, M. Green, L. Kissner, and A. Stamos. Zoom end-to-end encryption whitepaper. https://github.com/zoom/zoom-e2e-whitepaper Version 4.0 (Released on 18.11.2022).

[36] Q. Dang, *Recommendation for Applications Using Approved Hash Algorithms*. Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD, 2012-08-24 2012, https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=911479 (Accessed Feb 2023).

[37] J. Brendel, C. Cremers, D. Jackson, and M. Zhao, "The provable security of Ed25519: theory and practice," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 1659–1676.

[38] S. Arciszewski, "XChaCha: eXtended-nonce ChaCha and AEAD_XChaCha20_Poly1305," Internet Engineering Task Force, Internet-Draft draft-irtf-cfrg-xchacha-03, Jan. 2020, work in Progress: https://datatracker.ietf.org/doc/draft-irtf-cfrg-xchacha/03/.

[39] J. R. Vacca, *Public key infrastructure: building trusted applications and Web services*. Auerbach Publications, 2004.

[40] P. Eckersley and J. Burns, "The (Decentralized) SSL Observatory," in *20th USENIX Security Symposium (USENIX Security 11)*. San Francisco, CA: USENIX Association, Aug. 2011, https://www.usenix.org/conference/usenix-security-11/decentralized-ssl-observatory.

[41] J. Stapleton, "PKI Under Attack," *ISSA*, vol. 11, no. 3, 2013, https://cdn.ymaws.com/www.members.issa.org/resource/resmgr/JournalPDFs/PKI_Under_Attack_ISSA0313.pdf.

[42] M. Bellare and V. T. Hoang, "Efficient schemes for committing authenticated encryption," in *Advances in Cryptology–EUROCRYPT 2022: 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30–June 3, 2022, Proceedings, Part II*. Springer, 2022, pp. 845–875.

[43] P. Rogaway, "Authenticated-Encryption with Associated-Data," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, ser. CCS '02, 2002, p. 98–107.

[44] G. Barwell, D. Page, and M. Stam, "Rogue decryption failures: Reconciling AE robustness notions," in *Cryptography and Coding: 15th IMA International Conference, IMACC 2015, Oxford, UK, December 15-17, 2015. Proceedings 15*. Springer, 2015, pp. 94–111.

[45] J. Brendel, M. Fischlin, F. Günther, and C. Janson, "PRF-ODH: Relations, instantiations, and impossibility results," in *Advances in Cryptology–CRYPTO 2017: 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20–24, 2017, Proceedings, Part III 37*. Springer, 2017, pp. 651–681.

[46] D. J. Bernstein, T. Lange, and P. Schwabe. NaCl: Networking and Cryptography library. https://nacl.cr.yp.to/, (Accessed Jan 2023).

[47] F. Denis. The Sodium cryptography library. https://doc.libsodium.org/, (Accessed Jan 2023).

Appendix A. Preliminaries

Due to the page limit, we only introduce novel and customized security notions here and recall other notions in our full version [4].

A.1. Digital Signature

Definition 9. *A digital signature scheme over message space $\mathcal{M}$ is a tuple of algorithms* $\mathsf{DS} = (\mathsf{DS.KG}, \mathsf{DS.Sign}, \mathsf{DS.Vrfy})$ *as defined below.*

- ***Key Generation*** $(\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{DS.KG}(\mathsf{pp})$*: inputs the public parameter* pp *and outputs a public verification and private signing key pair* $(\mathsf{vk}, \mathsf{sk})$.
- ***Signing*** $\sigma \xleftarrow{\$} \mathsf{DS.Sign}(\mathsf{sk}, m)$*: inputs a signing key* sk *and a message* $m \in \mathcal{M}$ *and outputs a signature* σ.
- ***Verification*** $\mathsf{true}/\mathsf{false} \leftarrow \mathsf{DS.Vrfy}(\mathsf{vk}, m, \sigma)$*: inputs a verification key* vk*, a message* m*, and a signature* σ *and outputs a boolean value either true* true *or* false.

We say a DS is δ-correct if for every $(\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{DS.KG}()$ and every message $m \in \mathcal{M}$, we have

$$\Pr[\mathsf{false} \leftarrow \mathsf{DS.Vrfy}(\mathsf{vk}, m, \mathsf{DS.Sign}(\mathsf{sk}, m))] \leq \delta$$

In particular, we call a DS *(perfectly) correct* if $\delta = 0$.

In terms of the security notations, we recall the standard *existential unforgeability against chosen message attack* $\mathsf{EUF\text{-}CMA}$.

Definition 10. *Let* $\mathsf{DS} = (\mathsf{DS.KG}, \mathsf{DS.Sign}, \mathsf{DS.Vrfy})$ *be a digital signature scheme with message space* $\mathcal{M}$*. We say* DS *is* ϵ-$\mathsf{euf\text{-}cma}$ *secure, if for every* PPT *attacker* $\mathcal{A}$*, we have*

$$\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{DS}}(\mathcal{A}) := \Pr[\mathsf{Expr}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{DS}}(\mathcal{A}) = 1] \leq \epsilon$$

where the experiment $\mathsf{Expr}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{DS}}(\mathcal{A})$ *is defined in Figure 2.*

$\mathsf{Expr}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{DS}}(\mathcal{A})$:
1 $\mathcal{L} \leftarrow \emptyset$
2 $(\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{DS.KG}()$
3 $(m^\star, \sigma^\star) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Sign}}}(\mathsf{vk})$
4 **if** $m^\star \in \mathcal{L}$
5 **return** 0
6 **return** $[\![\mathsf{DS.Vrfy}(\mathsf{vk}, m^\star, \sigma^\star)]\!]$

$\mathcal{O}_{\mathsf{Sign}}(m)$:
7 $\sigma \xleftarrow{\$} \mathsf{DS.Sign}(\mathsf{sk}, m)$
8 $\mathcal{L} \xleftarrow{+} m$
9 **return** σ

Figure 2. $\mathsf{EUF\text{-}CMA}$ experiment for $\mathsf{DS} = (\mathsf{DS.KG}, \mathsf{DS.Sign}, \mathsf{DS.Vrfy})$.

A.2. Authenticated Encryption with Associated Data

Definition 11 ([43])**.** *Let* Key, Nonce, Data, $\mathsf{Message}$, $\mathsf{Ciphertext}$ *respectively denote the space of keys, nonces, associated data, messages, and ciphertexts. An authenticated encryption with associated data scheme* $\mathsf{AEAD} = (\mathsf{AEAD.Enc}, \mathsf{AEAD.Dec})$ *is a tuple of algorithms where*

- $\mathsf{AEAD.Enc}$ *the encryption algorithm inputs a key* $k \in \mathsf{Key}$*, a nonce* $\mathsf{nonce} \in \mathsf{Nonce}$*, an associated data* $D \in \mathsf{Data}$*, and*

a message m and (deterministically) outputs a ciphertext c, i.e., $c \leftarrow \mathsf{AEAD.Enc}(k, \mathsf{nonce}, D, m)$.

- $\mathsf{AEAD.Dec}$ *the decryption algorithm inputs a key $k \in \mathsf{Key}$, a nonce $\mathsf{nonce} \in \mathsf{Nonce}$, an associated data $D \in \mathsf{Data}$, and a ciphertext $c \in \mathsf{Ciphertext}$ and deterministically outputs a message $m \in \mathsf{Message} \cup \{\perp\}$, i.e., $m \leftarrow \mathsf{AEAD.Dec}(k, \mathsf{nonce}, D, c)$.*

Definition 12. *We say an* AEAD *is* ϵ-ind$-cca *(resp.* cti-cpa*) secure, if the below defined advantage of any* PPT *attacker $\mathcal{A}$ against $\mathsf{Expr}^{\mathsf{IND\$\text{-}CCA}}_{\mathsf{AEAD}}$ (resp. $\mathsf{Expr}^{\mathsf{CTI\text{-}CPA}}_{\mathsf{AEAD}}$) experiment in Figure 3 is bounded by,*

$$\mathsf{Adv}^{\mathsf{IND\$\text{-}CCA}}_{\mathsf{AEAD}} := |\Pr[\mathsf{Expr}^{\mathsf{IND\$\text{-}CCA}}_{\mathsf{AEAD}}(\mathcal{A}) = 1] - \frac{1}{2}| \leq \epsilon$$

$$\mathsf{Adv}^{\mathsf{CTI\text{-}CPA}}_{\mathsf{AEAD}} := \Pr[\mathsf{Expr}^{\mathsf{CTI\text{-}CPA}}_{\mathsf{AEAD}}(\mathcal{A}) = 1] \leq \epsilon$$

Below, we define two customized notions. The (n, m)-frob notion ensures that each ciphertext cannot be decrypted two distinct messages upon different nonces. The d-frob security ensures that each ciphertext cannot be decrypted to two valid messages upon different associated data. They both are implied by the CMT-4 security in [42].

Definition 13. *We say an* AEAD *has* ϵ-(n, m)-frob *(resp.* d-frob*) secure, if the below defined advantage of any* PPT *attacker $\mathcal{A}$ against $\mathsf{Expr}^{(\mathsf{n},\mathsf{m})\text{-}\mathsf{FROB}}_{\mathsf{AEAD}}$ (resp. $\mathsf{Expr}^{\mathsf{d\text{-}FROB}}_{\mathsf{AEAD}}$) experiment in Figure 4 is bounded by,*

$$\mathsf{Adv}^{(\mathsf{n},\mathsf{m})\text{-}\mathsf{FROB}}_{\mathsf{AEAD}} := \Pr[\mathsf{Expr}^{(\mathsf{n},\mathsf{m})\text{-}\mathsf{FROB}}_{\mathsf{AEAD}}(\mathcal{A}) = 1] \leq \epsilon$$

$$\mathsf{Adv}^{\mathsf{d\text{-}FROB}}_{\mathsf{AEAD}} := \Pr[\mathsf{Expr}^{\mathsf{d\text{-}FROB}}_{\mathsf{AEAD}}(\mathcal{A}) = 1] \leq \epsilon$$

A.3. Hash Functions and Pseudorandom Functions

Definition 14. *We say a function $\mathsf{H} : \{0,1\}^* \to \{0,1\}^l$ for some length l is ϵ-collision resistant if for any* PPT *attacker $\mathcal{A}$, the probability that $\mathcal{A}$ can produce two distinct string m_1 and m_2 such that $\mathsf{H}(m_1) = \mathsf{H}(m_2)$ is bounded by ϵ.*

A.4. lr-prf-ODH Assumption

We recall the generic lr-prf-ODH definition in [45].

Definition 15. *Let* ECDH *denote a cyclic group G with order q over an elliptic curve EC with a generator g. Let $\mathsf{H} : \mathbb{G} \times \{0,1\}^\star \to \{0,1\}^{l_\mathsf{H}}$ denote a function. We define a generic security notion* lr-prf-ODH *which is parameterized by $\mathsf{l}, \mathsf{r} \in \{\mathsf{n}, \mathsf{s}, \mathsf{m}\}$ indicating how often the attacker is allowed to query a certain "left" resp. "right" oracle (*ODH_u *resp.* ODH_v*) where* n *indicates that no query is allowed,* s *that a single query is allowed, and* m *that multiple (polynomially many) queries are allowed to the respective side. Consider the following security game $\mathsf{Expr}^{\mathsf{lr\text{-}prf\text{-}ODH}}_{\mathsf{ECDH},\mathsf{H}}$ between a challenger $\mathcal{C}$ and a* PPT *attacker $\mathcal{A}$.*

1) *The challenger $\mathcal{C}$ samples $u \xleftarrow{\$} \mathbb{Z}_q$ and provides $\mathbb{G}$, g, and g^u to the attacker $\mathcal{A}$.*
2) *If $\mathsf{l} = \mathsf{m}$, $\mathcal{A}$ can issue arbitrarily many queries to the following oracle* ODH_u.

   ODH_u **oracle.** *On a query of the form (S, x), $\mathcal{C}$ first checks if $S \notin \mathbb{G}$ and returns $\perp$ if this is the case. Otherwise, it computes $y \leftarrow \mathsf{H}(S^u, x)$ and returns y.*
3) *Eventually, $\mathcal{A}$ issues a challenge query $x^\star$. On this query, $\mathcal{C}$ samples $v \xleftarrow{\$} \mathbb{Z}_q$ and a bit $\mathsf{b} \xleftarrow{\$} \{0,1\}$ uniformly at random. It then computes $y^\star_0 = \mathsf{H}(g^{uv}, x^\star)$ and samples $y^\star_q \xleftarrow{\$} \{0,1\}^{l_\mathsf{H}}$ uniformly at random. The challenger returns $(g^v, y^\star_\mathsf{b})$ to $\mathcal{A}$.*
4) *Next, $\mathcal{A}$ may issue (arbitrarily interleaved) queries to the following oracles* ODH_u *and* ODH_v *(depending on* l *and* r*).*

   ODH_u **oracle.** *The attacker $\mathcal{A}$ may ask no ($\mathsf{l} = \mathsf{n}$), a single ($\mathsf{l} = \mathsf{s}$), or arbitrarily many ($\mathsf{l} = \mathsf{m}$) queries to this oracle. On a query of the form (S, x), the challenger first checks if $S \notin G$ or $(S, x) = (g^v, x^\star)$ and returns $\perp$ if this is the case. Otherwise, it computes $y \leftarrow \mathsf{H}(S^u, x)$ and returns y.*

   ODH_v **oracle.** *The attacker $\mathcal{A}$ may ask no ($\mathsf{r} = \mathsf{n}$), a single ($\mathsf{r} = \mathsf{s}$), or arbitrarily many ($\mathsf{r} = \mathsf{m}$) queries to this oracle. On a query of the form (T, x), the challenger first checks if $T \notin G$ or $(T, x) = (g^u, x^\star)$ and returns $\perp$ if this is the case. Otherwise, it computes $y \leftarrow \mathsf{H}(T^v, x)$ and returns y.*
5) *At some point, $\mathcal{A}$ stops and outputs a guess $\mathsf{b}' \in \{0,1\}$.*

We say that the attacker wins the lr-prf-ODH *game if $\mathsf{b}' = \mathsf{b}$. We say the* lr-prf-ODH *problem is ϵ-hard over* ECDH *and* H*, if the advantage of any* PPT *attacker $\mathcal{A}$ that wins above $\mathsf{Expr}^{\mathsf{lr\text{-}prf\text{-}ODH}}_{\mathsf{ECDH},\mathsf{H}}$ experiment is bounded by ϵ.*

In this paper, we only need the mn-prf-ODH assumption.

A.5. Password-Authenticated Key Exchange

The password-authenticated key exchange (PAKE) protocols allow two parties to establish a high-entropy key over an insecure channel using a shared low-entropy password. Below, we first define a weak security (w-PAKE) security model against a PAKE protocol. This model is weaker than and therefore implied by the security model defined in [26], [28], [29]. Thus, some modern and widely used PAKE schemes, including CPace [25] or SPAKE2 [27] that are respectively proven secure in [26] and [28], [29], are provably secure in this w-PAKE model.

Protocol Members. This weak semantic security model only considers the two-party setting. I.e., the PAKE protocol has only two members: either an initiator I or a responder R.

The initiator I indeed captures the behaviors of (all) participants in each group gid in our mGKD protocol. The responder R indeed captures the behaviors of the (unique) leader in each group gid in our mGKD protocol.

Long-Lived Keys / Passwords. The initiator I and the responder R hold the same password pw, which is sampled from a distribution $\mathcal{D}$. In many literature, the password is also called the "long-lived key".

Protocol Execution. The interaction between an attacker $\mathcal{A}$ and the protocol members occurs only via oracle queries,

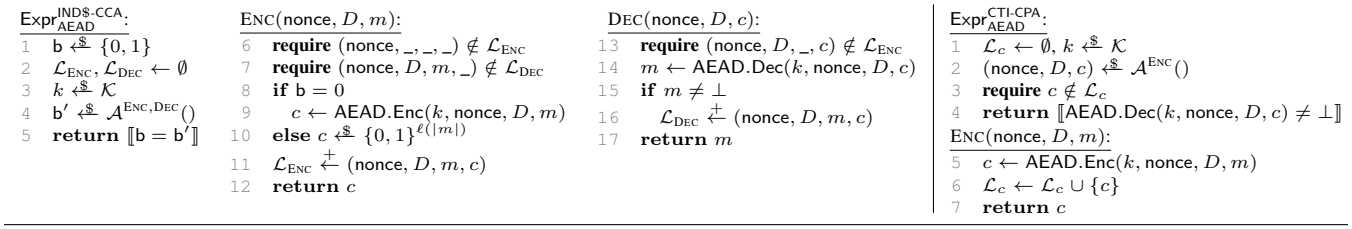

Figure 3. IND$-CCA security [44] and CTI-CPA security [43] for an AEAD scheme. The functi on ℓ is a function that inputs the message length and outputs the corresponding ciphertext length.

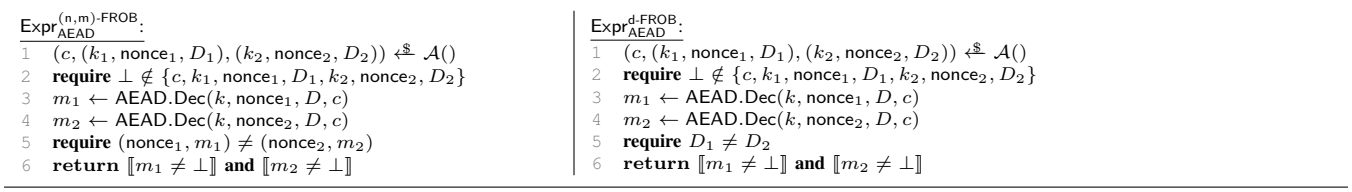

Figure 4. (n, m)-FROB security and d-FROB security for an AEAD scheme.

which model the attacker capabilities in a real attack. During the execution, the attacker may create several instances of a member. We consider the concurrent model, i.e., several instances may be active at any given time. Let U^{id} denote the instance with identifier id of a member U. Let $\mathsf{b} \in \{0,1\}$ be a bit chosen uniformly at random. The attacker $\mathcal{A}$ can query following three oracles:

- SENDPAKE(U^{id}, m): This query models an active attack, in which the adversary may tamper with the message being sent over the public channel. The output of this query is the message that the member instance U^{id} would generate upon receipt of message m.
- COMPROMISEPAKE(U^{id}): This query models the misuse of session keys by a member. If a session key is not defined for instance U^{id} or if a TESTPAKE query was asked to either U^{id} or to its partner, then return $\perp$. Otherwise, return the session key held by the instance U^{id}.
- TESTPAKE(U^{id}): This query tries to capture the adversary's ability to tell apart a real session key from a random one. If no session key for instance U^{id} is defined or U^{id} is not fresh (which is defined below), then return the undefined symbol $\perp$. Otherwise, return the session key for instance U^{id} if $\mathsf{b} = 1$ or a random key of the same size if $\mathsf{b} = 0$.

Notation. We say an instance U^{id} *opened* if a query COMPROMISEPAKE(U^{id}) has been made by the attacker. We say an instance U^{id} is *unopened* if it is not opened. We say an instance U^{id} *tested* if a query TESTPAKE(U^{id}) has been made by the attacker. We say an instance U^{id} is *untested* if it is not tested. We say an instance U^{id} has *accepted* if it goes into an accept mode after receiving the last expected protocol message.

Session Identifiers and Partnering. We define the *session identifiers* (sid) as the transcript of the conversation between the initiator and the responder instances before acceptance.

We say two instances I^{id_1} and R^{id_2} to be *partners* if the following conditions are met:

(a) Both I^{id_1} and R^{id_2} accept, and
(b) Both I^{id_1} and R^{id_2} share the same identifiers sid.

Freshness. The notion of freshness is defined to avoid cases in which adversary can trivially break the security of the scheme. The goal is to only allow the attacker to ask TESTPAKE queries to fresh oracle instances. More specifically, we say an instance U^{id} is fresh if it has accepted and if both U^{id} and its partner are unopened and untested.

Semantic Security. Consider an execution of the above experiment for an attacker $\mathcal{A}$ against a PAKE protocol Π. The attacker $\mathcal{A}$ wins the experiment if and only if $\mathcal{A}$ guesses $\mathsf{b}' = \mathsf{b}$, where b is the hidden bit used by the TESTPAKE oracle. The advantage of $\mathcal{A}$ breaking the weak security of Π is defined as

$$\mathsf{Adv}^{\mathsf{w\text{-}PAKE}}_{\Pi,\mathcal{D}}(\mathcal{A}) := |\Pr[\mathcal{A} \text{ wins}] - \frac{1}{2}|$$

We say that Π is $\epsilon^{\mathsf{w\text{-}PAKE}}_{\mathsf{PAKE},\mathcal{D}}$-w-PAKE secure if for any PPT attackers $\mathcal{A}$ it always holds that

$$\mathsf{Adv}^{\mathsf{w\text{-}PAKE}}_{\Pi,\mathcal{D}}(\mathcal{A}) \leq \epsilon^{\mathsf{w\text{-}PAKE}}_{\Pi,\mathcal{D}}$$

Appendix B. Cryptographic Algorithms

The Zoom library makes use of the interface and implementation of two building blocks in the NaCl [46]-inspired libsodium library [47]: *Signing* and *Authenticated Public-Key Encryption* (aka. Box).

Zoom Signing Algorithm: The construction of the Zoom Signing algorithm ZSign = (ZSign.KG, ZSign.Sign, ZSign.Vrfy) is depicted in Figure 5.

- The key generation algorithm ZSign.KG simply generates and outputs a DS key pair.
- The signing algorithm ZSign.Sign inputs a secret key sk_{ZSign}, a context ctxt, and a message m. The ZSign.Sign

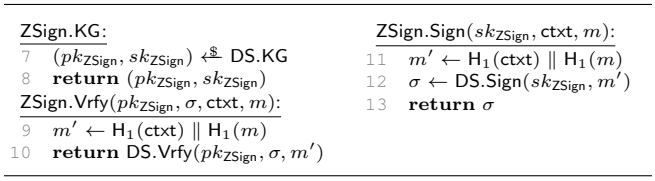

Figure 5. The Zoom-Signing algorithm ZSign. Zoom instantiates H_1 with SHA256 and DS with EdDSA over Ed25519.

first computes the hash function H_1 over respective context ctxt and message m, followed by concatenating them. Then, the ZSign.Sign computes and outputs the signature of the concatenation using DS upon the input secret key sk_{ZSign}.
- The verification ZSign.Vrfy algorithm inputs a public key pk_{ZSign}, a signature σ, a context ctxt, and a message m. This ZSign.Vrfy algorithm simply computes the concatenation as in the signing algorithm and outputs the DS verification result DS.Vrfy upon the public key pk_{ZSign}, the signature σ, and the concatenation.

Zoom Authenticated Public-Key Encryption (aka. Box) Algorithm: The Zoom Box algorithm $\mathsf{ZBox} = (\mathsf{ZBox.KG}, \mathsf{ZBox.Enc}, \mathsf{ZBox.Dec})$ is depicted in Figure 6.
- The key generation algorithm ZBox.KG samples and outputs a Diffie-Hellman key pair over an elliptic curve ECDH.
- The encryption algorithm ZBox.Enc takes as inputs a sender's secret key $sk^{\mathsf{S}}_{\mathsf{ZBox}}$, a receiver's public key $pk^{\mathsf{R}}_{\mathsf{ZBox}}$, two contexts $\mathsf{ctxt}_{\mathsf{H}_2}$ and $\mathsf{ctxt}_{\mathsf{Cipher}}$, a meta data Meta, and a message m. It first samples a random nonce of bit length l. Next, it computes the Diffie-Hellman exchange of $sk^{\mathsf{S}}_{\mathsf{ZBox}}$ and $pk^{\mathsf{R}}_{\mathsf{ZBox}}$, which is combined with the context $\mathsf{ctxt}_{\mathsf{H}_2}$ and used as input to the hash function H_2 for a key K. Then, it computes the associated data D by concatenating $\mathsf{H}_1(\mathsf{ctxt}_{\mathsf{Cipher}})$ and $\mathsf{H}_1(\mathsf{Meta})$. The output is the nonce nonce and an AEAD ciphertext produced from the key K, nonce nonce, data D, and message m.
- The decryption algorithm ZBox.Dec takes as input a receiver's secret key $sk^{\mathsf{R}}_{\mathsf{ZBox}}$, a sender's public key $pk^{\mathsf{S}}_{\mathsf{ZBox}}$, two contexts $\mathsf{ctxt}_{\mathsf{H}_2}$ and $\mathsf{ctxt}_{\mathsf{Cipher}}$, a meta data Meta, and a ciphertext c. This algorithm parses the nonce nonce from the ciphertext c, computes the key K and the associated data D as in the encryption algorithm, and executes the AEAD decryption. If the AEAD outputs a message $m \neq \perp$, this algorithm simply outputs this m, and aborts otherwise.

Appendix C. Proof of Theorem 1

We give the proof of Zoom's security in Sec-mGKD-pki as a sequence of games. Let $\mathsf{Adv}_i(\mathcal{A})$ denote the advantage of an attacker $\mathcal{A}$ in winning **Game** i.

Game 0: This game is identical to the original Sec-mGKD-pki experiment. Thus, we have that

$$\mathsf{Adv}_0(\mathcal{A}) = \mathsf{Adv}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}_{\Pi}(\mathcal{A})$$

Game 1: This game is identical to **Game** 0, except that the challenger $\mathcal{C}$ let the attacker $\mathcal{A}$ immediately win if there exists collision on function H_1. That is, there exists two distinct inputs m_1 and m_2 such that $\mathsf{H}_1(m_1) = \mathsf{H}_1(m_2)$. By this, we ensure that there exists no collision on the function H_1 in the following games. Due to the collision resistance of H_1, we can easily have that:

$$\mathsf{Adv}_0(\mathcal{A}) \leq \mathsf{Adv}_1(\mathcal{A}) + \epsilon^{\mathsf{coll\text{-}res}}_{\mathsf{H}_1}$$

Game 2: This game is identical to the **Game** 1 except the following modification:
- The challenger $\mathcal{C}$ aborts the game if $\mathcal{A}$ can trigger the following event:

  Event E_1**:** There exists any party P^1, any party P^2, and any group identifier gid such that
  - the long-term state st_{P^1} is not corrupted before P^1 and P^2 both joined the group gid,
  - the sign-up messages, i.e., the identity public keys $\mathsf{st}_{P^1}.ipk = ipk_{P^1}$ of P^1 is honestly delivered to P^2 in the Participant Join phase between P^1 and P^2 for the group gid, and
  - P^1 and P^2 have disagreement on the binding information $\mathsf{Binding}^{\mathsf{gid}}_{P^1}$.

By this, we ensure that in the following games, if the sign-up message of a party P^1 is delivered to another party P^2 before the corruption of the long-term state st_{P^1} in any group gid, then P^2 and P^1 must agree on the Binding information $\mathsf{Binding}^{\mathsf{gid}}_{P^1}$.

Obviously, it holds that

$$\mathsf{Adv}_1 \leq \mathsf{Adv}_2(\mathcal{A}) + \Pr[E_1]$$

Below, we analyze the probability that $\mathcal{A}$ can trigger E_1 by reduction. If the attacker $\mathcal{A}$ can trigger the event E_1, then we construct an attacker $\mathcal{B}_1$ that breaks the euf-cma security of the underlying DS scheme. The attacker $\mathcal{B}_1$ receives a public verification key $\mathsf{vk}^\star$ and honestly initializes **Game** 1. Moreover, $\mathcal{B}_1$ guesses the index $i^\star$of one NEWPARTY query that will create the party P^1 in the event E_1. Note that there are at most NEWPARTY queries in the game. $\mathcal{B}_1$ guesses correctly with probability at least $\frac{1}{q_{\text{NEWPARTY}}}$. Then, $\mathcal{B}_1$ honestly answers $\mathcal{A}$'s queries except the following ones:
- NEWPARTY(id_P): If this is the $i^\star$-th query, then $\mathcal{B}_1$ initializes a state st_P by setting $\mathsf{st}_P.\mathsf{id} \leftarrow \mathsf{id}_P$. Then, $\mathcal{C}$ sets $\mathsf{st}_P.ipk$ to $\mathsf{vk}^\star$ that is given by its challenger. Finally, $\mathcal{B}_1$ forwards $m^P_{\mathsf{SignUp}} = \mathsf{vk}^\star$ to $\mathcal{A}$ and marks P as "created". For other queries to this oracle, $\mathcal{B}_1$ executes them honestly.
- REGISTERAUTH$(\mathsf{id}_P, \mathsf{gid}, m)$: If the input party P is created via the $i^\star$-th query to the NEWPARTY oracle, then $\mathcal{B}_1$ honestly executes the checks. If no error occurs, $\mathcal{B}_1$ honestly produces its binding information $\mathsf{Binding}^{\widetilde{\mathsf{gid}}}_P$ and send $\mathsf{H}_1(\mathsf{ctxt}_1) \parallel \mathsf{H}_1(\mathsf{Binding}^{\widetilde{\mathsf{gid}}}_P)$ to its DS signing oracle. Then, $\mathcal{B}_1$ receives a signature $\sigma^{\widetilde{\mathsf{gid}}}_P$ and use it as the output of the ZSign signature. The rest of this query is honestly executed.
  For other queries to this oracle, $\mathcal{B}_1$ executes them honestly.
- REGISTERINJECT$(\mathsf{id}_P, \mathsf{gid}, gs, m)$: If the input party P is created via the $i^\star$-th query to the NEWPARTY oracle and

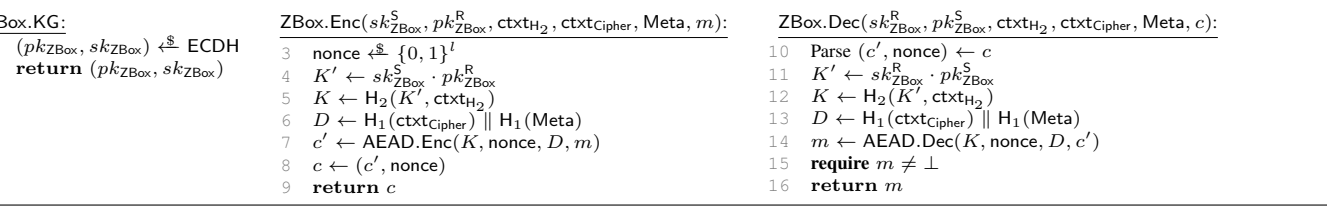

Figure 6. The Zoom-Box algorithm ZBox. We have that $l = 192$, and the underlying function H_1 denotes SHA256. The function H_2 denotes HKDF (using an empty salt parameter). ECDH is performed on Curve25519. "·" denotes scalar multiplication. The AEAD is instantiated with xchacha20poly1305.

$\mathsf{gid} = \widetilde{\mathsf{gid}}$, then $\mathcal{B}_1$ honestly executes the checks. If no error occurs, $\mathcal{B}_1$ honestly produces its binding information $\mathsf{Binding}^{\widetilde{\mathsf{gid}}}_P$ and send $\mathsf{H}_1(\mathsf{ctxt}_1) \| \mathsf{H}_1(\mathsf{Binding}^{\widetilde{\mathsf{gid}}}_P)$ to its DS signing oracle. Then, $\mathcal{B}_1$ receives a signature $\sigma^{\widetilde{\mathsf{gid}}}_P$ and use it as the output of the ZSign signature. The rest of this query is honestly executed.
For other queries to this oracle, $\mathcal{B}_1$ executes them honestly.

- SENDJOINAUTH($\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, m$): If the input party P' is created via the $i^\star$-th query to the NEWPARTY oracle, then $\mathcal{B}_1$ extracts the binding information $\mathsf{Binding}^{\widetilde{\mathsf{gid}}}_{P'}$ and a signature $\sigma^{\widetilde{\mathsf{gid}}}_{P'}$. If $\sigma^{\widetilde{\mathsf{gid}}}_{P'}$ is not output by any REGISTERAUTH or REGISTERINJECT oracle for the binding information $\mathsf{Binding}^{\widetilde{\mathsf{gid}}}_{P'}$ but the verification $\mathsf{DS.Vrfy}(\mathsf{vk}^\star, \mathsf{H}_1(\mathsf{ctxt}_1) \| \mathsf{H}_1(\mathsf{Binding}^{\widetilde{\mathsf{gid}}}_{P'}), \sigma^{\widetilde{\mathsf{gid}}}_{P'}) = \mathsf{true}$ passes, then $\mathcal{B}_1$ immediately returns $(\mathsf{H}_1(\mathsf{ctxt}_1) \| \mathsf{H}_1(\mathsf{Binding}^{\widetilde{\mathsf{gid}}}_{P'}), \sigma^{\widetilde{\mathsf{gid}}}_{P'})$ to its challenger and aborts the experiment.
In all other cases inside this query or for other queries to this oracle, $\mathcal{B}_1$ executes them honestly.
- SENDJOININJECT($\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, gs, m$): If the input party P is created via the $i^\star$-th query to the NEWPARTY oracle, then $\mathcal{B}_1$ extracts the binding information $\mathsf{Binding}^{\widetilde{\mathsf{gid}}}_{P'}$ and a signature $\sigma^{\widetilde{\mathsf{gid}}}_{P'}$. If $\sigma^{\widetilde{\mathsf{gid}}}_{P'}$ is not output by any REGISTERAUTH or REGISTERINJECT oracle for the binding information $\mathsf{Binding}^{\widetilde{\mathsf{gid}}}_{P'}$ but the verification $\mathsf{DS.Vrfy}(\mathsf{vk}^\star, \mathsf{H}_1(\mathsf{ctxt}_1) \| \mathsf{H}_1(\mathsf{Binding}^{\widetilde{\mathsf{gid}}}_{P'}), \sigma^{\widetilde{\mathsf{gid}}}_{P'}) = \mathsf{true}$ passes, then $\mathcal{B}_1$ immediately returns $(\mathsf{H}_1(\mathsf{ctxt}_1) \| \mathsf{H}_1(\mathsf{Binding}^{\widetilde{\mathsf{gid}}}_{P'}), \sigma^{\widetilde{\mathsf{gid}}}_{P'})$ to its challenger and aborts the experiment.
In all other cases inside this query or for other queries to this oracle, $\mathcal{B}_1$ executes them honestly.
- CORRUPT(id_P): If the input party P is created via the $i^\star$-th query to the NEWPARTY oracle, $\mathcal{B}_1$ aborts. Otherwise, $\mathcal{B}_1$ honestly executes this oracle.

If the attacker $\mathcal{A}$ can trigger the event E_1 and $\mathcal{B}_1$ guesses the oracle that creates party P^1 correctly, then $\mathcal{A}$ must trigger the event E_1 before querying the CORRUPT that causes the abortion. Moreover, there must also exist a group identifier gid and a party P^2 such that

1) P^2 receives the honest sign-up message $\mathsf{st}_{P^1}.ipk = ipk_{P^1}$ of the the party P^1 in the group gid,
2) the long-term state st_{P^1} is not corrupted before P^1 and P^2 joined the group gid, and
3) the parties P^1 and P^2 have disagreement on P^1's binding information $\mathsf{Binding}^{\mathsf{gid}}_{P^1}$.

This means, $\mathcal{B}_1$ can always win in the SENDJOINAUTH or SENDJOININJECT oracle.

Note also that the event "the attacker $\mathcal{A}$ can trigger event E_1" and the event "$\mathcal{B}_1$ guesses correctly" are independent. Thus

$$\begin{aligned}
&\epsilon^{\mathsf{euf\text{-}cma}}_{\mathsf{DS}} \\
&\geq \Pr[\mathcal{B}_1 \text{ wins}] \\
&\geq \Pr[\mathcal{A} \text{ can trigger event } E_1 \textbf{ and } \mathcal{B}_1 \text{ guesses correctly}] \\
&\geq \Pr[\mathcal{A} \text{ can trigger event }] \cdot \Pr[\mathcal{B}_1 \text{ guesses correctly}] \\
&\geq \Pr[E_1] \cdot \frac{1}{q_{\text{NEWPARTY}}}
\end{aligned}$$

The above equation can be rewritten as:

$$\Pr[E_1] \leq q_{\text{NEWPARTY}} \epsilon^{\mathsf{euf\text{-}cma}}_{\mathsf{DS}}$$

Thus, we have that

$$\mathsf{Adv}_1(\mathcal{A}) \leq \mathsf{Adv}_2(\mathcal{A}) + q_{\text{NEWPARTY}} \epsilon^{\mathsf{euf\text{-}cma}}_{\mathsf{DS}}$$

Game 3: This game is identical to **Game** 2 except the at the beginning of the experiment the challenger $\mathcal{C}$ guesses a group identifier $\widetilde{\mathsf{gid}}$, whose associated leader is denoted by $P^{\widetilde{\mathsf{gid}}}$, that will lead $\mathcal{A}$ to win, and lets $\mathcal{A}$ immediately lose if the guess is wrong. By this, we ensure that the attacker $\mathcal{A}$ can win only by triggering one of the events as follows:

- [In the case of event E_{KAuth}] there exists any party P' that is authorized for the group $\widetilde{\mathsf{gid}}$ and any group key index gkid such that $gk^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}_{P'} \neq \perp$ but $gk^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}_{P^{\widetilde{\mathsf{gid}}}} \neq gk^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}_{P'}$, without the violation of the freshness condition $\mathsf{fresh}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}_{\mathsf{KAuth}}(\mathsf{id}_{P'}, \widetilde{\mathsf{gid}}, \mathsf{gkid})$, or
- [In the case of event E_{KPriv}] $\mathcal{A}$ will query TEST oracle with input $(\mathsf{id}_{P'}, \widetilde{\mathsf{gid}}, \mathsf{gkid})$ for some party identifier P' and some group key index gkid and correctly guess the challenge bit $\mathsf{b} = \mathsf{b}'$, without violation of the freshness $\mathsf{fresh}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}_{\mathsf{KPriv}}(\mathsf{id}_{P'}, \widetilde{\mathsf{gid}}, \mathsf{gkid})$.

Note that each group must be created via NEWGROUP oracle. There are at most q_{NEWGROUP} groups in the experiment. Thus, the guess is correct with probability at least $\frac{1}{q_{\text{NEWGROUP}}}$. Note that whether $\mathcal{A}$ wins in **Game** 2 and whether the challenger guesses correctly in **Game** 3 is independent. We have that

$$\mathsf{Adv}_2(\mathcal{A}) \leq q_{\text{NEWGROUP}} \mathsf{Adv}_3(\mathcal{A})$$

Below, we analyze the advantage that $\mathcal{A}$ wins in **Game** 3 by case distinction, i.e., whether $\mathcal{A}$ wins by triggering

E_{KAuth} in Case 1, the advantage of which is denoted by Adv_3^{C1}, or by triggering E_{KPriv} in Case 2, the advantage of which is denoted by Adv_3^{C2}. Thus, we have that

$$\mathsf{Adv}_3(\mathcal{A}) := \max\left(\mathsf{Adv}_3^{C1}(\mathcal{A}), \mathsf{Adv}_3^{C2}(\mathcal{A})\right)$$

Case 1: $\mathcal{A}$ wins by triggering event E_{KAuth}.

In this case, due to the winning conditions and the freshness requirement, we know that for the group identifier $\widetilde{\mathsf{gid}}$ with a leader denoted by $P^{\widetilde{\mathsf{gid}}}$, there must exist an authorized party P' and a group key index gkid such that:

1) $gk_{P'}^{(\widetilde{\mathsf{gid}},\mathsf{gkid})} \neq \perp$ and $gk_{P^{\widetilde{\mathsf{gid}}}}^{(\widetilde{\mathsf{gid}},\mathsf{gkid})} \neq gk_{P'}^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}$,
2) neither $\pi_{P'}^{\widetilde{\mathsf{gid}}}$ nor $\pi_{P^{\widetilde{\mathsf{gid}}}}^{\widetilde{\mathsf{gid}}}$ is compromised,
3) the long-term states $\mathsf{st}_{P'}$ and $\mathsf{st}_{P^{\widetilde{\mathsf{gid}}}}$ are not corrupted before P' and $P^{\widetilde{\mathsf{gid}}}$ both joined the group $\widetilde{\mathsf{gid}}$, and
4) the sign-up messages, i.e., the identity public keys $\mathsf{st}_{P'}.ipk = ipk_{P'}$ and $\mathsf{st}_{P^{\widetilde{\mathsf{gid}}}}.ipk = ipk_{P^{\widetilde{\mathsf{gid}}}}$, of P' and $P^{\widetilde{\mathsf{gid}}}$ both honestly arrive at the other.

By **Game** 2, it must further hold that P' and $P^{\widetilde{\mathsf{gid}}}$ agree on each other's binding information $\mathsf{Binding}_{P'}^{\widetilde{\mathsf{gid}}}$ and $\mathsf{Binding}_{P^{\widetilde{\mathsf{gid}}}}^{\widetilde{\mathsf{gid}}}$, which include:

- the group identifier $\widetilde{\mathsf{gid}}$,
- the server-controlled randomness mUUID,
- both parties' identifier $(\mathsf{uid}_{P^{\widetilde{\mathsf{gid}}}}, \mathsf{hid}_{P^{\widetilde{\mathsf{gid}}}})$ and $(\mathsf{uid}_{P'}, \mathsf{hid}_{P'})$,
- both parties' identity keys $ipk_{P^{\widetilde{\mathsf{gid}}}}$ and $ipk_{P'}$, and
- both parties' per-group public key $pk_{P^{\widetilde{\mathsf{gid}}}}^{\widetilde{\mathsf{gid}}}$ and $pk_{P'}^{\widetilde{\mathsf{gid}}}$.

Game $C1.4$**:** This game is identical to **Game** 3 except the following modification:

- The challenger guesses the index of query to the RegisterAuth or RegisterInject oracle, which creates the per-group state $\pi_{P'}^{\widetilde{\mathsf{gid}}}$ in the winning event E_{KAuth}, at the beginning of the experiment and aborts the game if the guess is wrong.

Note that there are at most c_{maxReg} register queries to the group $\widetilde{\mathsf{gid}}$ via the RegisterAuth or RegisterInject oracles in the game. The probability that $\mathcal{C}$ guesses correctly is at least $\frac{1}{c_{\mathsf{maxReg}}}$. Thus, we have that

$$\mathsf{Adv}_3^{C1}(\mathcal{A}) \leq c_{\mathsf{maxReg}}\mathsf{Adv}_4^{C1}(\mathcal{A})$$

Note that the challenger $\mathcal{C}$ will know the identifier of the party P' in the winning event E_{KAuth} at the time of receiving the guessed query to RegisterAuth or RegisterInject oracle. In the following games, we denote the party P' in the winning event E_{KAuth} with $\widetilde{P}$.

Game $C1.5$**:** This game is identical to the **Game** $C1.4$ except the following modifications:

- At the beginning of the experiment, the challenger $\mathcal{C}$ samples a random $\widetilde{K}$ of bit length l_{H_2}.
- When the leader $P^{\widetilde{\mathsf{gid}}}$ of the the group $\widetilde{\mathsf{gid}}$ needs to compute the output K of the Hash function H_2 over a Diffie-Hellman exchange key K', which is computed by the leader's identity private key $\pi_{P^{\widetilde{\mathsf{gid}}}}^{\widetilde{\mathsf{gid}}}.sk$ and the party $\widetilde{P}$'s public key $pk_{\widetilde{P}}^{\widetilde{\mathsf{gid}}}$, and a constant $\mathsf{ctxt}_{\mathsf{H}_2}$ in Line 5 in Figure 6 during the Participant Join phase and the Key Rotation phase of $\widetilde{P}$, $\mathcal{C}$ replaces K with $\widetilde{K}$.
- When $\widetilde{P}$ needs to compute the output K of the Hash function H_2 over a Diffie-Hellman exchange key K', which is computed by the party $\widetilde{P}$'s identity private key $\pi_{\widetilde{P}}^{\widetilde{\mathsf{gid}}}.sk$ and the leader $P^{\widetilde{\mathsf{gid}}}$'s public key $pk_{P^{\widetilde{\mathsf{gid}}}}^{\widetilde{\mathsf{gid}}}$, and a constant $\mathsf{ctxt}_{\mathsf{H}_2}$ in Line 12 in Figure 6 during the Participant Join phase and the Key Rotation phase with the leader of the group $\widetilde{\mathsf{gid}}$, $\mathcal{C}$ replaces K with $\widetilde{K}$.

We analyze the gap between **Game** $C1.4$ and **Game** $C1.5$ by reduction to the hardness of mn-prf-ODH problem over ECDH and H_2. If the attacker $\mathcal{A}$ can distinguish **Game** $C1.4$ and **Game** $C1.5$, then we can construct another attacker $\mathcal{B}_2$ that breaks mn-prf-ODH assumption over ECDH and H_2.

The attacker $\mathcal{B}_2$ receives the ECDH parameters and $U = g^u$ for some unknown u. Next, $\mathcal{B}_2$ immediately issues a challenger query $x^\star = \mathsf{ctxt}_{\mathsf{H}_2}$ to its challenger and receives a tuple $(V = g^v, y^\star)$ for some unknown v. Then $\mathcal{B}_2$ invokes $\mathcal{A}$ and simulates **Game** $C1.4$ honestly, except for answering the queries to the following oracles.

- RegisterAuth$(\mathsf{id}_P, \mathsf{gid}, m)$: If the input party P is the leader of the group $\widetilde{\mathsf{gid}}$, i.e., $P = P^{\widetilde{\mathsf{gid}}}$, and the group identifier $\mathsf{gid} = \widetilde{\mathsf{gid}}$, then $\mathcal{B}_2$ does not sample the ZBox key pair uniformly at random. Instead, $\mathcal{B}_2$ replaces $\pi_{P^{\widetilde{\mathsf{gid}}}}^{\widetilde{\mathsf{gid}}}.pk$ with $U = g^u$ that is given by challenger. The rest of this query is executed honestly.
  If the input party P is the participant that is guessed in **Game** $C1.4$, i.e., $P = \widetilde{P}$, and the group identifier $\mathsf{gid} = \widetilde{\mathsf{gid}}$, then $\mathcal{B}_2$ does not sample the ZBox key pair uniformly at random. Instead, $\mathcal{B}_2$ replaces $\pi_{\widetilde{P}}^{\widetilde{\mathsf{gid}}}.pk$ with $V = g^v$ that is given by challenger. The rest of this query is executed honestly.
  For the other queries to this oracle, $\mathcal{B}_2$ executes them honestly.
- RegisterInject$(\mathsf{id}_P, \mathsf{gid}, gs, m)$: If the input party P is the participant that is guessed in **Game** $C1.4$, i.e., $P = \widetilde{P}$, and the group identifier $\mathsf{gid} = \widetilde{\mathsf{gid}}$, then $\mathcal{B}_2$ does not sample the ZBox key pair uniformly at random. Instead, $\mathcal{B}_2$ replaces $\pi_{\widetilde{P}}^{\widetilde{\mathsf{gid}}}.pk$ with $V = g^v$ that is given by challenger. The rest of this query is executed honestly.
  For the other queries to this oracle, $\mathcal{B}_2$ executes them honestly.
- SendJoinAuth$(\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, m)$: If the input party $P = P^{\widetilde{\mathsf{gid}}}$, the group identifier $\mathsf{gid} = \widetilde{\mathsf{gid}}$, and ZBox public key pk included in the fourth input m equals $V = g^v$ that is given by the challenger, then $\mathcal{B}_2$ does not compute the the Diffie-Hellman exchange in Line 4 and the computation of H_2 in Line 5 in Figure 6. Instead, $\mathcal{B}_2$ replaces the output of H_2 with $y^\star$ that is given by the challenger. The rest of this query is executed honestly.
  If the input party $P = P^{\widetilde{\mathsf{gid}}}$, the group identifier $\mathsf{gid} = \widetilde{\mathsf{gid}}$, and ZBox public key pk included in the fourth input m

does *not* equal to $V = g^v$ that is given by the challenger, then $\mathcal{B}_2$ does not compute the the Diffie-Hellman exchange in Line 4 and the computation of H_2 in Line 5 in Figure 6. Instead, $\mathcal{B}_2$ queries its ODH_u oracle with input $(pk, \mathsf{ctxt}_{\mathsf{H}_2})$ for a reply y, followed by replacing the output of H_2 with the reply y. The rest of this query is executed honestly.
If the input party $P = \widetilde{P}$, the group identifier $\mathsf{gid} = \widetilde{\mathsf{gid}}$, and ZBox public key pk included in the third input m equals $U = g^u$ that is given by the challenger, then $\mathcal{B}_2$ does not compute the the Diffie-Hellman exchange in Line 11 and the computation of H_2 in Line 12 in Figure 6. Instead, $\mathcal{B}_2$ replaces the output of H_2 with $y^\star$ that is given by the challenger. The rest of this query is executed honestly.
For the other queries to this oracle, $\mathcal{B}_2$ executes them honestly.

- SENDJOININJECT$(\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, gs, m)$: If the input party $P = \widetilde{P}$, the group identifier $\mathsf{gid} = \widetilde{\mathsf{gid}}$, and ZBox public key pk included in the fifth input m equals $U = g^u$ that is given by the challenger, then $\mathcal{B}_2$ does not compute the the Diffie-Hellman exchange in Line 11 and the computation of H_2 in Line 12 in Figure 6. Instead, $\mathcal{B}_2$ replaces the output of H_2 with $y^\star$ that is given by the challenger. The rest of this query is executed honestly.
For the other queries to this oracle, $\mathcal{B}_2$ executes them honestly.
- SENDKEYROTAT$(\mathsf{id}_P, \mathsf{gid}, m)$: Note that this oracle requires that the P must have already joined the group gid. In particular, $\mathcal{B}_2$ must have already computed the output of H_2 for every communication between the leader of the group gid and the participants. In this oracle, $\mathcal{B}_2$ does not re-compute the Diffie-Hellman exchange and the computation of H_2. Instead, $\mathcal{B}_2$ simply reuses the corresponding values derived in the SENDJOINAUTH or SENDJOININJECT oracles.
The rest of this oracle is executed honestly.
- COMPROMISE$(\mathsf{id}_P, \mathsf{gid})$: If the first input party $P = P^{\widetilde{\mathsf{gid}}}$ or $P = \widetilde{P}$ and the second input $\mathsf{gid} = \widetilde{\mathsf{gid}}$, then $\mathcal{B}_2$ aborts.
The rest of this oracle is executed honestly.

Note that if the attacker $\mathcal{A}$ can trigger the winning event E_{KAuth} without violating the freshness condition, then neither $\pi^{\widetilde{\mathsf{gid}}}_{P^{\widetilde{\mathsf{gid}}}}$ nor $\pi^{\widetilde{\mathsf{gid}}}_{\widetilde{P}}$ is allowed to be compromised due to the freshness condition $\mathsf{fresh}_{\mathsf{KAuth}}$. This game abortion in the COMPROMISE oracle will not happen.

If the attacker $\mathcal{A}$ is able to distinguish **Game** $C1.4$ and **Game** $C1.5$, then the attacker $\mathcal{B}_2$ returns to 0 to its challenger if the $\mathcal{A}$ thinks this is **Game** $C1.4$ and 1 to its challenger if the $\mathcal{A}$ thinks this is **Game** $C1.5$.

Note that $\mathcal{B}_2$ perfectly simulates **Game** $C1.4$ if $y^\star = \mathsf{H}_2(g^{uv}, \mathsf{ctxt}_{\mathsf{H}_2})$ and **Game** $C1.5$ if $y^\star$ is sampled uniformly at random. $\mathcal{B}_2$ wins whenever $\mathcal{A}$ can distinguish the games. Thus, we have that

$$\mathsf{Adv}^{C1}_4(\mathcal{A}) \leq \mathsf{Adv}^{C1}_5(\mathcal{A}) + \epsilon^{\text{mn-prf-ODH}}_{\mathsf{ECDH},\mathsf{H}_2}$$

Final Analysis for Case 1: Finally, we analyze the advantage that $\mathcal{A}$ can win by triggering E_{KAuth}. This means, there exists any group key index gkid such that

- $gk^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}_{\widetilde{P}} \neq \bot$ and $gk^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}_{P^{\widetilde{\mathsf{gid}}}} \neq gk^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}_{\widetilde{P}}$,
- $\widetilde{P}$ and $P^{\widetilde{\mathsf{gid}}}$ agree on each other's binding information $\mathsf{Binding}^{\widetilde{\mathsf{gid}}}_{\widetilde{P}}$ and $\mathsf{Binding}^{\widetilde{\mathsf{gid}}}_{P^{\widetilde{\mathsf{gid}}}}$ in the Participant Join phase, and
- the per-group states $\pi^{\widetilde{\mathsf{gid}}}_{\widetilde{P}}$ and $\pi^{\widetilde{\mathsf{gid}}}_{P^{\widetilde{\mathsf{gid}}}}$ are not compromised.

From **Game** $C1.5$, we know that $\widetilde{P}$ and $P^{\widetilde{\mathsf{gid}}}$ shares the same random key $\widetilde{K}$ for computing AEAD. From $gk^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}_{\widetilde{P}} \neq \bot$, we know that $\widetilde{P}$ must receives a AEAD nonce and ciphertext tuple (nonce, c), which is decrypted to $(gk^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}_{\widetilde{P}}, \mathsf{gkid}) \neq (gk^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}_{P^{\widetilde{\mathsf{gid}}}}, \mathsf{gkid})$ for some gkid. We consider the following four cases:

- Case 1: $gk^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}_{P^{\widetilde{\mathsf{gid}}}} = \bot$. In this case, the leader $P^{\widetilde{\mathsf{gid}}}$ has not generated the gkid-th group key. This mean, the tuple (nonce, c) must be forged by $\mathcal{A}$. Note that the AEAD associated data D is known by $\mathcal{A}$. If $\mathcal{A}$ can trigger this case, then we can easily construct an attacker $\mathcal{B}_3$ that breaks the CTI-CPA security of the underlying AEAD.
- Case 2: $gk^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}_{P^{\widetilde{\mathsf{gid}}}} \neq \bot$ and c is not produced by $P^{\widetilde{\mathsf{gid}}}$. Similar to the above, if $\mathcal{A}$ can trigger this case, then we can easily construct an attacker $\mathcal{B}_3$ that breaks the CTI-CPA security of the underlying AEAD.
- Case 3: $gk^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}_{P^{\widetilde{\mathsf{gid}}}} \neq \bot$ and c is produced by $P^{\widetilde{\mathsf{gid}}}$ but the nonce nonce is not produced by $P^{\widetilde{\mathsf{gid}}}$ for this $gk^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}_{P^{\widetilde{\mathsf{gid}}}}$. Let $\mathsf{nonce}^{\widetilde{\mathsf{gid}}}$ denote the nonce produced by $P^{\widetilde{\mathsf{gid}}}$ for this $gk^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}_{P^{\widetilde{\mathsf{gid}}}}$. If $\mathcal{A}$ can trigger this case, we can easily construct an attacker $\mathcal{B}_3$ that breaks the customized (n,m)-FROB security of AEAD by outputting $\Big(c, (\widetilde{K}, \mathsf{nonce}, D), (\widetilde{K}, \mathsf{nonce}^{\widetilde{\mathsf{gid}}}, D)\Big)$.
- Case 4: $gk^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}_{P^{\widetilde{\mathsf{gid}}}} \neq \bot$ and (nonce, c) is produced by $P^{\widetilde{\mathsf{gid}}}$ for this $gk^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}_{P^{\widetilde{\mathsf{gid}}}}$. This case is impossible due to the perfect correctness.

Merging the cases analysis above, it holds that

$$\mathsf{Adv}^{C1}_5(\mathcal{A}) \leq \max\left(\epsilon^{\text{cti-cpa}}_{\mathsf{AEAD}}, \epsilon^{(\mathsf{n},\mathsf{m})\text{-frob}}_{\mathsf{AEAD}}\right) \leq \epsilon^{\text{cti-cpa}}_{\mathsf{AEAD}} + \epsilon^{(\mathsf{n},\mathsf{m})\text{-frob}}_{\mathsf{AEAD}}$$

We further have that

$$\mathsf{Adv}^{C1}_3 \leq c_{\mathsf{maxReg}}(\epsilon^{\text{mn-prf-ODH}}_{\mathsf{ECDH},\mathsf{H}_2} + \epsilon^{\text{cti-cpa}}_{\mathsf{AEAD}} + \epsilon^{(\mathsf{n},\mathsf{m})\text{-frob}}_{\mathsf{AEAD}})$$

Case 2: $\mathcal{A}$ wins by triggering event E_{KPriv}.

In this case, due to the winning event E_{KPriv} and the freshness requirement $\mathsf{fresh}^{\text{Sec-mGKD-pki}}_{\mathsf{KPriv}}$, it must hold for the guessed the group identifier $\widetilde{\mathsf{gid}}$ with some leader $P^{\widetilde{\mathsf{gid}}}$ and some group key index gkid that

- $\mathsf{b} = \mathsf{b}'$,
- the group key $gk^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}_{P}$ is not leaked for all P such that $\mathsf{id}_P \in GP^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}$,
- the short-term state $\pi^{\widetilde{\mathsf{gid}}}_{P}$ is not compromised for all P such that $\mathsf{id}_P \in GP^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}$,

- the long-term state $\mathsf{st}_{P^{\widetilde{\mathsf{gid}}}}$ of the leader $P^{\widetilde{\mathsf{gid}}}$ is not corrupted before all other participants P such that $\mathsf{id}_P \in GP^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}$ and $\mathsf{id}_P \neq \mathsf{id}_{P^{\widetilde{\mathsf{gid}}}}$ joined the group $\widetilde{\mathsf{gid}}$,
- the long-term state st_P of all participants P such that $\mathsf{id}_P \in GP^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}$ is not corrupted before P joined the group $\widetilde{\mathsf{gid}}$, and
- the sign-up messages of all parties P such that $\mathsf{id}_P \in GP^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}$ are honestly distributed inside the group $\widetilde{\mathsf{gid}}$.

Game $C2.4$**:** This game is identical the **Game** 3 except for the following modification:

- At the beginning of the experiment, the challenger $\mathcal{C}$ guesses the number n_{party} of parties in the set $GP^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}$, where gkid is the tested group key identifier. The challenger $\mathcal{C}$ aborts if the guess is wrong.

Note that there are at most c_{maxParty} parties in a group simutanously. The probability that $\mathcal{C}$ guesses correctly is bounded by $\frac{1}{c_{\mathsf{maxParty}}}$. Thus, it holds that

$$\mathsf{Adv}_3^{C2}(\mathcal{A}) \leq c_{\mathsf{maxParty}} \mathsf{Adv}_4^{C2}(\mathcal{A})$$

Game $C2.5$**:** This game is identical the **Game** $C2.4$ except for the following modification:

- At the beginning of the experiment, the challenger $\mathcal{C}$ guesses $(n_{\mathsf{party}}-1)$ indices of the queries REGISTERAUTH or REGISTERINJECT that create the per-group states $\pi_{P^i}^{\widetilde{\mathsf{gid}}}$ for some parties P^i, where $1 \leq i \leq (n_{\mathsf{party}}-1)$. The challenger aborts if $\{P^i\}_i$ are not participants in the set $GP^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}$, where gkid is the tested group key identifier.

Note that there are $(n_{\mathsf{party}}-1)$ participants in the set $GP^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}$ and each per-group state can be created in at most c_{maxReg} queries to the REGISTERAUTH or REGISTERINJECT oracles. Thus, the challenger guesses correctly except for the probability $\frac{1}{c_{\mathsf{maxReg}}^{(n_{\mathsf{party}}-1)}}$.

$$\mathsf{Adv}_4^{C2}(\mathcal{A}) \leq c_{\mathsf{maxReg}}^{(n_{\mathsf{party}}-1)} \mathsf{Adv}_5^{C2}(\mathcal{A})$$

Note that the challenger $\mathcal{C}$ will know the identifier of the participants $P \in GP^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}$ in the winning event E_{KPriv} at the time of receiving the $(n_{\mathsf{party}}-1)$ guessed queries to REGISTERAUTH or REGISTERINJECT oracle. In the following games, we denote the n_{party} participants in the winning event E_{KPriv} with P^1, ..., $P^(n_{\mathsf{party}}-1)$. This means, $GP^{(\widetilde{\mathsf{gid}},\mathsf{gkid})} = \{\mathsf{id}_{P^i}\}_i \cup \{\mathsf{id}_{P^{\widetilde{\mathsf{gid}}}}\}$, where gkid is the tested group key identifier.

Game $C2.6$**:** This game is identical the **Game** $C2.5$ except for the following modification:

- At the beginning of the experiment, the challenger $\mathcal{C}$ samples $(n_{\mathsf{party}}-1)$ random string $\widetilde{K}^1$, ..., $\widetilde{K}^{(n_{\mathsf{party}}-1)}$ of bit length l_{H_2}.
- When the leader $P^{\widetilde{\mathsf{gid}}}$ of the the group $\widetilde{\mathsf{gid}}$ needs to compute the output K of the Hash function H_2 over a Diffie-Hellman exchange key K', which is computed by the leader's identity private key $\pi_{P^{\widetilde{\mathsf{gid}}}}^{\widetilde{\mathsf{gid}}}.sk$ and the party P^i's public key $pk_{P^i}^{\widetilde{\mathsf{gid}}}$ for any $1 \leq i \leq (n_{\mathsf{party}}-1)$, and a constant $\mathsf{ctxt}_{\mathsf{H}_2}$ in Line 5 in Figure 6 during the Participant Join phase and the Key Rotation phase of $\widetilde{P}$, $\mathcal{C}$ replaces K with $\widetilde{K}^i$.
- When P^i for any $1 \leq i \leq (n_{\mathsf{party}}-1)$ needs to compute the output K of the Hash function H_2 over a Diffie-Hellman exchange key K', which is computed by the party P^i's identity private key $\pi_{P^i}^{\widetilde{\mathsf{gid}}}.sk$ and the leader $P^{\widetilde{\mathsf{gid}}}$'s public key $pk_{P^{\widetilde{\mathsf{gid}}}}^{\widetilde{\mathsf{gid}}}$, and a constant $\mathsf{ctxt}_{\mathsf{H}_2}$ in Line 12 in Figure 6 during the Participant Join phase and the Key Rotation phase with the leader of the group $\widetilde{\mathsf{gid}}$, $\mathcal{C}$ replaces K with $\widetilde{K}^i$.

The gap between **Game** $C2.5$ and **Game** $C2.6$ can be given by a sequence of hybrid games.

Hybrid Game 0. This game is identical to **Game** $C2.5$. Thus, we have that

$$\mathsf{Adv}_{\mathsf{hy}.0}(\mathcal{A}) = \mathsf{Adv}_5^{C2}(\mathcal{A})$$

Hybrid Game i, where $1 \leq i \leq (n_{\mathsf{party}}-1)$. This game is identical to **Hybrid Game** $(i-1)$ except the following modification

- At the beginning of the experiment, the challenger $\mathcal{C}$ samples a random string $\widetilde{K}^i$ of bit length l_{H_2}.
- When the leader $P^{\widetilde{\mathsf{gid}}}$ of the the group $\widetilde{\mathsf{gid}}$ needs to compute the output K of the Hash function H_2 over a Diffie-Hellman exchange key K', which is computed by the leader's identity private key $\pi_{P^{\widetilde{\mathsf{gid}}}}^{\widetilde{\mathsf{gid}}}.sk$ and the party P^i's public key $pk_{P^i}^{\widetilde{\mathsf{gid}}}$, and a constant $\mathsf{ctxt}_{\mathsf{H}_2}$ in Line 5 in Figure 6 during the Participant Join phase and the Key Rotation phase of $\widetilde{P}$, $\mathcal{C}$ replaces K with $\widetilde{K}^i$.
- When P^i needs to compute the output K of the Hash function H_2 over a Diffie-Hellman exchange key K', which is computed by the party P^i's identity private key $\pi_{P^i}^{\widetilde{\mathsf{gid}}}.sk$ and the leader $P^{\widetilde{\mathsf{gid}}}$'s public key $pk_{P^{\widetilde{\mathsf{gid}}}}^{\widetilde{\mathsf{gid}}}$, and a constant $\mathsf{ctxt}_{\mathsf{H}_2}$ in Line 12 in Figure 6 during the Participant Join phase and the Key Rotation phase with the leader of the group $\widetilde{\mathsf{gid}}$, $\mathcal{C}$ replaces K with $\widetilde{K}^i$.

If the attacker $\mathcal{A}$ can distinguish **Hybrid Game** $(i-1)$ and **Hybrid Game** i, then we can easily construct an attacker $\mathcal{B}_4$ that breaks the mn-prf-ODH security of the underlying ECDH and H_2, similar to the reduction in **Game** $C1.5$. Thus, we can easily have that

$$\mathsf{Adv}_{\mathsf{hy}.(i-1)}(\mathcal{A}) = \mathsf{Adv}_{\mathsf{hy}.i}(\mathcal{A}) + \epsilon_{\mathsf{ECDH},\mathsf{H}_2}^{\mathsf{mn\text{-}prf\text{-}ODH}}$$

Hybrid Game $(n_{\mathsf{party}}-1)$. This game is identical to **Game** $C2.6$. Thus, we have that

$$\mathsf{Adv}_{\mathsf{hy}.(n_{\mathsf{party}}-1)}(\mathcal{A}) = \mathsf{Adv}_6^{C2}(\mathcal{A})$$

To sum up, it holds that

$$\mathsf{Adv}_5^{C2}(\mathcal{A}) \leq \mathsf{Adv}_6^{C2}(\mathcal{A}) + (n_{\mathsf{party}}-1)\epsilon_{\mathsf{ECDH},\mathsf{H}_2}^{\mathsf{mn\text{-}prf\text{-}ODH}}$$

Game $C2.7$**:** This game is identical to **Game** $C2.6$ except the following modification:

- All ciphertexts between the leader $P^{\widetilde{\mathsf{gid}}}$ and P^i, for every $1 \le i \le (n_{\mathsf{party}} - 1)$, are replaced by random strings of the same length.

The gap between **Game** $C2.6$ and **Game** $C2.7$ can be given by $(n_{\mathsf{party}} - 1)$ hybrid games, where the i-th hybrid game replaces all ciphertexts between $P^{\widetilde{\mathsf{gid}}}$ and P^i in the group $\widetilde{\mathsf{gid}}$ encrypted using $\widetilde{K}^i$ for every $1 \le i \le (n_{\mathsf{party}} - 1)$. It is easy to know that the gap between every adjacent hybrid games can be reduced to the IND$-CCA security of the underlying AEAD. Thus, we have that

$$\mathsf{Adv}_6(\mathcal{A}) \le \mathsf{Adv}_7(\mathcal{A}) + (n_{\mathsf{party}} - 1)\epsilon^{\mathsf{ind\$\text{-}cca}}_{\mathsf{AEAD}}$$

Now, the attacker $\mathcal{A}$ obtains no information about the challenge bit b and can only randomly guess. The probability that $\mathcal{A}$ wins is $\frac{1}{2}$, i.e.,

$$\mathsf{Adv}_7(\mathcal{A}) = 0$$

We further have that

$$\mathsf{Adv}^{C2}_3 \le c_{\mathsf{maxParty}} c^{(n_{\mathsf{party}}-1)}_{\mathsf{maxReg}} (n_{\mathsf{party}} - 1)(\epsilon^{\mathsf{mn\text{-}prf\text{-}ODH}}_{\mathsf{ECDH},\mathsf{H}_2} + \epsilon^{\mathsf{ind\$\text{-}cca}}_{\mathsf{AEAD}})$$

Final Analysis. By merging the statements above, the proof is concluded by,

$$\begin{aligned}\mathsf{Adv}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}_{\Pi}(\mathcal{A}) \le\ & \epsilon^{\mathsf{coll\text{-}res}}_{\mathsf{H}_1} + q_{\textsc{NewParty}} \epsilon^{\mathsf{euf\text{-}cma}}_{\mathsf{DS}} \\ & + c_{\mathsf{maxReg}} q_{\textsc{NewGroup}} \Big(\epsilon^{\mathsf{cti\text{-}cpa}}_{\mathsf{AEAD}} + \epsilon^{(\mathsf{n},\mathsf{m})\text{-}\mathsf{frob}}_{\mathsf{AEAD}} \\ & + c^{(n_{\mathsf{party}}-1)}_{\mathsf{maxReg}} (n_{\mathsf{party}} - 1)(\epsilon^{\mathsf{mn\text{-}prf\text{-}ODH}}_{\mathsf{ECDH},\mathsf{H}_2} + \epsilon^{\mathsf{ind\$\text{-}cca}}_{\mathsf{AEAD}}) \Big)\end{aligned}$$

where c_{maxParty} denotes the maximal number of parties per meeting, $l = 192$ denote the length of random nonce in ZBox algorithm, and $q_{\mathcal{O}}$ denote the maximal number of the queries to any oracle $\mathcal{O}$.

Appendix D. Proof of Theorem 2

The proof is given by a sequence of games. Let $\mathsf{Adv}_i(\mathcal{A})$ denote the advantage of an attacker $\mathcal{A}$ in winning **Game** i.

Game 0: This game is identical to the original Sec-mGKD-pw experiment. Thus, we have that

$$\mathsf{Adv}_0(\mathcal{A}) = \mathsf{Adv}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}_{\Pi'}(\mathcal{A})$$

Game 1: This game is identical to the **Game** 0 except the following modification:

- The challenger $\mathcal{C}$ guesses an group identifier $\widetilde{\mathsf{gid}}$ with some leader $P^{\widetilde{\mathsf{gid}}}$ that makes $\mathcal{A}$ win, and aborts the game if the guess is wrong. Namely,
  1) there exists any party P' that is authorized for the group $\widetilde{\mathsf{gid}}$ and any group key index gkid, such that $gk^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}_{P'} \neq \perp$ but $gk^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}_{P^{\widetilde{\mathsf{gid}}}} \neq gk^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}_{P'}$, without violating the freshness condition $\mathsf{fresh}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}_{\mathsf{KAuth}}(\mathsf{id}_{P'}, \widetilde{\mathsf{gid}}, \mathsf{gkid})$.
  2) $\mathsf{b} = \mathsf{b}'$ without violating the freshness condition $\mathsf{fresh}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}_{\mathsf{KPriv}}(\mathsf{id}_{P'}, \widetilde{\mathsf{gid}}, \mathsf{gkid})$, where P', $\widetilde{\mathsf{gid}}$, and gkid are respectively the tested party, group identifier, and group key index.

Note that each group must be created via the NEWGROUP and that the NEWGROUP can be queried at most NEWGROUP times. The challenger guesses correctly with probability at least $\frac{1}{q_{\textsc{NewGroup}}}$. Thus, we have that

$$\mathsf{Adv}_0 \le q_{\textsc{NewGroup}} \mathsf{Adv}_1(\mathcal{A})$$

Note that the freshness conditions $\mathsf{fresh}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}_{\mathsf{KAuth}}(\mathsf{id}_{P'}, \widetilde{\mathsf{gid}}, \mathsf{gkid})$ and $\mathsf{fresh}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}_{\mathsf{KPriv}}(\mathsf{id}_{P'}, \widetilde{\mathsf{gid}}, \mathsf{gkid})$ both require that the group $\widetilde{\mathsf{gid}}$ is not revealed. In particular, this means that if the challenger guesses correctly, then the attacker $\mathcal{A}$ cannot query the REVEAL oracle with input $\widetilde{\mathsf{gid}}$.

Game 2: This game is identical to **Game** 1 except the following modifications:

- Whenever the attacker $\mathcal{A}$ sends queries to the SENDJOINAUTH$(\mathsf{id}_P, \mathsf{id}_{P'}, \widetilde{\mathsf{gid}}, m)$ oracle for some parties P and P' and group $\mathsf{gid} = \widetilde{\mathsf{gid}}$ and the party P is expected to derive a key k_{PP} of the PAKE, the challenger does the following:
  - If there exists no authorized party $P'' \neq P$ in the group $\widetilde{\mathsf{gid}}$ such that P and P'' have the same transcript for the PAKE execution, then the challenger samples the key k_{PP} for the conversation between P and P' uniformly at random instead of computing it from $\mathsf{PAKE}_{\mathsf{PP}}$.
  - If there exists any other authorized party $P'' \neq P$ in the group $\widetilde{\mathsf{gid}}$ such that P and P'' have the same transcript for the PAKE execution, then P'' must have already sampled the key k_{PP} (for a conversation with some party P'''). In this case, the challenger replaces the key k_{PP} of P with the one of P''.

We analyze the gap between **Game** 1 and **Game** 2 by reduction to the w-PAKE security of the PAKE scheme $\mathsf{PAKE}_{\mathsf{PP}}$. Namely, if the attacker $\mathcal{A}$ can distinguish **Game** 1 and **Game** 2, then we can construct an attacker $\mathcal{B}$ that breaks the w-PAKE security of the PAKE scheme $\mathsf{PAKE}_{\mathsf{PP}}$. The attacker $\mathcal{B}_1$ simulates **Game** 1 honestly except for answering the following oracles:

- NEWGROUP$(\mathsf{id}_P, \mathsf{gid})$ if $\mathsf{gid} = \widetilde{\mathsf{gid}}$: $\mathcal{B}_1$ samples gs_{Π} from the distribution $\mathcal{D}_{\Pi}$ and runs the $m^{\widetilde{\mathsf{gid}}}_{\mathsf{GSch}} \xleftarrow{\$} \mathsf{Schedule}(P, \widetilde{\mathsf{gid}}, gs^{\widetilde{\mathsf{gid}}}_{\Pi})$ of Π rather than $\mathsf{Schedule}'$ of Π' for an associated outgoing message $m^{\widetilde{\mathsf{gid}}}_{\mathsf{GSch}}$. Then, $\mathcal{B}_1$ marks the group gid as "created" and "valid" and marks P as the leader of the group gid and "authorized". Finally, $\mathcal{B}_1$ returns $m^{\mathsf{gid}}_{\mathsf{GSch}}$ to $\mathcal{A}$.
- REGISTERAUTH$(\mathsf{id}_P, \mathsf{gid}, m)$ if $\mathsf{gid} = \widetilde{\mathsf{gid}}$ and $\mathsf{id}_P \neq \mathsf{id}_{P^{\widetilde{\mathsf{gid}}}}$: $\mathcal{B}_1$ aborts if this oracle has been queried on the same tuple $(\mathsf{id}_P, \widetilde{\mathsf{gid}})$, or $\widetilde{\mathsf{gid}}$ is not marked as created and valid, or the party P is not authorized for the group $\widetilde{\mathsf{gid}}$. Otherwise, $\mathcal{B}_1$ first simply runs $m' \xleftarrow{\$} \mathsf{Register\text{-}P}(P, \mathsf{gid}, gs^{\widetilde{\mathsf{gid}}}_{\Pi}, m)$, where $gs^{\widetilde{\mathsf{gid}}}_{\Pi}$ is the group secret of protocol Π. Then, $\mathcal{B}_1$ sends queries SENDPAKE$(U^{(\mathsf{id}_P, \widetilde{\mathsf{gid}})}, \epsilon)$ to its challenger

and receives a reply $c_{\mathsf{PP}}^{(P,\widetilde{\mathsf{gid}})}$. Finally, $\mathcal{B}_1$ forwards the tuple $(m', c_{\mathsf{PP}}^{(P,\widetilde{\mathsf{gid}})})$ to $\mathcal{A}$.

- SENDJOINAUTH$(\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, m)$ if $\mathsf{gid} = \widetilde{\mathsf{gid}}$: $\mathcal{B}_1$ first checks
  - whether $\widetilde{\mathsf{gid}}$ is marked as created and valid,
  - whether P is authorized for this group $\widetilde{\mathsf{gid}}$,
  - whether both parties P and P' have been created and registered for this group,
  - whether either P or P' is the leader of the group $\widetilde{\mathsf{gid}}$,
  - whether the leader of the group, either P or P', has joined the group, and that the other party hasn't joined the group yet.

  If any of the check fails, $\mathcal{B}_1$ directly returns $\perp$ to $\mathcal{A}$. Otherwise, $\mathcal{B}_1$ behaviors differently depending on whether the P in the group gid has produced a key k_{PP} of $\mathsf{PAKE}_{\mathsf{PP}}$ during the communication with P'.
  - If P has not produced the $\mathsf{PAKE}_{\mathsf{PP}}$ key in the communication with P' in the group $\widetilde{\mathsf{gid}}$, then $\mathcal{B}_1$ first extracts a valid input message m_1 for running $\mathsf{PAKE}_{\mathsf{PP}}$ from the input message m. Then, $\mathcal{B}_1$ sends queries SENDPAKE$(U^{(\mathsf{id}_P,\widetilde{\mathsf{gid}})}, m_1)$ to its challenger and receives a reply c_{PP}. Finally, $\mathcal{B}_1$ checks whether $\mathsf{PAKE}_{\mathsf{PP}}$ is expected to output a key. If so, $\mathcal{B}_1$ further queries TESTPAKE$(U^{(\mathsf{id}_P,\widetilde{\mathsf{gid}})})$ to its challenger and uses the reply k_{PP} as the output key of $\mathsf{PAKE}_{\mathsf{PP}}$.
  - If the key k_{PP} of $\mathsf{PAKE}_{\mathsf{PP}}$ is produced, then $\mathcal{B}_1$ first extracts an input message m_2 for running Join-L or Join-P algorithm of Π (depending whether P is the leader or a participant of the group $\widetilde{\mathsf{gid}}$) from the input message m. Next, $\mathcal{B}_1$ runs $c \xleftarrow{\$} \mathsf{Join\text{-}L}(P, \mathsf{id}_{P'}, \widetilde{\mathsf{gid}}, gs_{\Pi}^{\widetilde{\mathsf{gid}}}, m_2)$ if P is the leader of the group $\widetilde{\mathsf{gid}}$ or $c \xleftarrow{\$} \mathsf{Join\text{-}P}(P, \mathsf{id}_{P'}, \widetilde{\mathsf{gid}}, gs_{\Pi}^{\widetilde{\mathsf{gid}}}, m_2)$ if P is a participant, where $gs_{\Pi}^{\widetilde{\mathsf{gid}}}$ is the group secret of the protocol Π. Then, $\mathcal{B}_1$ encrypts c using $\mathsf{AEAD}_{\mathsf{PP}}$ under the key k_{PP}, a random nonce $\mathsf{nonce}_{\mathsf{PP}}$, an associated data consisting the sign-up messages of both P and P' for a ciphertext c'.

  Finally, $\mathcal{B}_1$ returns c_{PP} and c' that are computed from above steps.
- REVEAL(gid) if $\mathsf{gid} = \widetilde{\mathsf{gid}}$: $\mathcal{B}_1$ simply aborts the game.

Note that if $\mathcal{A}$ wins, then $\mathcal{A}$ cannot violate the freshness condition and therefore never queries the REVEAL oracle upon $\widetilde{\mathsf{gid}}$. Thus, the game abortion never happens. $\mathcal{B}_1$ perfectly simulates **Game** 1 if the challenge bit of the w-PAKE game is 0 and **Game** 2 if the challenge bit of the w-PAKE game is 1. If $\mathcal{A}$ can distinguish **Game** 1 and **Game** 2, then $\mathcal{B}$ can also distinguish the challenge bit of the w-PAKE security game. Thus, we have that

$$\mathsf{Adv}_1(\mathcal{A}) \leq \mathsf{Adv}_2(\mathcal{A}) + \epsilon^{\mathsf{w\text{-}PAKE}}_{\mathsf{PAKE}_{\mathsf{PP}},\mathcal{D}_{pw}}$$

Game 3: This game is identical to **Game** 2 except the following modifications:

- The challenger $\mathcal{C}$ aborts the game if there exists a participant P' such that
  - P' is authorized for the group $\widetilde{\mathsf{gid}}$ and successfully completed the $\mathsf{PAKE}_{\mathsf{PP}}$ execution when joining the group $\widetilde{\mathsf{gid}}$,
  - P' successfully decrypts an $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext that is not output by $P^{\widetilde{\mathsf{gid}}}$ during the Participant Join phase or Key Rotation phase, and
  - the per-group state of P' and $P^{\widetilde{\mathsf{gid}}}$ in the group $\widetilde{\mathsf{gid}}$ are not compromised.

Recall that the key k_{PP} of the authorized party P' for the group $\widetilde{\mathsf{gid}}$ is sampled random uniformly at random. If P' can successfully decrypt an $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext that is not output by $P^{\widetilde{\mathsf{gid}}}$ during the Participant Join phase or Key Rotation phase, then this means $\mathcal{A}$ can forge an $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext for the key k_{PP} of P' and $P^{\widetilde{\mathsf{gid}}}$ and further breaks the CTI-CPA security of the underlying $\mathsf{AEAD}_{\mathsf{PP}}$ scheme. Thus, we can easily construct another attacker $\mathcal{B}_2$ that breaks the CTI-CPA security of $\mathsf{AEAD}_{\mathsf{PP}}$ by invoking $\mathcal{A}$. Note that there are at most c_{maxReg} participants in the group $\widetilde{\mathsf{gid}}$. The reduction $\mathcal{B}_2$ can simply guesses the index of the register request of P', which is correct with probability at least $\frac{1}{c_{\mathsf{maxReg}}}$, and honestly simulates **Game** 2 to $\mathcal{A}$. Note that $\mathcal{A}$ cannot query COMPROMISE oracle upon $(\mathsf{id}_{P'}, \widetilde{\mathsf{gid}})$ or $(\mathsf{id}_{P^{\widetilde{\mathsf{gid}}}}, \widetilde{\mathsf{gid}})$. $\mathcal{B}_2$ can perfectly simulates **Game** 2 to $\mathcal{A}$ and win whenever $\mathcal{A}$ can make the forgery. Thus, it holds that

$$\mathsf{Adv}_2(\mathcal{A}) \leq \mathsf{Adv}_3(\mathcal{A}) + c_{\mathsf{maxReg}}\epsilon^{\mathsf{cti\text{-}cpa}}_{\mathsf{AEAD}_{\mathsf{PP}}}$$

Game 4: This game is identical to **Game** 3 except the following modifications:

- The challenger $\mathcal{C}$ aborts the game if there exists a participant P' such that:
  - P' is authorized for the group $\widetilde{\mathsf{gid}}$ and successfully completed the Participant Join phase in the group $\widetilde{\mathsf{gid}}$,
  - P' and $P^{\widetilde{\mathsf{gid}}}$ have disagreement on sign-up messages of P' and $P^{\widetilde{\mathsf{gid}}}$, and
  - the per-group state of P' and $P^{\widetilde{\mathsf{gid}}}$ in the group $\widetilde{\mathsf{gid}}$ are not compromised.

Recall in **Game** 3 that we ensure that the authorized participant P' must agree on all received $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext with the leader $P^{\widetilde{\mathsf{gid}}}$, in particular, during the Participan Join phase in the group $\widetilde{\mathsf{gid}}$. If P' and $P^{\widetilde{\mathsf{gid}}}$ have disagreement on sign-up messages of P' and $P^{\widetilde{\mathsf{gid}}}$, which are the associated data for encrypting and decrypting the $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext, then we can easily construct an $\mathcal{B}_2$ that breaks the d-frob security of the underlying $\mathsf{AEAD}_{\mathsf{PP}}$ scheme. Thus, we can easily have that

$$\mathsf{Adv}_3(\mathcal{A}) \leq \mathsf{Adv}_4(\mathcal{A}) + \epsilon^{\mathsf{d\text{-}frob}}_{\mathsf{AEAD}_{\mathsf{PP}}}$$

In particular, this game ensures that for all participant P' that is authorized for the group $\widetilde{\mathsf{gid}}$ and successfully completed the Participant Join phase in the group $\widetilde{\mathsf{gid}}$, if the per-group state of P' and $P^{\widetilde{\mathsf{gid}}}$ in the group $\widetilde{\mathsf{gid}}$ are not

compromised, then the participant P' and the leader $P^{\widetilde{\mathsf{gid}}}$ must agree on the each other's sign-up messages. In other words, the sign-up messages $m^{P^{\widetilde{\mathsf{gid}}}}_{\mathsf{SignUp}}$ and $m^{P'}_{\mathsf{SignUp}}$ of $P^{\widetilde{\mathsf{gid}}}$ and P' must be honestly delivered to the other.

Below, we analyze the advantage that $\mathcal{A}$ wins in **Game** 4 by case distinction, i.e., whether $\mathcal{A}$ wins by triggering E_{KAuth} in Case 1, the advantage of which is denoted by Adv^{C1}_4, or by triggering E_{KPriv} in Case 2, the advantage of which is denoted by Adv^{C2}_4. Thus, we have that

$$\mathsf{Adv}_4(\mathcal{A}) := \max\left(\mathsf{Adv}^{C1}_4(\mathcal{A}), \mathsf{Adv}^{C2}_4(\mathcal{A})\right)$$

Case 1: $\mathcal{A}$ wins by triggering event E_{KAuth}.

Final Analysis of Case 1. In this case, $\mathcal{A}$ wins by triggering event E_{KAuth}. This means,there exists any party P' that is authorized for the group $\widetilde{\mathsf{gid}}$ and any group key index gkid, such that $gk^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}_{P'} \neq \perp$ but $gk^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}_{P^{\widetilde{\mathsf{gid}}}} \neq gk^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}_{P'}$, without violating the freshness condition $\mathsf{fresh}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}_{\mathsf{KAuth}}(\mathsf{id}_{P'}, \widetilde{\mathsf{gid}}, \mathsf{gkid})$.

Recall that $\mathsf{fresh}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}_{\mathsf{KAuth}}(\mathsf{id}_{P'}, \widetilde{\mathsf{gid}}, \mathsf{gkid})$ holds if and only if

1) the per-group states $\pi^{\widetilde{\mathsf{gid}}}_{P^{\widetilde{\mathsf{gid}}}}$ and $\pi^{\widetilde{\mathsf{gid}}}_{P'}$ are not compromised,
2) the long-term states $\mathsf{st}_{P^{\widetilde{\mathsf{gid}}}}$ and $\mathsf{st}_{P'}$ are not corrupted before $P^{\widetilde{\mathsf{gid}}}$ and P' joined the group $\widetilde{\mathsf{gid}}$, and
3) the group $\widetilde{\mathsf{gid}}$ is not revealed.

Recall also that the freshness condition $\mathsf{fresh}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}_{\mathsf{KAuth}}(\mathsf{id}_{P'}, \widetilde{\mathsf{gid}}, \mathsf{gkid})$ holds if and only if

1) the per-group states $\pi^{\widetilde{\mathsf{gid}}}_{P^{\widetilde{\mathsf{gid}}}}$ and $\pi^{\widetilde{\mathsf{gid}}}_{P'}$ are not compromised,
2) the long-term states $\mathsf{st}_{P^{\widetilde{\mathsf{gid}}}}$ and $\mathsf{st}_{P'}$ are not corrupted before $P^{\widetilde{\mathsf{gid}}}$ and P' joined the group $\widetilde{\mathsf{gid}}$, and
3) the sign-up messages $m^{P^{\widetilde{\mathsf{gid}}}}_{\mathsf{SignUp}}$ and $m^{P'}_{\mathsf{SignUp}}$ of $P^{\widetilde{\mathsf{gid}}}$ and P' are honestly delivered to the other.

In **Game** 4, we ensure that the the sign-up messages $m^{P^{\widetilde{\mathsf{gid}}}}_{\mathsf{SignUp}}$ and $m^{P'}_{\mathsf{SignUp}}$ of $P^{\widetilde{\mathsf{gid}}}$ and P' are honestly delivered to the other if

1) the party P' is authorized for the group $\widetilde{\mathsf{gid}}$ completed the Participant Join phase in the group $\widetilde{\mathsf{gid}}$,
2) the per-group states $\pi^{\widetilde{\mathsf{gid}}}_{P^{\widetilde{\mathsf{gid}}}}$ and $\pi^{\widetilde{\mathsf{gid}}}_{P'}$ are not compromised,

Thus, if the freshness condition $\mathsf{fresh}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}_{\mathsf{KAuth}}(\mathsf{id}_{P'}, \widetilde{\mathsf{gid}}, \mathsf{gkid})$ holds, then the freshness condition $\mathsf{fresh}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}_{\mathsf{KAuth}}(\mathsf{id}_{P'}, \widetilde{\mathsf{gid}}, \mathsf{gkid})$ must also hold.

Below, we prove that if $\mathcal{A}$ can win via the event E_{KAuth} against Π', then we can construct another attacker $\mathcal{B}_3$ that breaks the Sec-mGKD-pki security of Π via the event E_{KAuth} by invoking $\mathcal{A}$. $\mathcal{B}_3$ answers the queries from $\mathcal{A}$ as follows:

- NEWPARTY(id_P): $\mathcal{B}_3$ simply forwards this query to its challenger and the reply to $\mathcal{A}$.
- NEWGROUP($\mathsf{id}_P, \mathsf{gid}$): $\mathcal{B}_3$ simply forwards this query to its challenger and the reply to $\mathcal{A}$. Moreover, $\mathcal{B}_3$ samples a random pw^{gid} from the distribution $\mathcal{D}_{pw}$ for the group gid.
- AUTHORIZE($\mathsf{gid}, \mathsf{id}_P$): $\mathcal{B}_3$ simply forwards this query to its challenger.
- REGISTERAUTH($\mathsf{id}_P, \mathsf{gid}, m$): $\mathcal{B}_3$ simply forwards this query to its challenger for a reply c. If P is not the leader of the group gid, then $\mathcal{B}_3$ additionally runs the first pass of $\mathsf{PAKE}_{\mathsf{PP}}$ upon pw^{gid} and returns c together with the outgoing message of $\mathsf{PAKE}_{\mathsf{PP}}$ to $\mathcal{A}$.
- REGISTERINJECT($\mathsf{id}_P, \mathsf{gid}, gs, m$): $\mathcal{B}_3$ parses gs into two portions (gs_Π, pw). Next, $\mathcal{B}_3$ simply forwards the query REGISTERINJECT($\mathsf{id}_P, \mathsf{gid}, gs_\Pi, m$) to its challenger for a reply c. If P is not the leader of the group gid, then $\mathcal{B}_3$ additionally runs the first pass of $\mathsf{PAKE}_{\mathsf{PP}}$ upon pw and returns c together with the outgoing message of $\mathsf{PAKE}_{\mathsf{PP}}$ to $\mathcal{A}$.
- SENDJOINAUTH($\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, m$): We consider two cases: For the first case that $\mathsf{gid} = \widetilde{\mathsf{gid}}$, we consider the following two steps:
  - If the party P of the group gid has not derived the key k_{PP}, then $\mathcal{B}_3$ runs the next pass of $\mathsf{PAKE}_{\mathsf{PP}}$ upon necessary information from the input message m and the password pw^{gid}. If the party P of the group gid now is expected to derive the key k_{PP} of $\mathsf{PAKE}_{\mathsf{PP}}$ and there is no other party P'' in the group $\widetilde{\mathsf{gid}}$ that has the same transcript of $\mathsf{PAKE}_{\mathsf{PP}}$ as P, then $\mathcal{B}_3$ replaces it by a random key of the same length.
    If the party P of the group gid now is expected to derive the key k_{PP} of $\mathsf{PAKE}_{\mathsf{PP}}$ and there is a party P'' in the group $\widetilde{\mathsf{gid}}$ that has the same transcript of $\mathsf{PAKE}_{\mathsf{PP}}$ as P, then $\mathcal{B}_3$ replaces the key of P with the one of P''.
  - If the party P of the group gid now has derived the key k_{PP}, then $\mathcal{B}_3$ first checks P agrees on the party P''s sign-up message $m^{P'}_{\mathsf{SignUp}}$. If not, then $\mathcal{B}_3$ simply aborts. Otherwise, $\mathcal{B}_3$ checks whether m should include an $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext. If so, $\mathcal{B}_3$ decrypts the $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext using the random key k_{PP} and other necessary information from the input m for a message m_1. Then, $\mathcal{B}_3$ sends the query SENDJOINAUTH($\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, m_1$) to its challenger for a reply m_2. After that, if m_2 is not an empty string, then $\mathcal{B}_3$ encrypts m_2 using $\mathsf{AEAD}_{\mathsf{PP}}$ upon the random key k_{PP}, a random nonce, an associated data consisting the sign-up messages of P and P'.

  Finally, $\mathcal{B}_3$ forwards all outgoing messages in this algorithm to $\mathcal{A}$.
  For the second case that $\mathsf{gid} \neq \widetilde{\mathsf{gid}}$, we consider the following two steps:
  - If the party P of the group gid has not derived the key k_{PP}, then $\mathcal{B}_3$ runs the next pass of $\mathsf{PAKE}_{\mathsf{PP}}$ upon necessary information from the input message m and the password pw^{gid}.
  - If the party P of the group gid now has derived the key k_{PP}, then $\mathcal{B}_3$ checks whether m should include an $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext. If so, $\mathcal{B}_3$ decrypts the $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext using the key k_{PP} and other necessary information from the input m for a message m_1. Then, $\mathcal{B}_3$ sends the query SENDJOINAUTH($\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, m_1$) to its challenger for a reply m_2. After that, if m_2 is not an empty string, then $\mathcal{B}_3$ encrypts m_2 using $\mathsf{AEAD}_{\mathsf{PP}}$ upon the key k_{PP}, a random nonce, an associated data consisting the sign-up messages of P and P'.

Finally, $\mathcal{B}_3$ forwards all outgoing messages in this algorithm to $\mathcal{A}$.

- SENDJOININJECT$(\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, gs, m)$: $\mathcal{B}_3$ parses gs into two portion gs_Π and pw. We consider the following two steps:
  - If the party P of the group gid has not derived the key k_{PP}, then $\mathcal{B}_3$ runs the next pass of $\mathsf{PAKE}_{\mathsf{PP}}$ upon necessary information from the input message m and the password pw.
  - If the party P of the group gid now has derived the key k_{PP}, then $\mathcal{B}_3$ checks whether m should include an $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext. If so, $\mathcal{B}_3$ decrypts the $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext using the key k_{PP} and other necessary information from the input m for a message m_1. Then, $\mathcal{B}_3$ sends the query SENDJOININJECT$(\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, gs_\Pi, m_1)$ to its challenger for a reply m_2. After that, if m_2 is not an empty string, then $\mathcal{B}_3$ encrypts m_2 using $\mathsf{AEAD}_{\mathsf{PP}}$ upon the key k_{PP}, a random nonce, an associated data consisting the sign-up messages included in m.

  Finally, $\mathcal{B}_3$ forwards all outgoing messages in this algorithm to $\mathcal{A}$.
- SENDLEAVE$(\mathsf{id}_P, \mathsf{gid}, \mathsf{id}_{P'})$: $\mathcal{B}_3$ simply forwards this query to its challenger. If a per-group state π needs to be erased, then $\mathcal{B}_3$ also erases the corresponding $\mathsf{PAKE}_{\mathsf{PP}}$ secrets of π.
- ENDGROUP(gid): $\mathcal{B}_3$ simply forwards this query to its challenger. If the leader's per-group state π needs to be erased, then $\mathcal{B}_3$ also erases the corresponding $\mathsf{PAKE}_{\mathsf{PP}}$ secrets of π.
- SENDKEYROTAT$(\mathsf{id}_P, \mathsf{gid}, m)$: We consider two cases:
  - If P is the leader in the group gid, then $\mathcal{B}_3$ first queries SENDKEYROTAT$(\mathsf{id}_P, \mathsf{gid}, m)$ to its challenger for a reply c. Then, $\mathcal{B}_3$ extracts the portion $c_{\overline{P}}$ in c that is specific to every participant $\overline{P}$ in the group gid, followed by encrypting it using the corresponding $\mathsf{AEAD}_{\mathsf{PP}}$ key k_{PP}, a random nonce, and an associated data that is same as the one in the corresponding Participant Join phase. Finally, $\mathcal{B}_3$ outputs the $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext and nonce for every participant $\overline{P}$ in the group gid. If any error occurs during the above execution, then $\mathcal{B}_3$ aborts and undoes the above executions.
  - If P is a participant in the group gid, then $\mathcal{B}_3$ first extracts an $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext and an $\mathsf{AEAD}_{\mathsf{PP}}$ nonce from the input m. Next, $\mathcal{B}_3$ recovers a message m_1 from the $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext using the corresponding $\mathsf{AEAD}_{\mathsf{PP}}$ key k_{PP}, the $\mathsf{AEAD}_{\mathsf{PP}}$ nonce, and an associated data that is same as the one in the corresponding Participant Join phase. Then, $\mathcal{B}_3$ extracts other necessary information m_2 from the input m for querying SENDKEYROTAT$(\mathsf{id}_P, \mathsf{gid}, m_1 \parallel m_2)$ returns the reply m_{KRot} to $\mathcal{A}$. If any error occurs during the above execution, then $\mathcal{B}_3$ aborts and undoes the above executions.
- CORRUPT(id_P): $\mathcal{B}_3$ simply forwards this query to its challenger and the reply to $\mathcal{A}$.
- COMPROMISE$(\mathsf{id}_P, \mathsf{gid})$: $\mathcal{B}_3$ forwards this query to its challenger. If the reply is not $\perp$, then $\mathcal{B}_3$ forwards the reply together with the key k_{PP} of P in the group gid to $\mathcal{A}$. Otherwise, $\mathcal{B}_3$ simply returns $\perp$ to $\mathcal{A}$.
- LEAK$(\mathsf{id}_P, \mathsf{gid}, \mathsf{gkid})$: $\mathcal{B}_3$ simply forwards this query to its challenger and the reply to $\mathcal{A}$.
- REVEAL(gid): $\mathcal{B}_3$ forwards this query to its challenger. Then, $\mathcal{B}_3$ forwards the reply together with the password pw^{gid} to $\mathcal{A}$.
- TEST$(\mathsf{id}_P, \mathsf{gid}, \mathsf{gkid})$: $\mathcal{B}_3$ simply forwards this query to its challenger and the reply to $\mathcal{A}$.

Note that $\mathcal{B}_3$ perfectly simulates **Game** 4 to $\mathcal{A}$. If $\mathcal{A}$ can trigger the event E_{KAuth} against Π', then $\mathcal{B}_3$ can also trigger the event E_{KAuth} against Π. Recall that if the freshness condition $\mathsf{fresh}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}_{\mathsf{KAuth}}(\mathsf{id}_{P'}, \widetilde{\mathsf{gid}}, \mathsf{gkid})$ holds, then the freshness condition $\mathsf{fresh}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}_{\mathsf{KAuth}}(\mathsf{id}_{P'}, \widetilde{\mathsf{gid}}, \mathsf{gkid})$ must also hold. Thus, if $\mathcal{A}$ can win the game against Π' by triggering the event E_{KAuth}, then $\mathcal{B}_3$ can also win the game against Π by triggering E_{KAuth}. We have that

$$\mathsf{Adv}^{C1}_4(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}_{\Pi}(\mathcal{B}_3)$$

Case 2: $\mathcal{A}$ wins by triggering event E_{KPriv}.

Game $C2.5$**:** This game is identical to **Game** 4 except the following modifications:

- Whenever the challenger needs to let the leader $P^{\widetilde{\mathsf{gid}}}$ compute a $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext for a participant P', where $\mathsf{id}_{P'} \in \pi^{\widetilde{\mathsf{gid}}}_{P^{\widetilde{\mathsf{gid}}}}.GP$ but P' is unauthorized for the group $\widetilde{\mathsf{gid}}$, the challenger replaces this $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext by a random ciphertext of the same length.

Recall in **Game** 2 that we have the key k_{PP} computed by all authorized parties P in the group $\widetilde{\mathsf{gid}}$ be uniformly at random. This in particular means that all keys k_{PP} generated by the leader $P^{\widetilde{\mathsf{gid}}}$ for all participants P', which are unauthorized for the group $\widetilde{\mathsf{gid}}$, are random. Note also that the leader must be in the party set $\mathsf{id}_{P^{\widetilde{\mathsf{gid}}}} \in GP^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}$ for all gkid unless the group ends. Thus for all party P' and group key index gkid, the short-term state $\pi^{\widetilde{\mathsf{gid}}}_{P^{\widetilde{\mathsf{gid}}}}$ must not be compromised, if $\mathsf{fresh}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}_{\mathsf{KPriv}}(\mathsf{id}_{P'}, \widetilde{\mathsf{gid}}, \mathsf{gkid})$ holds. Then, we analyze the gap between **Game** 4 and **Game** $C2.5$ by n hybrid games, where n denotes the number of register requests sent by unauthorized party. Obviously, it holds that $n \leq c_{\mathsf{maxReg}}$.

Hybrid Game 0. This game is identical to **Game** 4. Thus, we have that

$$\mathsf{Adv}_{\mathsf{hy}.0}(\mathcal{A}) = \mathsf{Adv}^{C2}_4(\mathcal{A})$$

Hybrid Game i, where $1 \leq i \leq n$. This game is identical to **Hybrid Game** $(i-1)$ except the following modifications:

- Whenever the challenger needs to sample a random key k_{PP} in the query SENDJOINAUTH$(\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, m)$, where $P = P^{\widetilde{\mathsf{gid}}}$, P' is the unauthorized party that sends the i-th register request, and $\mathsf{gid} = \widetilde{\mathsf{gid}}$, the challenger do not sample this key but mark this key as k^i_{PP}.
- Whenever the challenger needs to compute an $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext that is encrypted upon the key k^i_{PP}, the challenger first checks whether a ciphertext has been produced upon

the same input. If such ciphertext exists, then the challenger simply reuses this ciphertext. If not, then the challenger samples a ciphertext of the same length uniformly at random.

If the attacker $\mathcal{A}$ can distinguish **Hybrid Game** $(i-1)$ and **Hybrid Game** i, then we can easily construct an attacker $\mathcal{B}_4$ that breaks the IND$-CCA security of the underlying $\mathsf{AEAD}_{\mathsf{PP}}$. Thus, we can easily have that

$$\mathsf{Adv}_{\mathsf{hy}.(i-1)}(\mathcal{A}) = \mathsf{Adv}_{\mathsf{hy}.i}(\mathcal{A}) + \epsilon^{\mathsf{ind\$\text{-}cca}}_{\mathsf{AEAD}_{\mathsf{PP}}}$$

Hybrid Game n. This game is identical to **Game** $C2.5$. Thus, we have that

$$\mathsf{Adv}_{\mathsf{hy}.n}(\mathcal{A}) = \mathsf{Adv}^{C2}_{5}(\mathcal{A})$$

To sum up, it holds that

$$\begin{aligned}\mathsf{Adv}^{C2}_{4}(\mathcal{A}) &\leq \mathsf{Adv}^{C2}_{5}(\mathcal{A}) + n\epsilon^{\mathsf{ind\$\text{-}cca}}_{\mathsf{AEAD}_{\mathsf{PP}}} \\ &\leq \mathsf{Adv}^{C2}_{5}(\mathcal{A}) + c_{\mathsf{maxReg}}\epsilon^{\mathsf{ind\$\text{-}cca}}_{\mathsf{AEAD}_{\mathsf{PP}}}\end{aligned}$$

Final Analysis Of Case 2. Now, we prove that $\mathcal{A}$ cannot win by triggering the events E_{KPriv} by reduction. If the attacker $\mathcal{A}$ can win against Π' by triggering E_{KPriv}, the we can construct another attacker $\mathcal{B}_5$ that breaks the Sec-mGKD-pki security of Π by triggering the event E_{KPriv}. The attacker $\mathcal{B}_5$ honestly simulates **Game** $C2.5$ to $\mathcal{A}$ except for the following modifications:

- NEWPARTY(id_P): $\mathcal{B}_5$ simply forwards this query to its challenger and the reply to $\mathcal{A}$.
- NEWGROUP$(\mathsf{id}_P, \mathsf{gid})$: $\mathcal{B}_5$ simply forwards this query to its challenger and the reply to $\mathcal{A}$. Moreover, $\mathcal{B}_5$ samples a random pw^{gid} from the distribution $\mathcal{D}_{pw}$ for the group gid.
- AUTHORIZE$(\mathsf{gid}, \mathsf{id}_P)$: $\mathcal{B}_5$ simply forwards this query to its challenger.
- REGISTERAUTH$(\mathsf{id}_P, \mathsf{gid}, m)$: $\mathcal{B}_5$ simply forwards this query to its challenger for a reply c. If P is not the leader of the group gid, then $\mathcal{B}_5$ additionally runs the first pass of $\mathsf{PAKE}_{\mathsf{PP}}$ upon pw^{gid} and returns c together with the outgoing message of $\mathsf{PAKE}_{\mathsf{PP}}$ to $\mathcal{A}$.
- REGISTERINJECT$(\mathsf{id}_P, \mathsf{gid}, gs, m)$: We consider two case:
  - If $\mathsf{gid} = \widetilde{\mathsf{gid}}$, then $\mathcal{B}_5$ first sends a query CORRUPT(id_P) to its challenger. Afterwards, $\mathcal{B}_5$ honestly runs Register-P$'(P, \mathsf{gid}, gs, m)$ by himself.
  - If $\mathsf{gid} \neq \widetilde{\mathsf{gid}}$, then $\mathcal{B}_5$ parses gs into two portions (gs_Π, pw). Next, $\mathcal{B}_5$ simply forwards the query REGISTERINJECT$(\mathsf{id}_P, \mathsf{gid}, gs_\Pi, m)$ to its challenger for a reply c. Then $\mathcal{B}_5$ runs the first pass of $\mathsf{PAKE}_{\mathsf{PP}}$ upon pw and returns c together with the outgoing message of $\mathsf{PAKE}_{\mathsf{PP}}$ to $\mathcal{A}$.
- SENDJOINAUTH$(\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, m)$: We consider two cases: For the first case that $\mathsf{gid} = \widetilde{\mathsf{gid}}$, we consider the following two steps:
  - If the party P of the group gid has not derived the key k_{PP}, then $\mathcal{B}_5$ runs the next pass of $\mathsf{PAKE}_{\mathsf{PP}}$ upon necessary information from the input message m and the password pw^{gid}. If the party P of the group gid now is expected to derive the key k_{PP} of $\mathsf{PAKE}_{\mathsf{PP}}$ and there is no other party P'' in the group $\widetilde{\mathsf{gid}}$ that has the same transcript of $\mathsf{PAKE}_{\mathsf{PP}}$ as P, then $\mathcal{B}_5$ replaces it by a random key of the same length.
    If the party P of the group gid now is expected to derive the key k_{PP} of $\mathsf{PAKE}_{\mathsf{PP}}$ and there is a party P'' in the group $\widetilde{\mathsf{gid}}$ that has the same transcript of $\mathsf{PAKE}_{\mathsf{PP}}$ as P, then $\mathcal{B}_5$ replaces the key of P with the one of P''.
  - If the party P of the group gid now has derived the key k_{PP}, then $\mathcal{B}_5$ first checks whether P' is authorized for the group gid or not. If P' is authorized for the group gid, then $\mathcal{B}_5$ further checks whether P agrees on the party P''s sign-up message $m^{P'}_{\mathsf{SignUp}}$. If not, then $\mathcal{B}_5$ simply aborts. Otherwise, $\mathcal{B}_5$ checks whether m should include an $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext. If so, this $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext must be encrypted by the party P' from some message m_1. Then, $\mathcal{B}_5$ simply sends the query SENDJOINAUTH$(\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, m_1)$ to its challenger for a reply m_2. After that, if m_2 is not an empty string, then $\mathcal{B}_5$ encrypts m_2 using $\mathsf{AEAD}_{\mathsf{PP}}$ upon the random key k_{PP}, a random nonce, an associated data consisting the sign-up messages of P and P'.
    If P' is unauthorized for the group gid and this invocation needs to outputs some $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext, $\mathcal{B}_5$ simply samples the $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext randomly. If $\mathsf{id}_{P'}$ is expected to be added in to the set $\pi^{\widetilde{\mathsf{gid}}}_{P^{\widetilde{\mathsf{gid}}}}.GP$, $\mathcal{B}_5$ does not add $\mathsf{id}_{P'}$ into this set. Instead, $\mathcal{B}_5$ marks it as "fake".

  Finally, $\mathcal{B}_5$ forwards all outgoing messages in this algorithm to $\mathcal{A}$.

  For the second case that $\mathsf{gid} \neq \widetilde{\mathsf{gid}}$, we consider the following two steps:
  - If the party P of the group gid has not derived the key k_{PP}, then $\mathcal{B}_5$ runs the next pass of $\mathsf{PAKE}_{\mathsf{PP}}$ upon necessary information from the input message m and the password pw^{gid}.
  - If the party P of the group gid now has derived the key k_{PP}, then $\mathcal{B}_5$ checks whether m should include an $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext. If so, $\mathcal{B}_5$ decrypts the $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext using the key k_{PP} and other necessary information from the input m for a message m_1. Then, $\mathcal{B}_5$ sends the query SENDJOINAUTH$(\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, m_1)$ to its challenger for a reply m_2. After that, if m_2 is not an empty string, then $\mathcal{B}_5$ encrypts m_2 using $\mathsf{AEAD}_{\mathsf{PP}}$ upon the key k_{PP}, a random nonce, an associated data consisting the sign-up messages of P and P'.

  Finally, $\mathcal{B}_5$ forwards all outgoing messages in this algorithm to $\mathcal{A}$.
- SENDJOININJECT$(\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, gs, m)$: We consider two cases. For the first case that $\mathsf{gid} = \widetilde{\mathsf{gid}}$, $\mathcal{B}_5$ simply computes Join-P$(P, \mathsf{id}_{P'}, \mathsf{gid}, gs, m)$ by himself.

  For the second case that $\mathsf{gid} \neq \widetilde{\mathsf{gid}}$, $\mathcal{B}_5$ parses gs into two portion gs_Π and pw. Then, we consider the following two steps:
  - If the party P of the group gid has not derived the key k_{PP}, then $\mathcal{B}_5$ runs the next pass of $\mathsf{PAKE}_{\mathsf{PP}}$ upon necessary information from the input message m and the password pw.

– If the party P of the group gid now has derived the key k_{PP}, then $\mathcal{B}_5$ checks whether m should include an $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext. If so, $\mathcal{B}_5$ decrypts the $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext using the key k_{PP} and other necessary information from the input m for a message m_1. Then, $\mathcal{B}_5$ sends the query $\textsc{SendJoinInject}(\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, gs_\Pi, m_1)$ to its challenger for a reply m_2. After that, if m_2 is not an empty string, then $\mathcal{B}_5$ encrypts m_2 using $\mathsf{AEAD}_{\mathsf{PP}}$ upon the key k_{PP}, a random nonce, an associated data consisting the sign-up messages included in m.

Finally, $\mathcal{B}_5$ forwards all outgoing messages in this algorithm to $\mathcal{A}$.

- $\textsc{SendLeave}(\mathsf{id}_P, \mathsf{gid}, \mathsf{id}_{P'})$: We consider two cases: For the first case that $\mathsf{gid} = \widetilde{\mathsf{gid}}$, we further consider the following three sub-cases:
  – If $\mathsf{id}_P = \mathsf{id}_{P^{\widetilde{\mathsf{gid}}}}$ and P' is unauthorized for the group $\widetilde{\mathsf{gid}}$ and marked as "fake", then $\mathcal{B}$ removes the mark "fake" on P'.
  – If P is marked as "fake", then $\mathcal{B}_5$ executes $\mathsf{Leave\text{-}P}(P, \mathsf{gid}, \mathsf{id}_{P'})$ by himself.
  – For all other queries, $\mathcal{B}_5$ forwards them to its challenger and the reply to $\mathcal{A}$.

  For the second case that $\mathsf{gid} \neq \widetilde{\mathsf{gid}}$, $\mathcal{B}_5$ simply forwards this query to its challenger. If a per-group state π needs to be erased, then $\mathcal{B}_5$ also erases the corresponding $\mathsf{PAKE}_{\mathsf{PP}}$ secrets of π.
- $\textsc{EndGroup}(\mathsf{gid})$: We consider the following two case: For the first case that $\mathsf{gid} = \widetilde{\mathsf{gid}}$, if there exists no party P' that is still marked as "fake", then $\mathcal{B}_5$ forwards this query to its challenger. Otherwise, $\mathcal{B}_5$ aborts.

  For the second case that $\mathsf{gid} \neq \widetilde{\mathsf{gid}}$, $\mathcal{B}_5$ simply forwards this query to its challenger. If the leader's per-group state π needs to be erased, then $\mathcal{B}_5$ also erases the corresponding $\mathsf{PAKE}_{\mathsf{PP}}$ secrets of π.
- $\textsc{SendKeyRotat}(\mathsf{id}_P, \mathsf{gid}, m)$ We consider two cases: For the first case that $\mathsf{gid} = \widetilde{\mathsf{gid}}$, we further consider following three cases:
  – If P is the leader in the group gid, then $\mathcal{B}_5$ first queries $\textsc{SendKeyRotat}(\mathsf{id}_P, \mathsf{gid}, m)$ to its challenger for a reply c. Then, $\mathcal{B}_5$ extracts the portion $c_{\overline{P}}$ in c that is specific to every participant $\overline{P}$ in the group gid, followed by encrypting it using the corresponding $\mathsf{AEAD}_{\mathsf{PP}}$ key k_{PP}, a random nonce, and an associated data that is same as the one in the corresponding Participant Join phase. Moreover, for every party P' that is marked as "fake", $\mathcal{B}_5$ also samples a random $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext and a random nonce. Finally, $\mathcal{B}_5$ outputs all above $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertexts and nonces. If any error occurs during the above execution, then $\mathcal{B}_5$ aborts and undoes the above executions.
  – If P is an authorized participant in the group gid, then $\mathcal{B}_5$ first extracts an $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext and an $\mathsf{AEAD}_{\mathsf{PP}}$ nonce from the input m. If this $\mathsf{AEAD}_{\mathsf{PP}}$ is not produced by the leader for some message m_1, then $\mathcal{B}_5$ aborts. Otherwise, $\mathcal{B}_5$ extracts other necessary information m_2 from the input m for querying $\textsc{SendKeyRotat}(\mathsf{id}_P, \mathsf{gid}, m_1 \parallel m_2)$ returns the reply m_{KRot} to $\mathcal{A}$. If any error occurs during the above execution, then $\mathcal{B}_5$ aborts and undoes the above executions.
  – If P is an unauthorized participant in the group gid, then $\mathcal{B}_5$ executes $\mathsf{KeyRotat\text{-}P}(P, \mathsf{gid}, m)$ by himself.

  For the second case that $\mathsf{gid} \neq \widetilde{\mathsf{gid}}$, we further consider following two sub-cases:
  – If P is the leader in the group gid, then $\mathcal{B}_5$ first queries $\textsc{SendKeyRotat}(\mathsf{id}_P, \mathsf{gid}, m)$ to its challenger for a reply c. Then, $\mathcal{B}_5$ extracts the portion $c_{\overline{P}}$ in c that is specific to every participant $\overline{P}$ in the group gid, followed by encrypting it using the corresponding $\mathsf{AEAD}_{\mathsf{PP}}$ key k_{PP}, a random nonce, and an associated data that is same as the one in the corresponding Participant Join phase. Finally, $\mathcal{B}_5$ outputs the $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext and nonce for every participant $\overline{P}$ in the group gid. If any error occurs during the above execution, then $\mathcal{B}_5$ aborts and undoes the above executions.
  – If P is a participant in the group gid, then $\mathcal{B}_5$ first extracts an $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext and an $\mathsf{AEAD}_{\mathsf{PP}}$ nonce from the input m. Next, $\mathcal{B}_5$ recovers a message m_1 from the $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext using the corresponding $\mathsf{AEAD}_{\mathsf{PP}}$ key k_{PP}, the $\mathsf{AEAD}_{\mathsf{PP}}$ nonce, and an associated data that is same as the one in the corresponding Participant Join phase. Then, $\mathcal{B}_5$ extracts other necessary information m_2 from the input m for querying $\textsc{SendKeyRotat}(\mathsf{id}_P, \mathsf{gid}, m_1 \parallel m_2)$ returns the reply m_{KRot} to $\mathcal{A}$. If any error occurs during the above execution, then $\mathcal{B}_5$ aborts and undoes the above executions.
- $\textsc{Corrupt}(\mathsf{id}_P)$: $\mathcal{B}_5$ simply forwards this query to its challenger and the reply to $\mathcal{A}$.
- $\textsc{Compromise}(\mathsf{id}_P, \mathsf{gid})$: We consider the following two cases. For the first case that $\mathsf{gid} = \widetilde{\mathsf{gid}}$, we further consider the following two sub-cases:
  – If P is an authorized party for the group gid, then $\mathcal{B}_5$ forwards this query to its challenger. If the reply is not $\perp$, then $\mathcal{B}_5$ forwards the reply together with the key k_{PP} of P in the group gid to $\mathcal{A}$. Otherwise, $\mathcal{B}_5$ simply returns $\perp$ to $\mathcal{A}$.
  – If P is an unauthorized party for the group gid, then $\mathcal{B}_5$ must create π_P^{gid} by himself. In this case, $\mathcal{B}_5$ simply returns π_P^{gid} to $\mathcal{A}$.

  For the second case that $\mathsf{gid} \neq \widetilde{\mathsf{gid}}$, $\mathcal{B}_5$ forwards this query to its challenger. If the reply is not $\perp$, then $\mathcal{B}_5$ forwards the reply together with the key k_{PP} of P in the group gid to $\mathcal{A}$. Otherwise, $\mathcal{B}_5$ simply returns $\perp$ to $\mathcal{A}$.
- $\textsc{Leak}(\mathsf{id}_P, \mathsf{gid}, \mathsf{gkid})$: We consider the following two cases. For the first case that $\mathsf{gid} = \widetilde{\mathsf{gid}}$, we further consider the following two cases:
  – If P is an authorized party for the group gid, then $\mathcal{B}_5$ simply forwards this query to its challenger and the reply to $\mathcal{A}$.
  – If P is an unauthorized party for the group gid, then $\mathcal{B}_5$ must create $gk_P^{(\mathsf{gid},\mathsf{gkid})}$ by himself. In this case, $\mathcal{B}_5$ simply returns $gk_P^{(\mathsf{gid},\mathsf{gkid})}$ to $\mathcal{A}$.

For the second case that $\mathsf{gid} \neq \widetilde{\mathsf{gid}}$, $\mathcal{B}_5$ simply forwards this query to its challenger and the reply to $\mathcal{A}$.

- REVEAL(gid): $\mathcal{B}_5$ forwards this query to its challenger. Then, $\mathcal{B}_5$ forwards the reply together with the password pw^{gid} to $\mathcal{A}$.
- TEST($\mathsf{id}_P, \mathsf{gid}, \mathsf{gkid}$): $\mathcal{B}_5$ simply forwards this query to its challenger and the reply to $\mathcal{A}$.

Note that $\mathcal{B}_5$ perfectly simulates **Game** $C2.5$ to $\mathcal{A}$. Moreover, $\mathcal{B}_5$ simulates the behaviors of all unauthorized parties in the group $\widetilde{\mathsf{gid}}$ by himself. Thus, the parties in the group $\widetilde{\mathsf{gid}}$ in the Sec-mGKD-pw experiment between $\mathcal{B}_5$ and its challenger are identical to the authorized parties in the group $\widetilde{\mathsf{gid}}$ in the Sec-mGKD-pki experiment between $\mathcal{A}$ and $\mathcal{B}_5$.

Note in **Game** 4 that we ensure that the sign-up messages are honestly distributed between the leader and authorized participants $P \in GP^{(\widetilde{\mathsf{gid}},\mathsf{gkid})}$ for the tested group $\widetilde{\mathsf{gid}}$ and group key index gkid. Thus, for the tested party P', the tested group $\widetilde{\mathsf{gid}}$, and the tested group key index gkid, if $\mathcal{A}$ can trigger E_{KPriv} without violating the freshness condition $\mathsf{fresh}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}_{\mathsf{KPriv}}(\mathsf{id}_{P'}, \mathsf{gid}, \mathsf{gkid})$, then $\mathcal{B}$ can also trigger E_{KPriv} by outputting the same b' as $\mathcal{A}$, without violating the freshness condition $\mathsf{fresh}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}_{\mathsf{KPriv}}(\mathsf{id}_{P'}, \mathsf{gid}, \mathsf{gkid})$. Thus, it holds that

$$\mathsf{Adv}^{C2}_5(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}_{\Pi}(\mathcal{B}_5)$$

Final Analysis Of The Full Proof. By merging the statements above, if $\mathcal{A}$ can break the Sec-mGKD-pw security of Π', then there exists an attacker $\mathcal{B}$ that breaks the Sec-mGKD-pki security of Π, such that

$$\mathsf{Adv}^{\mathsf{Sec\text{-}mGKD\text{-}pw}}_{\Pi'}(\mathcal{A}) \leq q_{\textsc{NewGroup}}\Big(\epsilon^{\mathsf{w\text{-}PAKE}}_{\mathsf{PAKE}_{\mathsf{PP}},\mathcal{D}_{pw}} + \epsilon^{\mathsf{d\text{-}frob}}_{\mathsf{AEAD}_{\mathsf{PP}}} + c_{\mathsf{maxReg}}(\epsilon^{\mathsf{cti\text{-}cpa}}_{\mathsf{AEAD}_{\mathsf{PP}}} + \epsilon^{\mathsf{ind\$\text{-}cca}}_{\mathsf{AEAD}_{\mathsf{PP}}}) + \mathsf{Adv}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}_{\Pi}(\mathcal{B})\Big)$$

Appendix E. Proof of Theorem 3

The proof is given by reduction. If there exists any PPT attacker $\mathcal{A}$ that breaks the Sec-mGKD-pki of Π', then we construct an attacker $\mathcal{B}$ that breaks the Sec-mGKD-pki of Π. The attacker $\mathcal{B}$ initializes the Sec-mGKD-pki experiment and answers $\mathcal{A}$'s oracle queries as follows:

- NEWPARTY(id_P): $\mathcal{B}$ simply forwards this query to its challenger and then forwards the reply to $\mathcal{A}$.
- NEWGROUP($\mathsf{id}_P, \mathsf{gid}$): $\mathcal{B}$ forwards this query to its challenger. Then, $\mathcal{B}$ samples a password pw^{gid} from the distribution $\mathcal{D}_{pw}$ and associate the password with the group gid.
- AUTHORIZE($\mathsf{gid}, \mathsf{id}_P$): $\mathcal{B}$ simply forwards this query to its challenger.
- REGISTERAUTH($\mathsf{id}_P, \mathsf{gid}, m$): $\mathcal{B}$ forwards this query to its challenger for a reply c_1. If P is the leader of the group gid, $\mathcal{B}$ simply returns c_1. Otherwise, $\mathcal{B}$ runs the first pass of $\mathsf{PAKE}_{\mathsf{PP}}$ upon the password pw^{gid} for a ciphertext c_2. Finally, $\mathcal{B}$ returns (c_1, c_2) to $\mathcal{A}$.
- REGISTERINJECT($\mathsf{id}_P, \mathsf{gid}, gs, m$): $\mathcal{B}$ first parses gs into two portions gs_{Π} and pw. Next, $\mathcal{B}$ sends query REGISTERINJECT($\mathsf{id}_P, \mathsf{gid}, gs_{\Pi}, m$) to its challenger for a reply c_1. Then, $\mathcal{B}$ runs the first pass of $\mathsf{PAKE}_{\mathsf{PP}}$ upon the password pw for a ciphertext c_2. Finally, $\mathcal{B}$ returns (c_1, c_2) to $\mathcal{A}$.
- SENDJOINAUTH($\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, m$): We consider two steps.
  The first step is executed if the party P has not output the key k_{PP} of the $\mathsf{PAKE}_{\mathsf{PP}}$ with P' in the group gid. $\mathcal{B}$ first extracts necessary information from the input m and runs the next pass of $\mathsf{PAKE}_{\mathsf{PP}}$ upon pw^{gid}. If the key k_{PP} is available, $\mathcal{B}$ remembers this key for both parties P and P' in the group gid. Otherwise, $\mathcal{B}$ simply outputs the outgoing message of $\mathsf{PAKE}_{\mathsf{PP}}$.
  The second step is executed if the party P has already output the key k_{PP} of the $\mathsf{PAKE}_{\mathsf{PP}}$ with P' in the group gid. $\mathcal{B}$ first checks whether the input m includes any $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext and the $\mathsf{AEAD}_{\mathsf{PP}}$ nonce. If so, $\mathcal{B}$ first decrypts it using the key k_{PP}, the $\mathsf{AEAD}_{\mathsf{PP}}$ nonce, the associated data consisting both P and P''s sign-up messages, for a message m_1. Otherwise, the message m_1 is set to empty string ϵ. Then, $\mathcal{B}$ sends the query SENDJOINAUTH($\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, m_1$) to its challenger for a reply m_2. Finally, $\mathcal{B}$ encrypts the message m_2 using the key k_{PP}, a random nonce, the associated data consisting both P and P''s sign-up messages, for a ciphertext.
  The ciphertexts of $\mathsf{AEAD}_{\mathsf{PP}}$ and optionally the one of $\mathsf{PAKE}_{\mathsf{PP}}$ (if available) are returned to $\mathcal{A}$.
- SENDJOININJECT($\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, gs, m$): $\mathcal{B}$ first parses gs into two portions gs_{Π} and pw. Then, we consider two steps.
  The first step is executed if the party P has not output the key k_{PP} of the $\mathsf{PAKE}_{\mathsf{PP}}$ with P' in the group gid. $\mathcal{B}$ first extracts necessary information from the input m and runs the next pass of $\mathsf{PAKE}_{\mathsf{PP}}$ upon pw. If the key k_{PP} is available, $\mathcal{B}$ remembers this key for both parties P and P' in the group gid. Otherwise, $\mathcal{B}$ simply outputs the outgoing message of $\mathsf{PAKE}_{\mathsf{PP}}$.
  The second step is executed if the party P has already output the key k_{PP} of the $\mathsf{PAKE}_{\mathsf{PP}}$ with P' in the group gid. $\mathcal{B}$ first checks whether the input m includes any $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext and the $\mathsf{AEAD}_{\mathsf{PP}}$ nonce. If so, $\mathcal{B}$ first decrypts it using the key k_{PP}, the $\mathsf{AEAD}_{\mathsf{PP}}$ nonce, the associated data consisting both P and P''s sign-up messages, for a message m_1. Otherwise, the message m_1 is set to empty string ϵ. Then, $\mathcal{B}$ sends the query SENDJOININJECT($\mathsf{id}_P, \mathsf{id}_{P'}, \mathsf{gid}, gs_{\Pi}, m_1$) to its challenger for a reply m_2. Finally, $\mathcal{B}$ encrypts the message m_2 using the key k_{PP}, a random nonce, the associated data consisting both P and P''s sign-up messages, for a ciphertext.
  The ciphertexts of $\mathsf{AEAD}_{\mathsf{PP}}$ and optionally the one of $\mathsf{PAKE}_{\mathsf{PP}}$ (if available) are returned to $\mathcal{A}$.
- SENDLEAVE($\mathsf{id}_P, \mathsf{gid}, \mathsf{id}_{P'}$): $\mathcal{B}$ forwards this query to its challenger. If $\mathsf{id}_P = \mathsf{id}_{P'}$, then $\mathcal{B}$ also removes the key k_{PP} of the party P generated in the Participant Join phase for Pjoining the group gid. If P is the leader of the group

gid, then $\mathcal{B}$ removes the key k_{PP} of the party P generated in the Participant Join phase for P' the group gid.

- ENDGROUP(gid): $\mathcal{B}$ forwards this query to its challenger. Then $\mathcal{B}$ also removes all remaining keys k_{PP} of the leader of the group gid.
- SENDKEYROTAT($\mathsf{id}_P, \mathsf{gid}, m$): If P is the leader of the group gid, then $\mathcal{B}$ forwards this query to its challenger for a reply c. Then, the $\mathcal{B}$ extracts the portion $c_{\overline{P}}$ in c that is specific to every participant $\overline{P}$ in the group gid, followed by encrypting it using the stored corresponding $\mathsf{AEAD}_{\mathsf{PP}}$ key k_{PP}, a random nonce, and an associated data consisting of the leader P and the participant $\overline{P}$' sign-up messages as in the Participant Join phase. Finally, $\mathcal{B}$ outputs the $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext and nonce for every participant $\overline{P}$ in the group gid. If any error occurs during the above execution, then the leader P aborts and undoes the above executions.
  Otherwise, P is the participant of the group gid. $\mathcal{B}$ first extracts an $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext and an $\mathsf{AEAD}_{\mathsf{PP}}$ nonce from the input m. Next, $\mathcal{B}$ recovers a message m_1 from the $\mathsf{AEAD}_{\mathsf{PP}}$ ciphertext using the stored corresponding $\mathsf{AEAD}_{\mathsf{PP}}$ key k_{PP}, the $\mathsf{AEAD}_{\mathsf{PP}}$ nonce, and an associated data consisting of the participant P's and the leader's sign-up messages as in the Participant Join phase. Then, $\mathcal{B}$ extracts other necessary information m_2 from the input m for querying SENDKEYROTAT$(P, \mathsf{gid}, m_1 \parallel m_2)$ to its challenger. Finally, $\mathcal{B}$ forwards the reply from the challenger to $\mathcal{A}$. If any error occurs during the above execution, then the participant P aborts and undoes the above executions.
- CORRUPT(id_P): $\mathcal{B}$ simply forwards this query to its challenger and then forwards the reply to $\mathcal{A}$.
- COMPROMISE($\mathsf{id}_P, \mathsf{gid}$): $\mathcal{B}$ simply forwards this query to its challenger for a state π. Then, $\mathcal{B}$ forwards the state π together with all keys k_{PP} of party P for the group gid to $\mathcal{A}$.
- LEAK($\mathsf{id}_P, \mathsf{gid}, \mathsf{gkid}$): $\mathcal{B}$ simply forwards this query to its challenger and then forwards the reply to $\mathcal{A}$.
- REVEAL(gid): $\mathcal{B}$ simply forwards this query to its challenger for a group secret gs_{Π}. Then, $\mathcal{B}$ forwards the group secret gs_{Π} together with the password pw^{gid} to $\mathcal{A}$.
- TEST($\mathsf{id}_P, \mathsf{gid}, \mathsf{gkid}$): $\mathcal{B}$ simply forwards this query to its challenger and then forwards the reply to $\mathcal{A}$.

It is easy to know that $\mathcal{B}$ perfectly simulates Sec-mGKD-pki experiment to $\mathcal{A}$ and wins if $\mathcal{A}$ wins. The proof is concluded by

$$\mathsf{Adv}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}_{\mathsf{mGKD}'}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{Sec\text{-}mGKD\text{-}pki}}_{\mathsf{mGKD}}(\mathcal{B})$$

Contents