Multi-Cast Key Distribution: Scalable, Dynamic and Provably Secure Construction[⋆]

Kazuki Yoneyama[1], Reo Yoshida[2], Yuto Kawahara[2], Tetsutaro Kobayashi[2], Hitoshi Fuji[2], and Tomohide Yamamoto[2]

[1] Ibaraki University, Ibaraki, Japan
`kazuki.yoneyama.sec@vc.ibaraki.ac.jp`
[2] NTT Secure Platform Laboratories, Tokyo, Japan

Abstract. In this paper, we propose a two-round dynamic multi-cast key distribution (DMKD) protocol under the star topology with a central authentication server. Users can share a common session key without revealing any information of the session key to the server, and can join/leave to/from the group at any time even after establishing the session key. Our protocol is scalable because communication and computation costs of each user are independent from the number of users. Also, our protocol is still secure if either private key or session-specific randomness of a user is exposed. Furthermore, time-based backward secrecy is guaranteed by renewing the session key for every time period even if the session key is exposed. We introduce the first formal security definition for DMKD under the star topology in order to capture such strong exposure resilience and time-based backward secrecy. We prove that our protocol is secure in our security model in the standard model.

keywords: multi-cast key distribution, exposure resilience, star topology, backward secrecy

1 Introduction

HTML 5 is an emerging technology for next generation web applications [BLN+12]. Actually, web browser vendors support this new technology. Google said its Chrome browser would begin blocking Internet ads using Adobe's Flash tech, likely prompting advertisers to abandon the video format [Mar15]. Similarly, Mozilla, the Firefox vendor, is encouraging developers to adopt HTML5 and not to use Flash [Che15].

In HTML5, we have a simple method using WebRTC [BBJ+15] for a *full-mesh real time communication topology* [WW15]. WebRTC provides the confidentiality of real time transport protocol (RTP) [SCFJ03] by using a key exchange and encrypted transport protocol, DTLS-SRTP [FTR10], which has been suggested in IETF Draft [Res15]. In order to make the full-mesh encrypted communication topology, WebRTC needs full-mesh DTLS key exchanges to establish all SRTP sessions. In brief, WebRTC clients must exchange the keys with $n - 1$ users in the n clients case. This is very inefficient.

⋆ An extended abstract of this paper appeared in ProvSec 2016 [YYK+16].

Such key exchange protocols under the mesh topology are generally called group key exchange (GKE).

In this paper, we consider the *star topology* instead of the mesh topology for establishing the session key. In the star topology, each user communicates with a central authentication server, and users do not directly communicate with. Thus, it is possible to reduce costs of clients without depending on n; and therefore, WebRTC clients can do the key exchange part very efficiently. Key exchange protocols under the star (or tree) topology are generally called multi-cast key distribution (MKD) or group key management. Though the star topology can reduce the cost for clients, moving most of the burden to the server makes the server a natural target for a concentrated attack. Thus, the star topology is useful if the system is still secure when some part of secret information of the server is exposed by some attack.

Our Contribution. In this paper, we propose a new *provably secure two-round dynamic* MKD (DMKD) protocol under the star topology with a central authentication server. Because of the star topology, each user does not directly communicate with other users. Instead, the central server communicates with users, and distributes information for establishing the session key. If the server was malicious under the star topology, the session key could be known for the server by impersonating a user. Thus, we suppose that the server is honest-but-curious, and even the server must not know any information of the session key.

Each user has public information, called a *static public key (SPK)*, and the corresponding secret information, called a *static secret key (SSK)*. The SPK is also expected to be certified with user's ID through an infrastructure such as PKI. A user who wants to share a *session key* with other users exchanges *ephemeral public keys (EPKs)* that is generated from the corresponding session-specific randomness, called *ephemeral secret keys (ESKs)*, via the server.

The highlight of our protocol is as follows.

– **Dynamic Group.** Our protocol allows users to share the session key in the dynamic group manner. It means that, after establishing the session key among a group of users in a distribution phase, a set of users can join/leave the group without executing a new distribution phase among the new group. Users generate and keep state information for the join and leave phases at the end of the distribution phase. The join and leave phases of our protocol need smaller computation and communication costs than the distribution phase thanks to state information. Also, since the session key is refreshed after the join/leave phases, any information of the new session key do not exposed to leaving users.
– **Strong Exposure Resilience.** In real-world applications, there are several situations that secret information is exposed. For example, if a pseudo-random number generator implemented in a system is poor, ESKs may be guessed to the adversary. Also, when some devices containing SSKs are lost, then a malicious person may use SSKs to know the session key generated by the owner. Furthermore, the government may order the server to reveal the SSK. Thus, it is desirable that DMKD protocols are resilient to secret key exposure attacks. Our protocol is secure even if either of SSKs or ESKs used to generate the session key are exposed. We call

security when ESKs are exposed *ephemeral key exposure resilience*, and security when SSKs (including the server's) are exposed *strong server key forward secrecy*. To achieve ephemeral key exposure resilience, we use *the twisted pseudo-random function (PRF) trick* [FSXY15,KF14]. Strong server key forward secrecy guarantees even if the adversary is allowed to modify messages in the target session while most of AKE protocols prevent that (i.e., weak forward secrecy). Moreover, our protocol guarantees a distinguished security property, called *time-based backward secrecy*. It means that if the session key is exposed at a time frame, the exposed session key is revoked when a new time frame begins. Time-based backward secrecy is very useful to resist real-time session key exposure attacks like malwares. We achieve time-based backward secrecy by formalizing the notion of the time frame in the security model, and proposing a method to update the session key with a minimum cost.
– **Scalability.** Most of previous GKE protocols are constructed under the mesh topology. A user must combine information from users in order to establish the session key with contributions of all users. Thus, the user needs to broadcast a message to all users (i.e., computation and communication costs depend on the number of users), or the round complexity depends on the number of users. Hence, if we adopt the mesh topology, it is difficult to achieve scalability. On the other hand, our DMKD protocol is constructed under the star topology. Though the server needs computation and communication costs depending on the number of users, users can share the session key with constant costs. The load of the server is actually not a problem because the server can be very powerful in computational resource and communication bandwidth. Conversely, users may have poor resources like a mobile device; and thus, scalability is very important in reality.

Also, we propose a first formal security model for DMKD. Our security model captures the star topology and several exposure resilience. Especially, to grasp time-based backward secrecy, the notion of time frames is formulated to define session freshness.

Related Work. We revisit several related work of this work.

Group Key Exchange. The first provably secure GKE protocol is proposed by Bresson et al. [BCPQ01]. Their protocol is not dynamic (i.e., the group member is fixed before starting sessions.). Then, several dynamic GKE (DGKE) protocols and security models are proposed [BCP01,BCP02]. These protocols need a linear number of rounds. After that, several constant round DGKE protocols are studied [KLL04,DB05,YT10]. Exposure resilience of GKE protocols is firstly considered in the security model by Manulis et al. [MSU09]. Their model guarantees ephemeral key exposure resilience and weak forward secrecy. This security model is extended by Suzuki and Yoneyama [SY13] to grasp session state exposure. However, their proposed GKE protocol is not for general group setting but three-party setting. Since these GKE protocols are considered under the mesh topology, costs of users depend on the number of users.

Multicast Key Distribution. Since the main application of MKD is Mobile Ad Hoc Networks (MANET), most of MKD protocols use tree topology. The advantage of the tree-based MKD is that total communication complexity is reduced to $O(\log n)$. For example, MKD protocols based on logical key hierarchies [CWSP98,CGI+99,WCS+99,SM03] have been well studied. There are few papers studying MKD in the star topology [LTW10,SP12,MK14,SHC+15]. The motivation of previous star topology-based MKD protocols is to reduce the rekeying cost of tree topology-based MKD protocols. Most of these protocols have no formal security proof. Sun et al. [SHC+15] propose a provably secure star topology-based MKD protocol. However, their security model does not capture exposure resilience, and their protocol is not scalable because communication complexity for uses depend on the number of users. A formal security model for MKD protocols is introduced by Micciancio and Panjwani [MP04]. However, their model allows the server to know the session key shared by users. Also, exposure resilience is not considered.

2 Preliminaries

2.1 Notations

Throughout this paper we use the following notations. If Set is a set, then by $m \in_R \mathsf{Set}$ we denote that m is sampled uniformly from Set. If **ALG** is an algorithm, then by $y \leftarrow \mathbf{ALG}(x; r)$ we denote that y is output by **ALG** on input x and randomness r (if **ALG** is deterministic, r is empty).

2.2 Pseudo-Random Function, and Twisted Pseudo-Random Function Trick

Let κ be a security parameter and $\mathsf{F} = \{F_\kappa : Dom_\kappa \times Kspace_\kappa \to Rng_\kappa\}_\kappa$ be a function family with a family of domains $\{Dom_\kappa\}_\kappa$, a family of key spaces $\{Kspace_\kappa\}_\kappa$ and a family of ranges $\{Rng_\kappa\}_\kappa$.

Definition 1 (Pseudo-Random Function). *We say that function family* $\mathsf{F} = \{F_\kappa\}_\kappa$ *is the PRF family, if for any PPT distinguisher* $\mathcal{D}$, $|\Pr[1 \leftarrow \mathcal{D}^{F_\kappa(\cdot)}] - \Pr[1 \leftarrow \mathcal{D}^{RF_\kappa(\cdot)}]| \leq negl$, *where* $RF_\kappa : Dom_\kappa \to Rng_\kappa$ *is a truly random function.*

Next, we show the notion of the twisted PRF [FSXY15]. The twisted PRF $tPRF$ is a function that $tPRF : \{0,1\}^\kappa \times Kspace_\kappa \times \{0,1\}^\kappa \times Kspace_\kappa \to Rng_\kappa$. We construct $tPRF(a, a', b, b') := F_\kappa(a, b) \oplus F_\kappa(b', a')$ with PRF F_κ, where $a, b' \in \{0,1\}^\kappa$ and $a', b \in Kspace_\kappa$. The twisted PRF is used to guarantee that $tPRF(a, a', b, b')$ looks random even if either (a, a') or (b, b') is exposed.

Lemma 1 (Theorem 1 in [KF14]). *If* F_κ *is PRF, then*

- $[(a, a'), tPRF(a, a', b, b')]$ *is indistinguishable from* $[(a, a'), R]$ *where* R *is randomly chosen from* Rng_κ*, and*
- $[(b, b'), tPRF(a, a', b, b')]$ *is indistinguishable from* $[(b, b'), R]$ *where* R *is randomly chosen from* Rng_κ*.*

2.3 Target-Collision Resistant Hash Function

We say a function $TCR : Dom \rightarrow Rng$ is a target-collision resistant (TCR) hash function if the following condition holds for security parameter κ: For any PPT adversary $\mathcal{A}$, $\Pr[x \in_R Dom; x' \leftarrow \mathcal{A}(x) \text{ s.t. } x \neq x' \wedge TCR(x) = TCR(x')] \leq negl.$

2.4 Public Key Encryption

Definition 2 (Syntax for Public Key Encryption Schemes). *A PKE scheme consists of the following 3-tuple* $(\mathbf{Gen}, \mathbf{Enc}, \mathbf{Dec})$*:*

Gen : *a key generation algorithm which on input 1^κ, where κ is the security parameter, outputs a pair of public and secret keys (pk, sk).*
Enc : *an encryption algorithm which takes as input public key pk and plaintext m, outputs ciphertext CT.*
Dec : *a decryption algorithm which takes as input secret key sk and ciphertext CT, outputs plaintext m or reject symbol $\perp$.*

Definition 3 (Chosen-Ciphertext Security for Public Key Encryption). *A PKE scheme is CCA-secure if the following property holds for security parameter κ; For any adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, $|\Pr[(pk, sk) \leftarrow \mathbf{Gen}(1^\kappa); (m_0, m_1, state) \leftarrow \mathcal{A}_1^{\mathcal{DO}(sk,\cdot)}(pk); b \in_R \{0,1\}; CT^* \leftarrow \mathbf{Enc}(pk, m_b); b' \leftarrow \mathcal{A}_2^{\mathcal{DO}(sk,\cdot)}(pk, CT^*, state); b' = b] - 1/2| \leq negl(\kappa)$, where $\mathcal{DO}$ is the decryption oracle which outputs m or $\perp$ on receiving CT, state is state information (possibly including pk, m_0 and m_1) which $\mathcal{A}$ wants to preserve. $\mathcal{A}$ cannot submit the ciphertext $CT = CT^*$ to $\mathcal{DO}$.*

2.5 Ciphertext-Policy Attribute-based Encryption

Definition 4 (Syntax for Ciphertext-Policy Attribute-based Encryption Schemes). *A CP-ABE scheme consists of the following 4-tuple* $(\mathbf{Setup}, \mathbf{Der}, \mathbf{AEnc}, \mathbf{ADec})$*:*

$(Params, msk) \leftarrow \mathbf{Setup}(1^\kappa, att)$: *a setup algorithm which on inputs 1^κ and att, where κ is the security parameter and att is an attribute universe description, outputs a public parameter Params and a master secret key msk.*
$usk_A \leftarrow \mathbf{Der}(Params, msk, A)$: *a key derivation algorithm which on input Params, msk and attribute A, outputs a user secret key usk_A.*
$CT \leftarrow \mathbf{AEnc}(Params, P, m)$: *an encryption algorithm which on input Params, an access structure P and a plaintext m, outputs a ciphertext CT.*
$m \leftarrow \mathbf{ADec}(Params, usk_A, CT)$: *a decryption algorithm which on input usk_A and CT, outputs plaintext m if A satisfies P.*

Definition 5 (Chosen-Ciphertext Security for Ciphertext-Policy Attribute-based Encryption). *A CP-ABE scheme is CCA-secure if the following property holds for security parameter κ; For any PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, $|\Pr[(Params, msk) \leftarrow \mathbf{Setup}(1^\kappa, att); (m_0, m_1, P^*, s) \leftarrow \mathcal{A}_1^{\mathcal{EO}(Params,msk,\cdot),\mathcal{DO}(Params,usk_\cdot,\cdot)}(Params); b \in_R \{0,1\}; CT^* \leftarrow \mathbf{AEnc}(Params, P^*, m_b); b' \leftarrow \mathcal{A}_2^{\mathcal{EO}(Params,msk,\cdot),\mathcal{DO}(Params,usk_\cdot,\cdot)}(Params, CT^*, s); b' = b] - 1/2| \leq negl$, where $\mathcal{EO}$ is the key extraction oracle, $\mathcal{DO}$ is the decryption* *oracle and s is state information that $\mathcal{A}$ wants to preserve from $\mathcal{A}_1$ to $\mathcal{A}_2$. $\mathcal{A}$ cannot submit sets of attributes which satisfy P^* to $\mathcal{EO}$ and the ciphertext CT^* to $\mathcal{DO}$.*

We say a CP-ABE scheme is CPA-secure if $\mathcal{A}$ does not access $\mathcal{DO}$. Also, we say a CP-ABE scheme is selectively secure if the adversary must commit P^* before **Setup**.

2.6 Message Authentication Codes

Definition 6 (Syntax for Message Authentication Codes). *A MAC scheme consists of the following 3-tuple* (**MGen**, **Tag**, **Ver**)*:*

MGen : *a key generation algorithm which on input 1^κ, where κ is the security parameter, outputs a MAC key mk.*

Tag : *a tagging algorithm which on input mk and plaintext m, outputs an authentication-tag σ.*

Ver : *a verification algorithm which on input mk, m and σ, outputs 1 if accepts, 0 otherwise.*

Definition 7 (Unforgeability against Chosen-Message Attacks for Message Authentication Codes). *A MAC scheme is UF-CMA if the following property holds for security parameter κ; For any PPT forger $\mathcal{A}$, $\Pr[mk \leftarrow \mathbf{MGen}(1^\kappa); (m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{MO}(mk,\cdot)}; 1 \leftarrow \mathbf{Ver}(mk, m^*, \sigma^*)] \leq negl$, where $\mathcal{MO}$ is the MAC oracle. $\mathcal{A}$ cannot submit m^* to $\mathcal{MO}$.*

2.7 Decisional Diffie-Hellman Assumption

Definition 8 (Decisional Diffie-Hellman Assumption). *Let p be a prime and let g be a generator of a finite cyclic group G of order p. We define two experiments, $\mathsf{Exp}^{ddh\text{-}real}_{g,p}(\mathcal{D})$ and $\mathsf{Exp}^{ddh\text{-}rand}_{g,p}(\mathcal{D})$. For a distinguisher $\mathcal{D}$, inputs $(g, A = g^a, B = g^b, C)$ are provided, where $(a, b) \in_R (\mathbb{Z}_p)^2$. $C = g^{ab}$ in $\mathsf{Exp}^{ddh\text{-}real}_{g,p}(\mathcal{D})$ and $C = g^c$ in $\mathsf{Exp}^{ddh\text{-}rand}_{g,p}(\mathcal{D})$, where $c \in_R \mathbb{Z}_p$. Let $(g, A = g^a, B = g^b, C = g^{ab})$ be the tuple in $\mathsf{Exp}^{ddh\text{-}real}_{g,p}(\mathcal{D})$ and $(g, A = g^a, B = g^b, C = g^c)$ be the tuple in $\mathsf{Exp}^{ddh\text{-}rand}_{g,p}(\mathcal{D})$. We say that the DDH assumption in G holds for security parameter κ if for any PPT distinguisher $\mathcal{D}$ $|\Pr[\mathsf{Exp}^{ddh\text{-}real}_{g,p}(\mathcal{D}) = 1] - \Pr[\mathsf{Exp}^{ddh\text{-}rand}_{g,p}(\mathcal{D}) = 1]| \leq negl$.*

3 Security Definition

In this section, we introduce a new security definition of DMKD under the star topology. Our definition captures strong exposure-resilience and time-based forward secrecy. The model is based on [YT10,SY13,SY14].

3.1 Protocol Participants and Initialization

Let $\mathcal{U} := \{U_1, \ldots, U_N\}$ be a set of potential protocol users. Each user U_i is modelled as a PPT Turing machine w.r.t. security parameter κ. For user U_i, we denote static secret (public) key by SSK_i (SPK_i). U_i generates its own keys, SSK_i and SPK_i, and the static public key SPK_i is linked with U_i's identity in some systems like PKI. Each user U_i and the authentication server S are connected by unauthenticated the star topology. That is, they do communications through an unicast channel over an insecure network like the Internet. Users do not directly communicate. S is also modelled as a PPT Turing machine. S has the static secret key by SSK_S and the static public key SPK_S.

3.2 Session and State Information

There are three phases (Dist, Join, Leave) for DMKD. Dist means the session key distribution phase that a new group is established and a session key is generated for users in the group. Join means the user joining phase that a set of new users join an established group and a session key is re-generated for users in the new group. Leave means the user leaving phase that a set of users leave an established group and a session key is re-generated for remaining users in the group. An invocation of a phase is called a *session*. We suppose that a session contains n users $\{U_{i_1}, \ldots, U_{i_n}\}$, where $2 \leq n \leq N$. Let Π be a phase identifier such that $\Pi \in \{\mathsf{Dist}, \mathsf{Join}, \mathsf{Leave}\}$. A session owned by user instance $U_{i_\ell}^{j_\ell}$ is managed by a tuple $(\Pi, U_{i_\ell}^{j_\ell}, \{U_{i_1}^{j_1}, \ldots, U_{i_n}^{j_n}\})$. $U_{i_\ell}^{j_\ell}$ means the j_ℓ-th instance of U_{i_ℓ}. Sessions owned by user instances $\{U_{i_1}^{j_1}, \ldots, U_{i_{\ell-1}}^{j_{\ell-1}}, U_{i_{\ell+1}}^{j_{\ell+1}}, \ldots, U_{i_n}^{j_n}\}$ are called matching sessions of the session of $U_{i_\ell}^{j_\ell}$. Hereafter, for simplicity, we can describe U_{i_ℓ} as U_i without loss of generality. We suppose that the total number of sessions in the system is ℓ_{max}. We consider the notion of *time frames*. Each user U_i and S communicate to update some state information $state_i$ when the session is firstly executed in a time frame. Hereafter, U_i uses $state_i$ in sessions within the time frame. Also, we consider the session key update based on time frames. Update means the session key update phase that the shared session key is updated when a new time frame begins. If a session key is shared in the Dist/Join/Leave phase in the past time frame, the session key is updated by Update. We note that the session is not changed after Update phase but only the session key is changed. In Dist phase, U_i^j generates ephemeral secret key ESK_i^j and sends ephemeral public key EPK_i^j to S. When S receives all EPK_i^j for $i, j = 1, \ldots, n$, then S returns messages to users. Users and S repeat some rounds, and then users finally share session key SK and complete the session. After completing the session, each user U_i updates $state_i$ to remain necessary information for Update, Join and Leave phases. $state_i$ is passed to another inactivated instance $U_i^{j'}$ to participate in the next activation of Update, Join or Leave phase. Similarly, in Update, Join and Leave phases, users and S execute some interactions, and users update the session key. DMKD consists of many concurrent executions of Dist, Update, Join and Leave phases.

3.3 Adversary

The adversary $\mathcal{A}$, which is modelled as a PPT Turing machine, controls all communications between parties including session activation and registrations of users by performing the following adversary queries.

- Establish(U_i, SPK_i): This query allows $\mathcal{A}$ to introduce a new user. In response, if $U_i \notin \mathcal{U}$ (due to the uniqueness of identities) then U_i with the static public key

SPK_i is added to $\mathcal{U}$. Note that $\mathcal{A}$ is not required to prove the possession of the corresponding secret key SSK_i. If a party is registered by a $\mathsf{Establish}$ query issued by $\mathcal{A}$, then we call the party *dishonest*. If not, we call the party *honest*.

– $\mathsf{Send}(U_i^j, message)$: This query allows $\mathcal{A}$ to send *message* to instance U_i^j. *message* includes $\Pi \in \{\mathsf{Dist}, \mathsf{Join}, \mathsf{Leave}\}$. $\mathcal{A}$ obtains the response from U_i^j. If U_i^j is an inactivated instance and $\Pi = \{\mathsf{Join}, \mathsf{Leave}\}$, $state_i$ is passed to U_i^j.

To capture exposure of secret information, the adversary $\mathcal{A}$ is allowed to issue the following queries.

– $\mathsf{SessionReveal}(U_i^j)$: The adversary $\mathcal{A}$ obtains the session key SK for the session owned by U_i^j if the session is completed.
– $\mathsf{StateReveal}(U_i)$: The adversary $\mathcal{A}$ obtains current state information $state_i$ of U_i. State information do not include the static secret key.
– $\mathsf{ServerReveal}$: This query allows the adversary $\mathcal{A}$ to obtain static secret key SSK_S of the server S.
– $\mathsf{StaticReveal}(U_i)$: This query allows the adversary $\mathcal{A}$ to obtain static secret key SSK_i of the user U_i.
– $\mathsf{EphemeralReveal}(U_i^j)$: This query allows the adversary $\mathcal{A}$ to obtain ephemeral secret key ESK_i^j of U_i^j if the session is not completed (i.e., the session key is not established yet).

3.4 Freshness

For the security definition, we need the notion of freshness.

Definition 9 (Freshness). *Let* $\mathsf{sid}^* = (\Pi, U_i^j, \{U_{i_1}^{j_1}, \ldots, U_{i_n}^{j_n}\})$ *be a completed session between* honest *users* $\{U_1, \ldots, U_n\}$*, which is owned by* U_i^j*. Let* $\overline{\mathsf{sid}^*}$ *be a matching session of* sid^**. We say session* sid^* *is* fresh *if none of the following conditions hold:*

1. *The adversary $\mathcal{A}$ issues either of* $\mathsf{SessionReveal}(U_i^j)$ *or* $\mathsf{SessionReveal}(U_{i'}^{j'})$ *for any* $\overline{\mathsf{sid}^*}$ *in the current time frame,*
2. *The adversary $\mathcal{A}$ issues either of* $\mathsf{SessionReveal}(U_i^j)$ *or* $\mathsf{SessionReveal}(U_{i'}^{j'})$ *for any* $\overline{\mathsf{sid}^*}$ *in the past time frame if $\mathcal{A}$ issues either of* $\mathsf{ServerReveal}$, $\mathsf{StaticReveal}(U_i)$ *or* $\mathsf{StaticReveal}(U_{i'})$*,*
3. *The adversary $\mathcal{A}$ issues* $\mathsf{ServerReveal}$ *before completing* sid^**,*
4. *the adversary $\mathcal{A}$ makes either of* $\mathsf{StateReveal}(U_i)$ *or* $\mathsf{StateReveal}(U_{i'})$ *in the current time frame or any of its ancestors*[3]*,*
5. *the adversary $\mathcal{A}$ makes either of* $\mathsf{StaticReveal}(U_i)$ *before completing* sid^* *or in the current time frame, or* $\mathsf{StaticReveal}(U_{i'})$ *before completing* $\overline{\mathsf{sid}^*}$ *for any* $\overline{\mathsf{sid}^*}$ *or in the current time frame,*
6. *the adversary $\mathcal{A}$ makes both of* $\mathsf{StaticReveal}(U_i)$ *and* $\mathsf{EphemeralReveal}(U_i^j)$*, and*
7. *the adversary $\mathcal{A}$ makes both of* $\mathsf{StaticReveal}(U_{i'})$ *and* $\mathsf{EphemeralReveal}(U_{i'}^{j'})$ *for any* $\overline{\mathsf{sid}^*}$.

[3] We say that $state_i$ is an ancestor of $state_{i'}$ if there exists a path $(state_i, \ldots, state_{i'})$ such that each state in the path is updated to the next one.

We note that if both $\mathsf{EphemeralReveal}(U_i^j)$ and $\mathsf{StaticReveal}(U_i)$ are posed, then we regard that $\mathsf{StateReveal}(U_i)$ in the time frame for instance U_i^j is also posed because $state_i$ in the time frame is trivially derived from ESK_i^j and SSK_i.

3.5 Security Experiment

For the security definition, we consider the following security experiment. Initially, the adversary $\mathcal{A}$ is given a set of honest users and makes any sequence of the queries described above. During the experiment, the adversary $\mathcal{A}$ makes the following query.

– $\mathsf{Test}(\mathsf{sid}^*)$: Here, sid^* must be a fresh session. Select random bit $b \in_R \{0, 1\}$, and return the session key held by sid^* if $b = 0$, and return a random key if $b = 1$.

The experiment continues until the adversary $\mathcal{A}$ makes a guess b'. The adversary $\mathcal{A}$ *wins* the game if the test session sid^* is still fresh and if the guess of the adversary $\mathcal{A}$ is correct, i.e., $b' = b$. The advantage of the adversary $\mathcal{A}$ is defined as $\mathbf{Adv}^{\mathrm{dmkd}}(\mathcal{A}) = \Pr[\mathcal{A}\ wins] - \frac{1}{2}$. We define the security as follows.

Definition 10 (DMKD **Security).** *We say that a DMKD protocol* Π *is* secure in the DMKD model *if the following conditions hold:*

1. *If two honest parties complete matching sessions, then, except with negligible probability, they both compute the same session key.*
2. *For any PPT adversary* $\mathcal{A}$*,* $\mathbf{Adv}^{\mathrm{dmkd}}(\mathcal{A})$ *is negligible in security parameter* κ *for the test session* sid^*.

3.6 Summary of Our Security Definition

Here, we give an intuition of security properties captured by our security definition.

– **Ephemeral Key Exposure Resilience.** The adversary can obtain ESKs of users by $\mathsf{EphemeralReveal}$ queries. From the freshness definition, when the adversary does not pose $\mathsf{StaticReveal}(U_i)$ where U_i is the owner of the test session, then the adversary can pose $\mathsf{EphemeralReveal}(U_i^j)$ where the j-th session of U_i is the test session. Thus, it guarantees that the session key is still secure even if $ESKs$ used to generate the session key are *totally exposed.*
– **Time-based Backward Secrecy.** The adversary can obtain the session key of the test session by $\mathsf{SessionReveal}$ queries if the session key was generated at a past time frame. Generally, if the session key of the test session is exposed, the adversary easily distinguish the real session key from a random key. In our security model, we introduce the notion of the time frame, and consider the Update phase. Thus, when a new time frame begins, the session key may be updated. Hence, it guarantees that the updated session key looks independent from past session keys even in the test session.
– **Strong Server Key Forward Secrecy.** The adversary can obtain both SSKs of users and the server by $\mathsf{StaticReveal}$ and $\mathsf{ServerReveal}$ queries. From the freshness definition, when the adversary does not pose $\mathsf{EphemeralReveal}$ queries for the

test session, then the adversary can pose StaticReveal and ServerReveal for users in the test session after completion of the session.[4] Hence, it guarantees that past session keys are still secure even if SSKs of users and the server are exposed. Also, the adversary is allowed to modify messages in the test session (i.e., there is a non-matching session.) regardless of posing StaticReveal or ServerReveal. Thus, while in most of AKE protocols only weak forward secrecy is guaranteed (i.e., the adversary is prohibited to pose StaticReveal for non-matching sessions.), our security guarantees strong forward secrecy.

4 New Dynamic Multi-Cast Key Distribution Protocol under Star Topology

In this section, we show a DMKD protocol under the star topology. In the Dist phase, a group of users shares a session key with the help of the central server. In the Join phase, new users can join the group that previously established the session key with lower costs than executing a Dist phase by the new group members. In the Leave phase, a subset of group users leaves from the group with lower costs than executing a Dist phase by the remaining group members. After establishing the session key, users can update the key at a new time frame. In the Update phase, the server sends information to refresh the session key to users, and users can locally update the key.

For simplicity, we show a simple setting that only one user joins/leaves the group simultaneously. We show the general setting that multiple users can join/leave the group simultaneously in Appendix A.

4.1 Design Principle

The session key in our protocol is generated from two key materials K_1 and K_2. K_1 guarantees ephemeral key exposure resilience and strong server key forward secrecy, and K_2 guarantees time-based backward secrecy. Here, we give an intuition of the design of our protocol.

The way to share K_1 is based on the ring structure, and is similar to the previous dynamic group key exchange protocol (the YT protocol) [YT10]. In the YT protocol, each user broadcasts g^{r_i} in Round 1, and computes $g^{r_{i-1}r_i}$ and $g^{r_i r_{i+1}}$. Then, the left key $K_i^{(l)}$ based on $g^{r_{i-1}r_i}$ and the right key $K_i^{(r)}$ based on $g^{r_i r_{i+1}}$ are generated, and each user broadcasts $K_i^{(l)} \oplus K_i^{(r)}$ in Round 2. Also, a representative user generates k, and broadcasts the masked k with his left key to all users. Then, each user can recover the left and right keys for all group members with his/her $K_i^{(l)}$ and $K_i^{(r)}$. Thus, they can share k, and generate K_1 based on k. However, we cannot simply apply the YT protocol to our protocol. First, the YT protocol is insecure if ESKs of users are exposed; that means, ephemeral key exposure resilience is not satisfied. The other problem is scalability. To broadcast messages and to compute k, both communication and computational complexity of each user depend on the number of users; and thus, if the YT protocol is

[4] If the adversary poses StaticReveal or ServerReveal before completion of the test session, then the session key is trivially distinguished from a random key. Also, it means that the server is honest-but-curious.

used very large system, the load of users increases. Therefore, achieving both exposure resilience and scalability is not easy task.

We can solve the first problem on ephemeral key exposure resilience by using the twisted PRF trick. We use outputs of the twisted PRF based on the SSK and the ESK instead of all randomness of users in our protocol. From Lemma 1, it is guaranteed that an output of the twisted PRF is indistinguishable from the random value unless both SSK and ESK are exposed. The freshness definition also guarantee that both SSK and ESK are not exposed in the test session. Therefore, our protocol satisfies ephemeral key exposure resilience. Also, we can solve the second problem on scalability thanks to the difference of the network topology. In the YT protocol, each user must communicate with other users directly because of the mesh topology, and all costs inevitably depend on the number of users. On the other hand, in our protocol, each user only communicates with the server because of the star topology. Thus, we can confine commutation depending on the number of users to the server in our protocol. The server only sends a constant number of messages to each user. Therefore, communication and computational complexity of each user do not depend on the number of users; and thus, our protocol is scalable. We note that complexity of the server depends on the number of users, however, it is inevitable and not serious because the server has sufficient computational power and communication bandwidth in reality.

The other key material, K_2, is generated by the server. It is encrypted by a CP-ABE scheme with the access structure that the ID is of the recipient and the time is within the current time frame. Since for every time frame each user receives a new decryption key with attribute of his/her ID and the current time, K_2 can be decrypted if the ID of the recipient is valid and the decryption key is sent at the same time frame. The new decryption key is stored as state information. After generating the session key, when a new time frame begins, the server sends the encrypted form of new K_2 and each user locally updates the session key. Though the adversary can pose $\mathsf{StateReveal}$ queries, the freshness definition guarantee that state information of the test session in the current time frame or any of its ancestors is not exposed. Thus, even if the adversary obtains session keys at past time frames, the session key at the current time frame is still secure. Therefore, our protocol satisfies time-based backward secrecy.

4.2 System Setup

S runs the setup algorithm **Setup** of CP-ABE, and generates a public parameter $Params$ and a master secret key msk. Let p be a κ-bit prime, and G be a finite cyclic group of order p with generator g, h. Let $TCR : \{0,1\}^* \to \{0,1\}^\kappa$ be a TCR hash function. Let $tPRF : \{0,1\}^\kappa \times Kspace_\kappa \times \{0,1\}^\kappa \times Kspace_\kappa \to \mathbb{Z}_p$ and $tPRF' : \{0,1\}^\kappa \times Kspace_\kappa \times \{0,1\}^\kappa \times Kspace_\kappa \to Kspace_\kappa$ be twisted PRFs. Let $F : \{0,1\}^\kappa \times G \to \mathbb{Z}_p^2$, $F' : \{0,1\}^\kappa \times \mathbb{Z}_p \to Kspace_\kappa$ and $F'' : \{0,1\}^\kappa \times Kspace_\kappa \to \{0,1\}^\kappa$, $F''' : \{0,1\}^\kappa \times Kspace_\kappa \to \mathbb{Z}_p$ be PRFs. S stores msk as SSK_S, and publishes $(Params, p, G, g, h, TCR, tPRF, tPRF', F, F', F'', F''')$ as SPK_S.

There are N users $U_1 \ldots, U_N$. Each user U_i runs the key generation algorithm of PKE **Gen**, and generates a public key pk_i and a secret key sk_i. Also, U_i generates secret strings for the twisted PRF (st_i, st'_i) and (st_S, st'_S), where $st_i, st_S \in_R \mathsf{Kspace}_\kappa$ and $st'_i, st'_S \in_R \{0,1\}^\kappa$. U_i stores (sk_i, st_i, st'_i) as SSK_i, and publishes pk_i as SPK_i.

4.3 Dist Phase

A set of users $U_{i_1}, \ldots, U_{i_n}$ $(n \leq N)$ starts a new session and share a session key. For simplicity, w.l.o.g., we suppose that $(U_{i_1}, \ldots, U_{i_n}) = (U_1, \ldots, U_n)$.

(State Update at New Time Frame) If the session is the first session for U_i at the time frame TF, then for the current time $time$ S generates $usk_i \leftarrow \mathbf{Der}(Params, msk, A_i)$ with attribute $A_i = (U_i, time)$ and $mk_i \leftarrow \mathbf{MGen}$, and computes $CT_i \leftarrow \mathbf{Enc}_{pk_i}(usk_i, mk_i)$. Then, S sends CT_i to U_i, and U_i obtains $(usk_i, mk_i) \leftarrow \mathbf{Dec}_{sk_i}(CT_i)$ and updates (usk_i, mk_i) in $state_i$.

(Round 1 for Users) U_i generates $\tilde{r}_i \in_R \{0,1\}^\kappa$, $\tilde{r}'_i \in_R \mathsf{Kspace}_\kappa$, $\tilde{k}_i \in_R \{0,1\}^\kappa$, $\tilde{k}'_i \in_R \mathsf{Kspace}_\kappa$, $\tilde{s}_i \in_R \{0,1\}^\kappa$ and $\tilde{s}'_i \in_R \mathsf{Kspace}_\kappa$ as ESK_i, and computes $r_i = tPRF(\tilde{r}_i, \tilde{r}'_i, st_i, st'_i)$, $k_i = tPRF(\tilde{k}_i, \tilde{k}'_i, st_i, st'_i)$ and $s_i = tPRF(\tilde{s}_i, \tilde{s}'_i, st_i, st'_i)$. Then, U_i computes $R_i = g^{r_i}$ and $c_i = g^{k_i} h^{s_i}$, and sends (R_i, c_i) to S.

(Round 1 for Server) On receiving (R_i, c_i) from all users, S computes $sid = TCR(c_1, \ldots, c_n)$ and chooses a representative user from $(U_1, \ldots, U_n)$. Here, w.l.o.g., we suppose that U_1 is the representative user. For $i \in [1, n]$, S sends (sid, R_{i-1}, R_{i+1}) to U_i. Also, S notices that U_1 is the representative user.

(Round 2 for Users) For $i \in [2, n]$, on receiving (sid, R_{i-1}, R_{i+1}), U_i computes $K_i^{(l)} = F(sid, R_{i-1}^{r_i})$, $K_i^{(r)} = F(sid, R_{i+1}^{r_i})$ and $T_i = K_i^{(l)} \oplus K_i^{(r)}$. Then, U_i computes $\sigma_i \leftarrow \mathbf{Tag}_{mk_i}(R_i, c_i, R_{i-1}, R_{i+1}, k_i, s_i, T_i, U_i, sid)$ and sends $(k_i, s_i, T_i, \sigma_i)$ to S.
On receiving (sid, R_n, R_2), U_1 computes $K_1^{(l)} = F(sid, R_n^{r_1})$, $K_1^{(r)} = F(sid, R_2^{r_1})$, $T_1 = K_1^{(l)} \oplus K_1^{(r)}$ and $T' = K_1^{(l)} \oplus (k_1 \| s_1)$. Then, U_1 computes $\sigma_1 \leftarrow \mathbf{Tag}_{mk_1}(R_1, c_1, R_n, R_2, T_1, T', U_1, sid)$ and sends (T_1, T', σ_1) to S.

(Round 2 for Server) On receiving (T_1, T', σ_1) and $(k_i, s_i, T_i, \sigma_i)$, S verifies $\mathbf{Ver}_{mk_1}(R_1, c_1, R_n, R_2, T_1, T', U_1, sid, \sigma_1)$ and $\mathbf{Ver}_{mk_i}(R_i, c_i, R_{i-1}, R_{i+1}, k_i, s_i, T_i, U_i, sid, \sigma_i)$, and if the verification fails, then aborts. Also, for $i \in [2, n]$, S checks if $c_i = g^{k_i} h^{s_i}$ holds, and if the verification fails, then aborts. S generates $\tilde{k}_S \in_R \{0,1\}^\kappa$, $\tilde{k}'_S \in_R \mathsf{Kspace}_\kappa$, $\tilde{K}_1 \in_R \{0,1\}^\kappa$ and $\tilde{K}'_1 \in_R Kspace_\kappa$ as ESK_S, and computes $k_S = tPRF(\tilde{k}_S, \tilde{k}'_S, st_S, st'_S)$, $K_1 = tPRF'(\tilde{K}_1, \tilde{K}'_1, st_S, st'_S)$ and $k' = (\bigoplus_{2 \leq i \leq n} k_i) \oplus k_S$. For $i \in [2, n]$, S computes $T'_i = \bigoplus_{1 \leq j \leq i-1} T_j$. For $i \in [1, n]$, S computes $CT'_i \leftarrow \mathbf{AEnc}(Params, P_i, K_1)$ with access structure $P_i := (ID = U_i) \wedge (time \in TF)$. S computes $\sigma'_1 \leftarrow \mathbf{Tag}_{mk_1}(R_1, c_1, R_n, R_2, T_1, T', U_1, sid, k', CT'_1)$, and sends (k', CT'_1, σ'_1) to U_1. For $i \in [2, n]$, S computes $\sigma'_i \leftarrow \mathbf{Tag}_{mk_i}(R_i, c_i, R_{i-1}, R_{i+1}, k_i, s_i, T_i, U_i, sid, c_1, k', T'_i, T', CT'_i)$, and sends $(c_1, k', T'_i, T', CT'_i, \sigma'_i)$ to U_i.

(Session Key Generation and Post Computation) For $i \in [2, n]$, on receiving $(c_1, k', T'_i, T', CT'_i, \sigma'_i)$, U_i verifies $\mathbf{Ver}_{mk_i}(R_i, c_i, R_{i-1}, R_{i+1}, k_i, s_i, T_i, U_i, sid, c_1, k', T'_i, T', CT'_i, \sigma'_i)$, and if the verification fails, then aborts. U_i computes $K_1^{(l)} = T'_i \oplus K_i^{(l)}$ and $k_1 \| s_1 = T' \oplus K_1^{(l)}$, and checks if $c_1 = g^{k_1} h^{s_1}$ holds, and if the verification fails, then aborts. U_i decrypts $K_1 \leftarrow \mathbf{ADec}_{usk_i}(CT'_i, P_i)$, computes $K_2 = F'(sid, k' \oplus k_1)$, and outputs the session key $SK = F''(sid, K_1) \oplus F''(sid, K_2)$. As state information, U_i adds sid, $H_i^{(l)} = R_{i-1}^{r_i}$, $H_i^{(r)} = R_{i+1}^{r_i}$ and $r = F'''(sid, K_1) \oplus F'''(sid, K_2)$ to $state_i$.
On receiving (k', CT'_1, σ'_1), U_1 verifies $\mathbf{Ver}_{mk_1}(R_1, c_1, R_n, R_2, T_1, T', U_1, sid, k', CT'_1, \sigma'_1)$, and if the verification fails, then aborts. U_1 decrypts $K_1 \leftarrow \mathbf{ADec}_{usk_1}(CT'_1, P_1)$, computes $K_2 = F'(sid, k' \oplus k_1)$, and outputs the session key $SK = F''(sid, K_1) \oplus F''(sid, K_2)$. As state information, U_1 adds sid, $H_1^{(l)} = R_n^{r_1}$, $H_1^{(r)} = R_2^{r_1}$ and $r = F'''(sid, K_1) \oplus F'''(sid, K_2)$ to $state_i$.

4.4 Join Phase

A user $U_{i_{n+1}}$ joins an established session by $U_1, \ldots, U_n$. W.l.o.g., we suppose that $U_{i_{n+1}} = U_{n+1}$.

In the Join phase, users U_i for $i \in [2, n-1]$ can reduce computation than the Dist phase. They do not need to compute g^{r_i}. The ring structure to compute K_1 still works because r in $state_i$ is used to connect the ring instead of using r_i.

(State Update at New Time Frame) If the session is the first session for U_i at the time frame TF', then for the current time $time$ S generates $usk_i \leftarrow \mathbf{Der}(Params, msk, A_i)$ with attribute $A_i = (U_i, time)$ and $mk_i \leftarrow \mathbf{MGen}$, and computes $CT_i \leftarrow \mathbf{Enc}_{pk_i}(usk_i, mk_i)$. Then, S sends CT_i to U_i, and U_i obtains $(usk_i, mk_i) \leftarrow \mathbf{Dec}_{sk_i}(CT_i)$ and updates (usk_i, mk_i) in $state_i$.

(Round 1 for Users) For $i \in \{1, n, n+1\}$, U_i generates $\tilde{r}_i \in_R \{0,1\}^\kappa$, $\tilde{r}'_i \in_R \mathsf{Kspace}_\kappa$, $\tilde{k}_i \in_R \{0,1\}^\kappa$, $\tilde{k}'_i \in_R \mathsf{Kspace}_\kappa$, $\tilde{s}_i \in_R \{0,1\}^\kappa$ and $\tilde{s}'_i \in_R \mathsf{Kspace}_\kappa$ as ESK_i, and computes $r_i = tPRF(\tilde{r}_i, \tilde{r}'_i, st_i, st'_i)$, $k_i = tPRF(\tilde{k}_i, \tilde{k}'_i, st_i, st'_i)$ and $s_i = tPRF(\tilde{s}_i, \tilde{s}'_i, st_i, st'_i)$. U_i computes $R_i = g^{r_i}$ and $c_i = g^{k_i}h^{s_i}$, and sends (R_i, c_i) to S.
For $i \in [2, n-1]$, U_i generates $\tilde{k}_i \in_R \{0,1\}^\kappa$, $\tilde{k}'_i \in_R \mathsf{Kspace}_\kappa$, $\tilde{s}_i \in_R \{0,1\}^\kappa$ and $\tilde{s}'_i \in_R \mathsf{Kspace}_\kappa$ as ESK_i, and computes $k_i = tPRF(\tilde{k}_i, \tilde{k}'_i, st_i, st'_i)$ and $s_i = tPRF(\tilde{s}_i, \tilde{s}'_i, st_i, st'_i)$. U_i computes $c_i = g^{k_i}h^{s_i}$, and sends c_i to S.

(Round 1 for Server) On receiving (R_i, c_i) for $i \in \{1, n, n+1\}$ and c_i for $i \in [2, n-1]$, S computes $sid = TCR(c_1, \ldots, c_{n+k})$, and chooses a representative user from $i \in \{1, n, n+1\}$. Here, w.l.o.g., we suppose that U_1 is the representative user. S sends (sid, R_n, R_1) to U_{n+1}. For $i \in \{1, 2\}$, S sends (sid, R_{i-1}) to U_i where $R_0 = R_{n+1}$. For $i \in [3, n-2]$, S sends sid to U_i. Also, S notices that U_1 is the representative user.

(Round 2 for Users) On receiving (sid, R_{n+1}), U_1 computes $K_1^{(l)} = F(sid, R_{n+1}^{r_1})$, $K_1^{(r)} = F(sid, R_1^r)$, $T_1 = K_1^{(l)} \oplus K_1^{(r)}$ and $T' = K_1^{(l)} \oplus (k_1 \| s_1)$. U_1 computes $\sigma_1 \leftarrow \mathbf{Tag}_{mk_1}(R_1, c_1, R_{n+1}, T_1, T', U_1, sid)$, and sends (T_1, T', σ_1) to S.
On receiving (sid, R_1), U_2 computes $K_2^{(l)} = F(sid, R_1^r)$, $K_2^{(r)} = F(sid, g^r)$ and $T_2 = K_2^{(l)} \oplus K_2^{(r)}$. U_2 computes $\sigma_2 \leftarrow \mathbf{Tag}_{mk_2}(c_2, R_1, k_2, s_2, T_2, U_2, sid)$, and sends $(k_2, s_2, T_2, \sigma_2)$ to S.
For $i \in [3, n-2]$, on receiving sid, U_i computes $\sigma_i \leftarrow \mathbf{Tag}_{mk_i}(c_i, k_i, s_i, U_i, sid)$, and sends (k_i, s_i, σ_i) to S.
On receiving (sid, R_n), U_{n-1} computes $K_{n-1}^{(l)} = F(sid, g^r)$, $K_{n-1}^{(r)} = F(sid, R_n^r)$ and $T_{n-1} = K_{n-1}^{(l)} \oplus K_{n-1}^{(r)}$. U_{n-1} computes $\sigma_{n-1} \leftarrow \mathbf{Tag}_{mk_{n-1}}(c_{n-1}, R_n, k_{n-1}, s_{n-1}, T_{n-1}, U_{n-1}, sid)$, and sends $(k_{n-1}, s_{n-1}, T_{n-1}, \sigma_{n-1})$ to S.
On receiving (sid, R_{n+1}), U_n computes $K_n^{(l)} = F(sid, R_n^r)$, $K_n^{(r)} = F(sid, R_{n+1}^{r_n})$ and $T_n = K_n^{(l)} \oplus K_n^{(r)}$. U_n computes $\sigma_n \leftarrow \mathbf{Tag}_{mk_n}(R_n, c_n, R_{n+1}, k_n, s_n, T_n, U_n, sid)$, and sends $(k_n, s_n, T_n, \sigma_n)$ to S.
On receiving (sid, R_n, R_1), U_{n+1} computes $K_{n+1}^{(l)} = F(sid, R_n^{r_{n+1}})$, $K_{n+1}^{(r)} = F(sid, R_1^{r_{n+1}})$ and $T_{n+1} = K_{n+1}^{(l)} \oplus K_{n+1}^{(r)}$. U_{n+1} computes $\sigma_{n+1} \leftarrow \mathbf{Tag}_{mk_{n+1}}(R_{n+1}, c_{n+1}, R_n, R_1, k_{n+1}, s_{n+1}, T_{n+1}, U_{n+1}, sid)$, and sends $(k_{n+1}, s_{n+1}, T_{n+1}, \sigma_{n+1})$ to S.

(Round 2 for Server) On receiving (T_1, T', σ_1) from U_1, $(k_i, s_i, T_i, \sigma_i)$ for $i \in \{2\} \cup [n-1, n+1]$ and (k_i, s_i, σ_i) for $i \in [3, n-2]$, S verifies authentication-tags, and if the verification fails, then aborts. Also, for $i \in [2, n+1]$, S checks if $c_i = g^{k_i}h^{s_i}$ holds, and if the verification fails, then aborts. S generates $\tilde{k}_S \in_R \{0,1\}^\kappa$, $\tilde{k}'_S \in_R \mathsf{Kspace}_\kappa$, $\tilde{K}_1 \in_R$

$\{0,1\}^\kappa$ and $\tilde{K}'_1 \in_R Kspace_\kappa$ as ESK_S, and computes $k_S = tPRF(\tilde{k}_S, \tilde{k}'_S, st_S, st'_S)$, $K_1 = tPRF'(\tilde{K}_1, \tilde{K}'_1, st_S, st'_S)$ and $k' = (\bigoplus_{2 \le i \le n+k} k_i) \oplus k_S$. For $i \in [2, n+1]$, S computes $T'_i = \bigoplus_{1 \le j \le i-1} T_j$, where for $i \in [3, n-1]$, T_i is treated as empty (i.e., $T'_3 = \cdots = T'_{n-1}$). For $i \in [1, n+1]$, S computes $CT'_i \leftarrow \mathbf{AEnc}(Params, P_i, K_1)$ with access structure $P_i := (ID = U_i) \wedge (time \in TF)$.

S computes $\sigma'_1 \leftarrow \mathbf{Tag}_{mk_1}(R_1, c_1, R_{n+1}, T_1, T', U_1, sid, k', CT'_1)$, and sends (k', CT'_1, σ'_1) to U_1.

S computes $\sigma'_2 \leftarrow \mathbf{Tag}_{mk_2}(c_2, R_1, k_2, s_2, T_2, U_2, sid, c_1, k', T'_2, T', CT'_2)$, and sends $(c_1, k', T'_2, T', CT'_2, \sigma'_2)$ to U_2.

For $i \in [3, n-2]$, S computes $\sigma'_i \leftarrow \mathbf{Tag}_{mk_i}(c_i, k_i, s_i, U_i, sid, c_1, k', T'_i, T', CT'_i)$, and sends $(c_1, k', T'_i, T', CT'_i, \sigma'_i)$ to U_i.

S computes $\sigma'_{n-1} \leftarrow \mathbf{Tag}_{mk_{n-1}}(c_{n-1}, R_n, k_{n-1}, s_{n-1}, T_{n-1}, U_{n-1}, sid, c_1, k', T'_{n-1}, T', CT'_{n-1})$, and sends $(c_1, k', T'_{n-1}, T', CT'_{n-1}, \sigma'_{n-1})$to U_{n-1}.

S computes $\sigma'_n \leftarrow \mathbf{Tag}_{mk_n}(R_n, c_n, R_{n+1}, k_n, s_n, T_n, U_n, sid, c_1, k', T'_n, T', CT'_n)$, and sends $(c_1, k', T'_n, T', CT'_n, \sigma'_n)$ to U_n.

S computes $\sigma'_{n+1} \leftarrow \mathbf{Tag}_{mk_{n+1}}(R_{n+1}, c_{n+1}, R_n, R_1, k_{n+1}, s_{n+1}, T_{n+1}, U_{n+1}, sid, c_1, k', T'_{n+1}, T', CT'_{n+1})$, and sends $(c_1, k', T'_{n+1}, T', CT'_{n+1}, \sigma'_{n+1})$ to U_{n+1}.

(Session Key Generation and Post Computation) For $i \in [2, n+1]$, on receiving $(c_1, k', T'_i, T', CT'_i, \sigma'_i)$, U_i verifies the authentication-tag, and if the verification fails, then aborts. U_i computes $K_1^{(l)} = T'_i \oplus K_i^{(l)}$ where for $i \in [3, n-1]$ $K_1^{(l)} = T'_i \oplus g^r$ and $k_1 \| s_1 = T' \oplus K_1^{(l)}$, and checks if $c_1 = g^{k_1} h^{s_1}$ holds, and if the verification fails, then aborts. U_i decrypts $K_1 \leftarrow \mathbf{ADec}_{usk_i}(CT'_i, P_i)$, computes $K_2 = F'(sid, k' \oplus k_1)$, and outputs the session key $SK = F''(sid, K_1) \oplus F''(sid, K_2)$. As state information, U_i updates $r = F'''(sid, K_1) \oplus F'''(sid, K_2)$ in $state_i$. Also, U_n updates $H_n^{(r)} = R_{n+1}^{r_n}$ in $state_n$. U_{n+1} adds sid, $H_{n+1}^{(l)} = R_n^{r_{n+1}}$ and $H_{n+1}^{(r)} = R_1^{r_{n+1}}$ to $state_{n+1}$.

On receiving (k', CT'_1, σ'_1), U_1 verifies the authentication-tag, and if the verification fails, then aborts. U_1 decrypts $K_1 \leftarrow \mathbf{ADec}_{usk_1}(CT'_1, P_1)$, computes $K_2 = F'(sid, k' \oplus k_1)$, and outputs the session key $SK = F''(sid, K_1) \oplus F''(sid, K_2)$. As state information, U_1 updates sid, $H_1^{(l)} = R_{n+1}^{r_1}$ and $r = F'''(sid, K_1) \oplus F'''(sid, K_2)$ in $state_1$.

4.5 Leave Phase

A user U_j leaves an established session by $U_1, \ldots, U_n$.

In the Leave phase, users $U_i \in \mathcal{I} \setminus \{U_{j-1}, U_j, U_{j_1+1}\}$ can reduce computation than the Dist phase. They do not need to compute g^{r_i}. The ring structure to compute K_1 still works because $H_i^{(l)}$ and $H_i^{(r)}$ in $state_i$ are used to connect the ring instead of using $g^{r_{i-1} r_i}$ and $g^{r_i r_{i+1}}$.

(State Update at New Time Frame) If the session is the first session for U_i at the time frame TF', then for the current time $time$ S generates $usk_i \leftarrow \mathbf{Der}(Params, msk, A_i)$ with attribute $A_i = (U_i, time)$ and $mk_i \leftarrow \mathbf{MGen}$, and computes $CT_i \leftarrow \mathbf{Enc}_{pk_i}(usk_i, mk_i)$. Then, S sends CT_i to U_i, and U_i obtains $(usk_i, mk_i) \leftarrow \mathbf{Dec}_{sk_i}(CT_i)$ and updates (usk_i, mk_i) in $state_i$.

(Round 1 for Users) $U_i \in \{U_{j-1}, U_{j+1}\}$ generates $\tilde{r}_i \in_R \{0,1\}^\kappa$, $\tilde{r}'_i \in_R \mathsf{Kspace}_\kappa$, $\tilde{k}_i \in_R \{0,1\}^\kappa$, $\tilde{k}'_i \in_R \mathsf{Kspace}_\kappa$, $\tilde{s}_i \in_R \{0,1\}^\kappa$ and $\tilde{s}'_i \in_R \mathsf{Kspace}_\kappa$ as ESK_i, and computes $r_i = tPRF(\tilde{r}_i, \tilde{r}'_i, st_i, st'_i)$, $k_i = tPRF(\tilde{k}_i, \tilde{k}'_i, st_i, st'_i)$ and $s_i = tPRF(\tilde{s}_i, \tilde{s}'_i, st_i, st'_i)$. Then, U_i computes $R_i = g^{r_i}$ and $c_i = g^{k_i}h^{s_i}$, and sends (R_i, c_i) to S.

$U_i \in \mathcal{I} \setminus \{U_{j-1}, U_j, U_{j+1}\}$ generates $\tilde{k}_i \in_R \{0,1\}^\kappa$, $\tilde{k}'_i \in_R \mathsf{Kspace}_\kappa$, $\tilde{s}_i \in_R \{0,1\}^\kappa$ and $\tilde{s}'_i \in_R \mathsf{Kspace}_\kappa$ as ESK_i, and computes $k_i = tPRF(\tilde{k}_i, \tilde{k}'_i, st_i, st'_i)$ and $s_i = tPRF(\tilde{s}_i, \tilde{s}'_i, st_i, st'_i)$. Then, U_i computes $c_i = g^{k_i}h^{s_i}$, and sends c_i to S.

(Round 1 for Server) On receiving (R_i, c_i) from $U_i \in \{U_{j-1}, U_{j+1}\}$ and c_i from $U_i \in \mathcal{I} \setminus \{U_{j-1}, U_j, U_{j+1}\}$, for i such that $U_i \in \mathcal{I} \setminus \{U_j\}$, S computes $sid = TCR(\{c_i\}_{\mathcal{I}\setminus\{U_j\}})$, chooses a representative user from $U_i \in \{U_{j-1}, U_{j+1}\}$. Here, w.l.o.g., we suppose that U_{j-1} is the representative user. S sends (sid, R_{j+1}) to U_{j-1}. S sends (sid, R_{j-1}) to U_{j+1}. Then, S sends sid to $U_i \in \mathcal{I} \setminus \{U_{j-1}, U_j, U_{j+1}\}$. Also, S notices that U_{j-1} is the representative user.

(Round 2 for Users) On receiving (sid, R_{j+1}), U_{j-1} computes $K^{(l)}_{j-1} = F(sid, H^{(l)}_{j-1})$, $K^{(r)}_{j-1} = F(sid, R^{r_{j-1}}_{j+1})$, $T_{j-1} = K^{(l)}_{j-1} \oplus K^{(r)}_{j-1}$ and $T' = K^{(l)}_{j-1} \oplus (k_{j-1} \| s_{j-1})$. U_{j-1} computes $\sigma_{j-1} \leftarrow \mathbf{Tag}_{mk_{j-1}}(R_{j-1}, c_{j-1}, R_{j+1}, T_{j-1}, T', U_{j-1}, sid)$, and sends $(T_{j-1}, T', \sigma_{j-1})$ to S.

On receiving (sid, R_{j-1}), U_{j+1} computes $K^{(l)}_{j+1} = F(sid, R^{r_{j+1}}_{j-1})$, $K^{(r)}_{j+1} = F(sid, H^{(r)}_{j+1})$ and $T_{j+1} = K^{(l)}_{j+1} \oplus K^{(r)}_{j+1}$. U_{j+1} computes $\sigma_{j+1} \leftarrow \mathbf{Tag}_{mk_{j+1}}(R_{j+1}, c_{j+1}, R_{j-1}, k_{j+1}, s_{j+1}, T_{j+1}, U_{j+1}, sid)$, and sends $(k_{j+1}, s_{j+1}, T_{j+1}, \sigma_{j+1})$ to S.

On receiving sid, $U_i \in \mathcal{I} \setminus \{U_{j-1}, U_j, U_{j+1}\}$ computes $K^{(l)}_i = F(sid, H^{(l)}_i)$, $K^{(r)}_i = F(sid, H^{(r)}_i)$ and $T_i = K^{(l)}_i \oplus K^{(r)}_i$. U_i computes $\sigma_i \leftarrow \mathbf{Tag}_{mk_i}(c_i, k_i, s_i, T_i, U_i, sid)$, and sends $(k_i, s_i, T_i, \sigma_i)$ to S.

(Round 2 for Server) On receiving $(T_{j-1}, T', \sigma_{j-1})$ from U_{j-1} and $(k_i, s_i, T_i, \sigma_i)$ from other users, S verifies the authentication-tag, and if the verification fails, then aborts. Also, for $U_i \in \mathcal{I} \setminus \{U_{j-1}, U_j\}$, S checks if $c_i = g^{k_i}h^{s_i}$ holds, and if the verification fails, then aborts. S generates $\tilde{k}_S \in_R \{0,1\}^\kappa$, $\tilde{k}'_S \in_R \mathsf{Kspace}_\kappa$, $\tilde{K}_1 \in_R \{0,1\}^\kappa$ and $\tilde{K}'_1 \in_R Kspace_\kappa$ as ESK_S, and computes $k_S = tPRF(\tilde{k}_S, \tilde{k}'_S, st_S, st'_S)$ and $K_1 = tPRF'(\tilde{K}_1, \tilde{K}'_1, st_S, st'_S)$. For i such that $U_i \in \mathcal{I} \setminus \{U_{j-1}, U_j\}$, S computes $k' = (\bigoplus\{k_i\}) \oplus k_S$. For i such that $U_i \in \mathcal{I} \setminus \{U_j\}$ and $i < j-1$, S computes $T'_i = \bigoplus_{1 \le \ell \le i-1, j-1 \le \ell \le n} T_\ell$, where T_j is empty. For i such that $U_i \in \mathcal{I} \setminus \{U_j\}$ and $j+1 \le i$, S computes $T'_i = \bigoplus_{j-1 \le \ell \le i-1} T_\ell$, where T_j is empty. For $U_i \in \mathcal{I} \setminus \{U_j\}$, S computes $CT'_i \leftarrow \mathbf{AEnc}(Params, P_i, K_1)$ with access structure $P_i := (ID = U_i) \wedge (time \in TF)$.

S computes $\sigma'_{j-1} \leftarrow \mathbf{Tag}_{mk_{j-1}}(R_{j-1}, c_{j-1}, R_{j+1}, T_{j-1}, T', U_{j-1}, sid, k', CT'_{j-1})$, and sends $(k', CT'_{j-1}, \sigma'_{j-1})$ to U_{j-1}.

S computes $\sigma'_{j+1} \leftarrow \mathbf{Tag}_{mk_{j+1}}(R_{j+1}, c_{j+1}, R_{j-1}, k_{j+1}, s_{j+1}, T_{j+1}, U_{j+1}, sid, c_{j-1}, k', T'_{j+1}, T', CT'_{j+1})$, and sends $(c_{j-1}, k', T'_{j+1}, T', CT'_{j+1}, \sigma'_{j+1})$ to U_{j+1}.

For $U_i \in \mathcal{I} \setminus \{U_{j-1}, U_j, U_{j+1}\}$, S computes $\sigma'_i \leftarrow \mathbf{Tag}_{mk_i}(c_i, k_i, s_i, T_i, U_i, sid, c_{j-1}, k', T'_i, T', CT'_i)$, and sends $(c_{j-1}, k', T'_i, T', CT'_i, \sigma'_i)$ to U_i.

(Session Key Generation and Post Computation) On receiving $(c_{j-1}, k', T'_i, T', CT'_i, \sigma'_i)$, $U_i \in \mathcal{I} \setminus \{U_{j-1}, U_j\}$ verifies the authentication-tag, and if the verification fails, then aborts. U_i computes $K^{(l)}_{j-1} = T'_i \oplus K^{(l)}_i$ and $k_{j-1} \| s_{j-1} = T' \oplus K^{(l)}_{j-1}$, and checks if $c_{j-1} = g^{k_{j-1}}h^{s_{j-1}}$hold, and if the verification fails, then aborts. U_i decrypts $K_1 \leftarrow \mathbf{ADec}_{usk_i}(CT'_i, P_i)$, computes $K_2 = F'(sid, k' \oplus k_{j-1})$, and outputs the session key

$SK = F''(sid, K_1) \oplus F''(sid, K_2)$. As state information, U_i updates sid, $r = F'''(sid, K_1) \oplus F'''(sid, K_2)$ in $state_i$.

On receiving $(k', CT'_{j-1}, \sigma'_{j-1})$, U_{j-1} verifies the authentication-tag, and if the verification fails, then aborts. U_{j-1} decrypts $K_1 \leftarrow \mathbf{ADec}_{usk_{j-1}}(CT'_{j-1}, P_{j-1})$, computes $K_2 = F'(sid, k' \oplus k_{j-1})$, and outputs the session key $SK = F''(sid, K_1) \oplus F''(sid, K_2)$. As state information, U_1 updates sid, $r = F'''(sid, K_1) \oplus F'''(sid, K_2)$ in $state_1$. Additionally, U_{j-1} updates $H^{(r)}_{j-1} = R^{r_{j-1}}_{j+1}$ in $state_{j-1}$, and U_{j+1} updates $H^{(l)}_{j+1} = R^{r_{j+1}}_{j-1}$ in $state_{j+1}$.

4.6 Update Phase

When a new time frame begins, a set of users $U_{i_1}, \ldots, U_{i_n}$ $(n \leq N)$ updates the session key SK shared by them in the Dist/Join/Leave phase at the past time frame to a new session key SK'. For simplicity, w.l.o.g., we suppose that $(U_{i_1}, \ldots, U_{i_n}) = (U_1, \ldots, U_n)$.

(State Update at New Time Frame) If the session is the first session for U_i at the time frame TF, then for the current time $time$ S generates $usk_i \leftarrow \mathbf{Der}(Params, msk, A_i)$ with attribute $A_i = (U_i, time)$ and $mk_i \leftarrow \mathbf{MGen}$, computes $CT_i \leftarrow \mathbf{Enc}_{pk_i}(usk_i, mk_i)$. Then, S sends CT_i to U_i, and U_i obtains $(usk_i, mk_i) \leftarrow \mathbf{Dec}_{sk_i}(CT_i)$ and updates (usk_i, mk_i) in $state_i$.

(Information for Update) S generates $\tilde{K}_1 \in_R \{0,1\}^\kappa$ and $\tilde{K}'_1 \in_R Kspace_\kappa$, and computes $K_1 = tPRF'(\tilde{K}_1, \tilde{K}'_1, st_S, st'_S)$ and $CT'_i \leftarrow \mathbf{AEnc}(Params, P_i, K_1)$ with access structure $P_i := (ID = U_i) \wedge (time \in TF)$. Then, S sends CT'_i to U_i.

(Session Key Update) On receiving CT'_i, U_i decrypts $K_1 \leftarrow \mathbf{ADec}_{usk_i}(CT'_i, P_i)$, and outputs the updated session key $SK' = F''(sid, K_1) \oplus SK$.

5 Complexity for Users

5.1 Computational Complexity

We consider dominant operations like modular exponentiations and operations for public key crypto, and ignore other light-weight operations like XORs and operations for secret key crypto.

In the Dist phase, on-line computations (i.e., from Round 1 to post computations) for a user are g^{r_i} and $g^{k_i}h^{s_i}$ for Round 1, $R^{r_i}_{i-1}$ and $R^{r_i}_{i+1}$ for Round 2, and $g^{k_1}h^{s_1}$ and the decryption of CT'_i for the session key generation. In the Join phase, maximum on-line computations for a user are g^{r_i} and $g^{k_i}h^{s_i}$ for Round 1, $R^{r_i}_{i-1}$ and $R^{r_i}_{i+1}$ for Round 2, and $g^{k_1}h^{s_1}$ and the decryption of CT'_i for the session key generation. In the Leave phase, maximum on-line computations for a user are g^{r_i} and $g^{k_i}h^{s_i}$ for Round 1, $R^{r_i}_{i-1}$ for Round 2, and $g^{k_1}h^{s_1}$ and the decryption of CT'_i for the session key generation. In the Update phase, the on-line computation for a user is the decryption of CT'_i for the session key update.

Therefore, for all phases, computational complexity of users is constant for the number of users.

5.2 Communication Complexity

In the Dist phase, sent and received information for a user in on-line (i.e., from Round 1 to post computations) are (R_i, c_i) and (sid, R_{i-1}, R_{i+1}) for Round 1, and $(k_i, s_i, T_i, \sigma_i)$ and $(c_1,\ k', T'_i, T', CT'_i, \sigma'_i)$ for Round 2. In the Join phase, maximum sent and received information for a user in on-line are (R_i, c_i) and (sid, R_{i-1}, R_{i+1}) for Round 1, and $(k_i, s_i, T_i, \sigma_i)$ and $(c_1, k', T'_i, T', CT'_i, \sigma'_i)$ for Round 2. In the Leave phase, maximum sent and received information for a user in on-line are (R_i, c_i) and (sid, R_{i-1}) for Round 1, and $(k_i, s_i, T_i, \sigma_i)$ and $(c_{j-1}, k', T'_{j+1}, T', CT'_{j+1}, \sigma'_{j+1})$ for Round 2. In the Leave phase, received information for a user in on-line is CT'_i for the session key update.

Therefore, for all phases, communication complexity of users is constant for the number of users.

6 Security

Theorem 1. *We assume that TCR satisfies the TCR property, tPRF and tPRF′ are twisted PRFs, F, F′, F″ and F‴ are PRFs,* (**Gen**, **Enc**, **Dec**) *is a CCA-secure PKE,* (**Setup**, **Der**, **AEnc**, **ADec**) *is a selective CCA-secure CP-ABE,* (**MGen**, **Tag**, **Ver**) *is an UF-CMA MAC scheme and the DDH assumption in G holds. Then, our scheme is secure in the* DMKD *model.*

Here, we show a proof sketch. The proof can be divided four cases: (1) the test session is in the Dist phase, (2) the test session is in the Join phase, (3) the test session is in the Leave phase, and (4) the test session is in the Update phase. For Case (1), (2) and (3), secrecy of the session key is guaranteed by secrecy of K_1. Thus, we use the game hopping proof technique [Sho04], and, finally, K_1 is replaced with a random value. To prevent malicious behaviours of the adversary, we show that the probability that messages in the test session are modified is negligible thanks to the security of PKE and MAC, and the DDH assumption. For Case (4), secrecy of the session key is guaranteed by secrecy of K_2. Thus, K_2 is replaced with a random value similar to other cases. In this case, we rely on the security of CP-ABE to prevent malicious behaviours of the adversary.

Proof. We prove Theorem 1 for all four cases as follows.

6.1 Case of Dist Phase

We change the interface of oracle queries and the computation of the session key. These instances are gradually changed over hybrid experiments, depending on specific sub-cases. In the last hybrid experiment, the session key in the test session does not contain information of the bit b. Thus, the adversary clearly only output a random guess. We denote these hybrid experiments by $\mathbf{H}_0, \ldots, \mathbf{H}_7$, and the advantage of the adversary $\mathcal{A}$ when participating in experiment $\mathbf{H}_i$ by $\mathbf{Adv}(\mathcal{A}, \mathbf{H}_i)$.

Hybrid experiment $\mathbf{H}_0$: This experiment denotes the real experiment for DMKD security and in this experiment the environment for $\mathcal{A}$ is as defined in the protocol. Thus,

$\mathbf{Adv}(\mathcal{A}, \mathbf{H}_0)$ is the same as the advantage of the real experiment.

Hybrid experiment $\mathbf{H}_1$: If *sid* in two sessions are identical, the experiment halts.

When randomness in generating c_i are identical or the TCR property of H is broken, *sid* in two sessions may be identical. However, such an event occurs with negligible probability. Thus, $|\mathbf{Adv}(\mathcal{A}, \mathbf{H}_1) - \mathbf{Adv}(\mathcal{A}, \mathbf{H}_0)|$ is negligible.

Hybrid experiment $\mathbf{H}_2$: The experiment selects user instances $\{U_{i_1}^{j_1}, \ldots, U_{i_n}^{j_n}\}$) and the owner U_i^j randomly in advance. If $\mathcal{A}$ poses Test query to a session except $\{U_{i_1}^{j_1}, \ldots, U_{i_n}^{j_n}\}$) owned by U_i^j, the experiment halts.

Since guess of the test session matches with $\mathcal{A}$'s choice with probability $1/\ell_{max}$ where ℓ_{max} is the total number of sessions in the system, $\mathbf{Adv}(\mathcal{A}, \mathbf{H}_2) \geq (1/\ell_{max}) \cdot \mathbf{Adv}(\mathcal{A}, \mathbf{H}_1)$.

Hybrid experiment $\mathbf{H}_3$: If $\mathcal{A}$ modifies a message in the test session and the session is complete, the experiment halts. We denote such an event Bad.

Bad may occur only if either of the following events occur:

Bad_1 : $\mathcal{A}$ obtains information of mk_i from CT_i.
Bad_2 : $\mathcal{A}$ forges σ_i for a modified message.
Bad_3 : $\mathcal{A}$ forges σ'_i for a modified message.
Bad_4 : $\mathcal{A}$ finds (k'_i, s'_i) such that $c_i = g^{k_i} h^{s_i} = g^{k'_i} h^{s'_i}$ and $(k'_i, s'_i) \neq (k_i, s_i)$.

Since $\mathbf{H}_2$ does not differ from $\mathbf{H}_3$ if Bad does not occur, from the Difference Lemma [Sho04] $|\mathbf{Adv}(\mathcal{A}, \mathbf{H}_3) - \mathbf{Adv}(\mathcal{A}, \mathbf{H}_2)| \leq \Pr[\mathsf{Bad}]$. We show $\Pr[\mathsf{Bad}]$ is negligible.

First, we show that if Bad_1 occurs, we can construct an adversary B breaking the CCA-security of (**Gen**, **Enc**, **Dec**). B receives the public key pk^* and sets pk^* to SPK_i. When $\mathcal{A}$ poses $\mathsf{Send}(U_i^{j'}, CT_i)$ for sessions other than the test session, B poses CT_i to $\mathcal{DO}$ and continues to simulate the experiment with received (usk_i, mk_i). When $\mathcal{A}$ poses $\mathsf{Send}(S, init)$ for the test session, B chooses mk_0 and mk_1 and obtains the challenge ciphertext CT^* encrypting (usk, mk_0) or (usk, mk_1). Then, B sends CT^* as CT_i to $\mathcal{A}$. After that, if $\mathcal{A}$ poses Send containing a MAC tag of a modified message with $mk_{b'}$, then B outputs $b = b'$. Therefore, if $\mathcal{A}$ obtains information of mk_i from CT_i, B can break the CCA-security.

Next, we show that if Bad_2 or Bad_3 occurs, we can construct a forger B breaking unforgeability of (**MGen**, **Tag**, **Ver**). When $\mathcal{A}$ poses $\mathsf{Send}(U_i^j, (sid, R_{i-1}, R_{i+1}))$ or $\mathsf{Send}(U_i^j, (k_i, s_i, T_i, \sigma_i))$ for the test session, B poses messages to $\mathcal{MO}$ and continues to simulate the experiment with the received MAC tag. It means that mk_i is set as the challenge MAC key. When $\mathcal{A}$ poses Send containing a forged MAC tag σ^* for mk_i, then B outputs σ^*. Therefore, if $\mathcal{A}$ forges σ_i or σ'_i for a modified message, B can break UF-CMA.

Finally, we show that if Bad_4 occurs, we can construct a distinguisher B breaking the DDH assumption. B receives the tuple (g, A, B, C) and sets $g = g, h = A$ to SPK_S. When $\mathcal{A}$ poses $\mathsf{Send}(U_i^j, init)$ for the test session, B chooses (k_i, s_i), computes $c_i = g^{k_i} h^{s_i}$ and returns (R_i, c_i) to $\mathcal{A}$. After that, if $\mathcal{A}$ poses $\mathsf{Send}(S, (k'_i, s'_i, T_i, \sigma_i))$ such that $c_i = g^{k'_i} h^{s'_i}$ and $(k'_i, s'_i) \neq (k_i, s_i)$, then B computes $a = \frac{k'_i - k_i}{s'_i - s_i}$. B verifies if $B^a = C$, and if so, outputs

1, otherwise outputs 0. Therefore, if $\mathcal{A}$ finds (k'_i, s'_i) such that $c_i = g^{k_i}h^{s_i} = g^{k'_i}h^{s'_i}$ and $(k'_i, s'_i) \neq (k_i, s_i)$, B can break the DDH assumption.

Hence, $\Pr[\mathsf{Bad}]$ is negligible, and $|\mathbf{Adv}(\mathcal{A}, \mathbf{H}_3) - \mathbf{Adv}(\mathcal{A}, \mathbf{H}_2)|$ is negligible.

Hybrid experiment $\mathbf{H}_4$: The computation of (r_i, k_i, s_i, k_S) for all users in the test session is changed. Instead of computing tPRF, it is changed as choosing (r_i, k_i, s_i, k_S) randomly.

From the freshness definition (Definition 9) $\mathcal{A}$ cannot pose both of $\mathsf{StaticReveal}(U_i)$ and $\mathsf{EphemeralReveal}(U_i^j)$, and both of $\mathsf{StaticReveal}(U_{i'})$ and $\mathsf{EphemeralReveal}(U_{i'}^{j'})$ for any $\overline{\mathsf{sid}^*}$. Hence, $\mathcal{A}$ cannot see either of $(\tilde{r}_i, \tilde{r}'_i)$ or (st_i, st'_i). From Lemma 1 $r_i = tPRF(\tilde{r}_i, \tilde{r}'_i, st_i, st'_i)$ is indistinguishable from randomly chosen r_i. Similarly, $k_i = tPRF(\tilde{k}_i, \tilde{k}'_i, st_i, st'_i)$, $k_S = tPRF(\tilde{k}_S, \tilde{k}'_S, st_S, st'_S)$ and $s_i = tPRF(\tilde{s}_i, \tilde{s}'_i, st_i, st'_i)$ are indistinguishable from randomly chosen k_i, k_S and s_i, respectively.

Therefore, $|\mathbf{Adv}(\mathcal{A}, \mathbf{H}_4) - \mathbf{Adv}(\mathcal{A}, \mathbf{H}_3)|$ is negligible.

Hybrid experiment $\mathbf{H}_5$: The computation of $R_{i-1}^{r_i}$ and $R_{i+1}^{r_i}$ in the test session is changed. Instead of computing exponentiations, it is changed as choosing R and R' randomly.

In this experiment, r_{i-1}, r_i and r_{i+1} are randomly chosen, and $\mathcal{A}$ cannot see r_{i-1}, r_i and r_{i+1}. From the definition of DDH assumption (Definition 8) $R = R_{i-1}^{r_i}$ is indistinguishable from randomly chosen R. Similarly, $R' = R_{i+1}^{r_i}$ is indistinguishable from randomly chosen R'.

Therefore, $|\mathbf{Adv}(\mathcal{A}, \mathbf{H}_5) - \mathbf{Adv}(\mathcal{A}, \mathbf{H}_4)|$ is negligible.

Hybrid experiment $\mathbf{H}_6$: The computation of $K_i^{(l)}$ and $K_i^{(r)}$ in the test session is changed. Instead of computing PRF, it is changed as choosing $K_i^{(l)}$ and $K_i^{(r)}$ randomly.

In this experiment, $R_{i-1}^{r_i}$ and $R_{i+1}^{r_i}$ are randomly chosen. From the definition of PRF (Definition 1) $K_i^{(l)} = F(sid, R_{i-1}^{r_i})$ is indistinguishable from randomly chosen $K_i^{(l)}$. Similarly, $K_i^{(r)} = F(sid, R_{i+1}^{r_i})$ is indistinguishable from randomly chosen $K_i^{(r)}$.

Therefore, $|\mathbf{Adv}(\mathcal{A}, \mathbf{H}_6) - \mathbf{Adv}(\mathcal{A}, \mathbf{H}_5)|$ is negligible.

Hybrid experiment $\mathbf{H}_7$: The computation of K_2 in the test session is changed. Instead of computing PRF, it is changed as choosing K_2 randomly.

In this experiment, $k' = (\bigoplus_{2 \leq i \leq n} k_i) \oplus k_S$ is random because k_i and k_S are randomly chosen. Also, k_1 is randomly chosen, and thus; $k' \oplus k_1$ is random. From the definition of PRF 1 $K_2 = F'(sid, k' \oplus k_1)$ is indistinguishable from randomly chosen K_2.

Therefore, $|\mathbf{Adv}(\mathcal{A}, \mathbf{H}_7) - \mathbf{Adv}(\mathcal{A}, \mathbf{H}_6)|$ is negligible.

Hybrid experiment $\mathbf{H}_8$: The computation of $K'_2 = F''(sid, K_2)$ in the test session is changed. Instead of computing PRF, it is changed as choosing K'_2 randomly.

In this experiment, K_2 is random. From the definition of PRF 1 $K'_2 = F''(sid, K_2)$ is indistinguishable from randomly chosen K'_2.

Therefore, $|\mathbf{Adv}(\mathcal{A}, \mathbf{H}_8) - \mathbf{Adv}(\mathcal{A}, \mathbf{H}_7)|$ is negligible.

Bounding the advantage in $\mathbf{H}_8$: In $\mathbf{H}_8$, $F''(sid, K_2)$ is replaced with random K'_2.

Hence, $SK = F''(sid, K_1) \oplus K'_2$ is also random. Then, regardless of the challenge bit b for the test session, when $\mathcal{A}$ poses Test query, a random session key is returned.

Therefore, $\mathbf{Adv}(\mathcal{A}, \mathbf{H}_8)$ is negligible.

6.2 Case of Join Phase

The proof is almost the same as the case of the Dist phase. The difference is to add a hybrid experiment between $\mathbf{H}_4$ and $\mathbf{H}_5$. In this experiment, the computation of r in the test session is changed. Instead of computing PRF, it is changed as choosing r randomly.

From the freshness definition (Definition 9) $\mathcal{A}$ cannot pose either of StateReveal(U_i) or StateReveal$(U_{i'})$ in the current time frame or any of its ancestors. Also, though $\mathcal{A}$ can pose SessionReveal$(U_{i'}^{j'})$ for sessions other than the test session, $r = F'''(sid, K_1) \oplus F'''(sid, K_2)$ is independent to $SK = F''(sid, K_1) \oplus F''(sid, K_2)$ from the definition of PRF. Hence, $\mathcal{A}$ cannot see r. Therefore, the difference between this experiment and the previous experiment is negligible.

6.3 Case of Leave Phase

The proof is almost the same as the case of the Dist phase. The difference is to add a hybrid experiment between $\mathbf{H}_5$ and $\mathbf{H}_6$. In this experiment, the computation of $H_i^{(r)}$ and $H_i^{(l)}$ in the test session is changed. Instead of computing exponentiations, it is changed as choosing H and H' randomly.

From the freshness definition (Definition 9) $\mathcal{A}$ cannot pose either of StateReveal(U_i) or StateReveal$(U_{i'})$ in the current time frame or any of its ancestors. Also, since if both EphemeralReveal(U_i^j) and StaticReveal(U_i) are posed, then we regard that StateReveal(U_i) in the time frame for instance U_i^j is also posed, the adversary cannot see r_{i-1}, r_i and r_{i+1} corresponding to $H_i^{(r)}$ and $H_i^{(l)}$ from Lemma 1. From the definition of DDH assumption (Definition 8) $H_i^{(l)} = R_{i-1}^{r_i}$ is indistinguishable from randomly chosen H'. Similarly, $H_i^{(r)}$ is indistinguishable from randomly chosen H. Therefore, the difference between this experiment and the previous experiment is negligible.

6.4 Case of Update Phase

We denote these hybrid experiments by $\mathbf{H}_0, \ldots, \mathbf{H}_4$, and the advantage of the adversary $\mathcal{A}$ when participating in experiment $\mathbf{H}_i$ by $\mathbf{Adv}(\mathcal{A}, \mathbf{H}_i)$.

Hybrid experiment $\mathbf{H}_0$: This experiment denotes the real experiment for DMKD security and in this experiment the environment for $\mathcal{A}$ is as defined in the protocol. Thus, $\mathbf{Adv}(\mathcal{A}, \mathbf{H}_0)$ is the same as the advantage of the real experiment.

Hybrid experiment $\mathbf{H}_1$: If sid in two sessions are identical, the experiment halts.

As the case of the Join phase, $|\mathbf{Adv}(\mathcal{A}, \mathbf{H}_1) - \mathbf{Adv}(\mathcal{A}, \mathbf{H}_0)|$ is negligible.

Hybrid experiment $\mathbf{H}_2$: The experiment selects user instances $\{U_{i_1}^{j_1}, \ldots, U_{i_n}^{j_n}\})$ and the owner U_i^j randomly in advance. If $\mathcal{A}$ poses Test query to a session except $\{U_{i_1}^{j_1}, \ldots, U_{i_n}^{j_n}\}$) owned by U_i^j, the experiment halts.

As the case of the Join phase, $\mathbf{Adv}(\mathcal{A}, \mathbf{H}_2) \geq (1/\ell_{max}) \cdot \mathbf{Adv}(\mathcal{A}, \mathbf{H}_1)$.

Hybrid experiment $\mathbf{H}_3$: The computation of CT_i' in the test session is changed. Instead of encrypting K_1, it is changed as encrypting randomly chosen K_1'.

From the freshness definition (Definition 9) $\mathcal{A}$ can pose SessionReveal(U_i^j) or SessionReveal($U_{i'}^{j'}$) for any $\overline{\mathsf{sid}^*}$ in the past time frame, but cannot pose StaticReveal(U_i), StaticReveal($U_{i'}$) for any $\overline{\mathsf{sid}^*}$, ServerReveal, StateReveal(U_i) and StateReveal($U_{i'}$) in the current time frame. Hence, $\mathcal{A}$ cannot see usk_i or $usk_{i'}$. We show that if $\mathcal{A}$ distinguishes $\mathbf{H}_3$ from $\mathbf{H}_2$, we can construct an adversary B breaking the selective CCA-security of (**Setup**, **Der**, **AEnc**, **ADec**). First, B outputs $P^* := (ID = U_i) \wedge (time \in TF)$ for the test session. Then, B receives the public parameter $Params$ and sets $Params$ to SPK_S. When $\mathcal{A}$ poses Send($U_{i'}^{j'}, CT_{i'}'$) for $U_{i'} \neq U_i$, B poses $A_{i'}$ to $\mathcal{EO}$ where $A_{i'} := (U_{i'}, time)$ and continues to simulate the experiment with received $usk_{i'}$. When $\mathcal{A}$ poses Send($U_i^{j'}, CT_i'$) for sessions other than the test session, B poses CT_i' to $\mathcal{DO}$ and continues to simulate the experiment with received K_1. When $\mathcal{A}$ poses Send($S, init$) for the test session, B computes $K_0^* = tPRF'(\tilde{K}_1, \tilde{K}_1', st_S, st_S')$ and randomly chosen K_1^* and obtains the challenge ciphertext CT^* encrypting K_0^* or K_1^*. Then, B sends CT^* as CT_i' to $\mathcal{A}$. After that, if $\mathcal{A}$ outputs b', then B outputs $b = b'$. If $b = 0$ holds, the environment for $\mathcal{A}$ is identical to $\mathbf{H}_2$, and otherwise, the environment for $\mathcal{A}$ is identical to $\mathbf{H}_3$. Therefore, if $\mathcal{A}$ distinguishes $\mathbf{H}_3$ from $\mathbf{H}_2$, B can break the selective CCA-security.

Hence, $|\mathbf{Adv}(\mathcal{A}, \mathbf{H}_3) - \mathbf{Adv}(\mathcal{A}, \mathbf{H}_2)|$ is negligible.

Hybrid experiment $\mathbf{H}_4$: The computation of $K_1' = F''(sid, K_1)$ in the test session is changed. Instead of computing PRF, it is changed as choosing K_1' randomly.

In this experiment, K_1 is random. From the definition of PRF (Definition 1) $K_1' = F''(sid, K_1)$ is indistinguishable from randomly chosen K_1'.

Therefore, $|\mathbf{Adv}(\mathcal{A}, \mathbf{H}_4) - \mathbf{Adv}(\mathcal{A}, \mathbf{H}_3)|$ is negligible.

Bounding the advantage in $\mathbf{H}_4$: In $\mathbf{H}_4$, $F''(sid, K_1)$ is replaced with random K_1'. Though $\mathcal{A}$ can obtain SK in the past time frame, $SK' = K_1' \oplus SK$ is random. Then, regardless of the challenge bit b for the test session, when $\mathcal{A}$ poses Test query, a random session key is returned.

Therefore, $\mathbf{Adv}(\mathcal{A}, \mathbf{H}_4)$ is negligible.

References

[BBJ+15] Adam Bergkvist, Daniel C. Burnett, Cullen Jennings, Anant Narayanan, and Bernard Aboba. WebRTC 1.0: Real-time Communication Between Browsers. In *InfoQ:*, 2015.

[BCP01] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Provably authenticated group diffie-hellman key exchange - the dynamic case. In Colin Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia,* *December 9-13, 2001, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*, pages 290–309. Springer, 2001.

[BCP02] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Dynamic group diffie-hellman key exchange under standard assumptions. In Lars R. Knudsen, editor, *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*, volume 2332 of *Lecture Notes in Computer Science*, pages 321–336. Springer, 2002.

[BCPQ01] Emmanuel Bresson, Olivier Chevassut, David Pointcheval, and Jean-Jacques Quisquater. Provably authenticated group diffie-hellman key exchange. In Michael K. Reiter and Pierangela Samarati, editors, *CCS 2001, Proceedings of the 8th ACM Conference on Computer and Communications Security, Philadelphia, Pennsylvania, USA, November 6-8, 2001.*, pages 255–264. ACM, 2001.

[BLN$^+$12] Robin Berjon, Travis Leithead, Erika Doyle Navara, Edward O'Connor, and Silvia Pfeiffer. HTML5. In *W3C Working Draft:*, 2012.

[CGI$^+$99] Ran Canetti, Juan A. Garay, Gene Itkis, Daniele Micciancio, Moni Naor, and Benny Pinkas. Multicast security: A taxonomy and some efficient constructions. In *Proceedings IEEE INFOCOM '99, The Conference on Computer Communications, Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies, The Future Is Now, New York, NY, USA, March 21-25, 1999*, pages 708–716. IEEE, 1999.

[Che15] James Chesters. Mozilla Blocks Flash, Encourages HTML5 Adoption. In *InfoQ:*, 2015.

[CWSP98] Germano Caronni, Marcel Waldvogel, Dan Sun, and Bernhard Plattner. Efficient security for large and dynamic multicast groups. In *7th Workshop on Enabling Technologies (WETICE '98), Infrastructure for Collaborative Enterprises, June 17-19, 1998, Palo Alto, CAUSA, Proceedings*, pages 376–383. IEEE Computer Society, 1998.

[DB05] Ratna Dutta and Rana Barua. Constant round dynamic group key agreement. In Jianying Zhou, Javier Lopez, Robert H. Deng, and Feng Bao, editors, *Information Security, 8th International Conference, ISC 2005, Singapore, September 20-23, 2005, Proceedings*, volume 3650 of *Lecture Notes in Computer Science*, pages 74–88. Springer, 2005.

[FSXY15] Atsushi Fujioka, Koutarou Suzuki, Keita Xagawa, and Kazuki Yoneyama. Strongly secure authenticated key exchange from factoring, codes, and lattices. *Des. Codes Cryptography*, 76(3):469–504, 2015.

[FTR10] Jason Fischl, Hannes Tschofenig, and Eric Rescorla. Framework for Establishing a Secure Real-time Transport Protocol (SRTP), Security Context Using Datagram Transport Layer Security (DTLS). In *IEFT RFC 5763:*, 2010.

[KF14] Kaoru Kurosawa and Jun Furukawa. 2-pass key exchange protocols from cpa-secure KEM. In Josh Benaloh, editor, *Topics in Cryptology - CT-RSA 2014 - The Cryptographer's Track at the RSA Conference 2014, San Francisco, CA, USA, February 25-28, 2014. Proceedings*, volume 8366 of *Lecture Notes in Computer Science*, pages 385–401. Springer, 2014.

[KLL04] Hyun-Jeong Kim, Su-Mi Lee, and Dong Hoon Lee. Constant-round authenticated group key exchange for dynamic groups. In Pil Joong Lee, editor, *Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5-9, 2004, Proceedings*, volume 3329 of *Lecture Notes in Computer Science*, pages 245–259. Springer, 2004.

[LTW10] Iuon-Chang Lin, Shih-Shan Tang, and Chung-Ming Wang. Multicast key management without rekeying processes. *Comput. J.*, 53(7):939–950, 2010.

[Mar15] Jack Marshall. Google Chrome Will Begin Blocking Flash Web Ads. In *The Wall Street Journal:*, 2015.

[MK14] Neha Mittal and Vinod Kumar. An Efficient and Secure Multicast Key Management Scheme based on Star Topology. In *International Journal of Computer Science and Information Technologies, Vol.5 (3):*, pages 3777–3783, 2014.

[MP04] Daniele Micciancio and Saurabh Panjwani. Optimal communication complexity of generic multicast key distribution. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, volume 3027 of *Lecture Notes in Computer Science*, pages 153–170. Springer, 2004.

[MSU09] Mark Manulis, Koutarou Suzuki, and Berkant Ustaoglu. Modeling leakage of ephemeral secrets in tripartite/group key exchange. In Dong Hoon Lee and Seokhie Hong, editors, *Information, Security and Cryptology - ICISC 2009, 12th International Conference, Seoul, Korea, December 2-4, 2009, Revised Selected Papers*, volume 5984 of *Lecture Notes in Computer Science*, pages 16–33. Springer, 2009.

[Res15] Eric Rescorla. WebRTC Security Architecture, draft-ietf-rtcweb-security-arch-11. In *IETF Draft:*, 2015.

[SCFJ03] Henning Schulzrinne, Stephen L. Casner, Ron Frederick, and Van Jacobson. RTP: A Transport Protocol for Real-Time Applications. In *IEFT RFC 3550:*, 2003.

[SHC+15] Hung-Min Sun, Bing-Zhe He, Chien-Ming Chen, Tsu-Yang Wu, Chia-Hsien Lin, and Huaxiong Wang. A provable authenticated group key agreement protocol for mobile environment. *Inf. Sci.*, 321:224–237, 2015.

[Sho04] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. In *Cryptology ePrint Archive: 2004/332*, 2004.

[SM03] Alan T. Sherman and David A. McGrew. Key establishment in large dynamic groups using one-way function trees. *IEEE Trans. Software Eng.*, 29(5):444–458, 2003.

[SP12] K. Saravanan and T. Purusothaman. Efficient Star Topology based Multicast Key Management Algorithm. In *Journal of Computer Science 8 (6):*, pages 951–956, 2012.

[SY13] Koutarou Suzuki and Kazuki Yoneyama. Exposure-resilient one-round tripartite key exchange without random oracles. In Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *Applied Cryptography and Network Security - 11th International Conference, ACNS 2013, Banff, AB, Canada, June 25-28, 2013. Proceedings*, volume 7954 of *Lecture Notes in Computer Science*, pages 458–474. Springer, 2013.

[SY14] Koutarou Suzuki and Kazuki Yoneyama. Exposure-resilient one-round tripartite key exchange without random oracles. *IEICE Transactions*, 97-A(6):1345–1355, 2014.

[WCS+99] Marcel Waldvogel, Germano Caronni, Dan Sun, Nathalie Weiler, and Bernhard Plattner. The versakey framework: versatile group key management. *IEEE Journal on Selected Areas in Communications*, 17(9):1614–1631, 1999.

[WW15] Magnus Westerlund and Stephan Wenger. RTP Topologies, draft-ietf-avtcore-rtp-topologies-update-07. In *IETF Draft:*, 2015.

[YT10] Guomin Yang and Chik How Tan. Dynamic group key exchange revisited. In Swee-Huay Heng, Rebecca N. Wright, and Bok-Min Goi, editors, *Cryptology and Network Security - 9th International Conference, CANS 2010, Kuala Lumpur, Malaysia, December 12-14, 2010. Proceedings*, volume 6467 of *Lecture Notes in Computer Science*, pages 261–277. Springer, 2010.

[YYK+16] Kazuki Yoneyama, Reo Yoshida, Yuto Kawahara, Tetsutaro Kobayashi, Hitoshi Fuji, and Tomohide Yamamoto. Multi-Cast Key Distribution: Scalable, Dynamic and Provably Secure Construction. In *ProvSec 2016*, 2016. http://eprint.iacr.org/2016/833.

A General Setting of Our Protocol

A.1 System Setup

The system setup is the same as the simple case.

A.2 Dist Phase

A set of users $U_{i_1}, \ldots, U_{i_n}$ ($n \leq N$) starts a new session and share a session key. For simplicity, w.l.o.g., we suppose that $(U_{i_1}, \ldots, U_{i_n}) = (U_1, \ldots, U_n)$.

(State Update at New Time Frame) If the session is the first session for U_i at the time frame TF, then for the current time $time$ S generates $usk_i \leftarrow \mathbf{Der}(Params, msk, A_i)$ with attribute $A_i = (U_i, time)$ and $mk_i \leftarrow \mathbf{MGen}$, and computes $CT_i \leftarrow \mathbf{Enc}_{pk_i}(usk_i, mk_i)$. Then, S sends CT_i to U_i, and U_i obtains $(usk_i, mk_i) \leftarrow \mathbf{Dec}_{sk_i}(CT_i)$ and updates (usk_i, mk_i) in $state_i$.

(Round 1 for Users) U_i generates $\tilde{r}_i \in_R \{0,1\}^\kappa$, $\tilde{r}'_i \in_R \mathsf{Kspace}_\kappa$, $\tilde{k}_i \in_R \{0,1\}^\kappa$, $\tilde{k}'_i \in_R \mathsf{Kspace}_\kappa$, $\tilde{s}_i \in_R \{0,1\}^\kappa$ and $\tilde{s}'_i \in_R \mathsf{Kspace}_\kappa$ as ESK_i, and computes $r_i = tPRF(\tilde{r}_i, \tilde{r}'_i, st_i, st'_i)$, $k_i = tPRF(\tilde{k}_i, \tilde{k}'_i, st_i, st'_i)$ and $s_i = tPRF(\tilde{s}_i, \tilde{s}'_i, st_i, st'_i)$. Then, U_i computes $R_i = g^{r_i}$ and $c_i = g^{k_i} h^{s_i}$, and sends (R_i, c_i) to S.

(Round 1 for Server) On receiving (R_i, c_i) from all users, S computes $sid = TCR(c_1, \ldots, c_n)$ and chooses a representative user from $(U_1, \ldots, U_n)$. Here, w.l.o.g., we suppose that U_1 is the representative user. For $i \in [1, n]$, S sends (sid, R_{i-1}, R_{i+1}) to U_i. Also, S notices that U_1 is the representative user.

(Round 2 for Users) For $i \in [2, n]$, on receiving (sid, R_{i-1}, R_{i+1}), U_i computes $K_i^{(l)} = F(sid, R_{i-1}^{r_i})$, $K_i^{(r)} = F(sid, R_{i+1}^{r_i})$ and $T_i = K_i^{(l)} \oplus K_i^{(r)}$. Then, U_i computes $\sigma_i \leftarrow \mathbf{Tag}_{mk_i}(R_i, c_i, R_{i-1}, R_{i+1}, k_i, s_i, T_i, U_i, sid)$ and sends $(k_i, s_i, T_i, \sigma_i)$ to S.

On receiving (sid, R_n, R_2), U_1 computes $K_1^{(l)} = F(sid, R_n^{r_1})$, $K_1^{(r)} = F(sid, R_2^{r_1})$, $T_1 = K_1^{(l)} \oplus K_1^{(r)}$ and $T' = K_1^{(l)} \oplus (k_1 \| s_1)$. Then, U_1 computes $\sigma_1 \leftarrow \mathbf{Tag}_{mk_1}(R_1, c_1, R_n, R_2, T_1, T', U_1, sid)$ and sends (T_1, T', σ_1) to S.

(Round 2 for Server) On receiving (T_1, T', σ_1) and $(k_i, s_i, T_i, \sigma_i)$, S verifies $\mathbf{Ver}_{mk_1}(R_1, c_1, R_n, R_2, T_1, T', U_1, sid, \sigma_1)$ and $\mathbf{Ver}_{mk_i}(R_i, c_i, R_{i-1}, R_{i+1}, k_i, s_i, T_i, U_i, sid, \sigma_i)$, and if the verification fails, then aborts. Also, for $i \in [2, n]$, S checks if $c_i = g^{k_i} h^{s_i}$ holds, and if the verification fails, then aborts. S generates $\tilde{k}_S \in_R \{0,1\}^\kappa$, $\tilde{k}'_S \in_R \mathsf{Kspace}_\kappa$, $\tilde{K}_1 \in_R \{0,1\}^\kappa$ and $\tilde{K}'_1 \in_R Kspace_\kappa$ as ESK_S, and computes $k_S = tPRF(\tilde{k}_S, \tilde{k}'_S, st_S, st'_S)$, $K_1 = tPRF'(\tilde{K}_1, \tilde{K}'_1, st_S, st'_S)$ and $k' = (\bigoplus_{2 \leq i \leq n} k_i) \oplus k_S$. For $i \in [2, n]$, S computes $T'_i = \bigoplus_{1 \leq j \leq i-1} T_j$. For $i \in [1, n]$, S computes $CT'_i \leftarrow \mathbf{AEnc}(Params, P_i, K_1)$ with access structure $P_i := (ID = U_i) \wedge (time \in TF)$. S computes $\sigma'_1 \leftarrow \mathbf{Tag}_{mk_1}(R_1, c_1, R_n, R_2, T_1, T', U_1, sid, k', CT'_1)$, and sends (k', CT'_1, σ'_1) to U_1. For $i \in [2, n]$, S computes $\sigma'_i \leftarrow \mathbf{Tag}_{mk_i}(R_i, c_i, R_{i-1}, R_{i+1}, k_i, s_i, T_i, U_i, sid, c_1, k', T'_i, T', CT'_i)$, and sends $(c_1, k', T'_i, T', CT'_i, \sigma'_i)$ to U_i.

(Session Key Generation and Post Computation) For $i \in [2, n]$, on receiving $(c_1, k', T'_i, T', CT'_i, \sigma'_i)$, U_i verifies $\mathbf{Ver}_{mk_i}(R_i, c_i, R_{i-1}, R_{i+1}, k_i, s_i, T_i, U_i, sid, c_1, k', T'_i, T', CT'_i, \sigma'_i)$, and if the verification fails, then aborts. U_i computes $K_1^{(l)} = T'_i \oplus K_i^{(l)}$ and $k_1 \| s_1 = T' \oplus K_1^{(l)}$, and checks if $c_1 = g^{k_1} h^{s_1}$ holds, and if the verification fails, then aborts. U_i decrypts $K_1 \leftarrow \mathbf{ADec}_{usk_i}(CT'_i, P_i)$, computes $K_2 = F'(sid, k' \oplus k_1)$,

and outputs the session key $SK = F''(sid, K_1) \oplus F''(sid, K_2)$. As state information, U_i adds sid, $H_1^{(l)} = R_n^{r_1}$, $H_1^{(r)} = R_2^{r_1}$ and $r = F'''(sid, K_1) \oplus F'''(sid, K_2)$ to $state_i$.
On receiving (k', CT'_1, σ'_1), U_1 verifies $\mathbf{Ver}_{mk_1}(R_1, c_1, R_n, R_2, T_1, T', U_1, sid, k', CT'_1, \sigma'_1)$, and if the verification fails, then aborts. U_1 decrypts $K_1 \leftarrow \mathbf{ADec}_{usk_1}(CT'_1, P_1)$, computes $K_2 = F'(sid, k' \oplus k_1)$, and outputs the session key $SK = F''(sid, K_1) \oplus F''(sid, K_2)$. As state information, U_1 adds sid, $H_i^{(l)} = R_{i-1}^{r_i}$, $H_i^{(r)} = R_{i+1}^{r_i}$ and $r = F'''(sid, K_1) \oplus F'''(sid, K_2)$ to $state_i$.

A.3 Join Phase

A new set of users $U_{i_{n+1}}, \ldots, U_{i_{n+k}}$ join an established session by $U_1, \ldots, U_n$. W.l.o.g., we suppose that $(U_{i_{n+1}}, \ldots, U_{i_{n+k}}) = (U_{n+1}, \ldots, U_{n+k})$.

In the Join phase, users U_i for $i \in [2, n-1]$ can reduce computation than the Dist phase. They do not need to compute g^{r_i}. The ring structure to compute K_1 still works because r in $state_i$ is used to connect the ring instead of using r_i.

(State Update at New Time Frame) If the session is the first session for U_i at the time frame TF', then for the current time $time$ S generates $usk_i \leftarrow \mathbf{Der}(Params, msk, A_i)$ with attribute $A_i = (U_i, time)$ and $mk_i \leftarrow \mathbf{MGen}$, and computes $CT_i \leftarrow \mathbf{Enc}_{pk_i}(usk_i, mk_i)$. Then, S sends CT_i to U_i, and U_i obtains $(usk_i, mk_i) \leftarrow \mathbf{Dec}_{sk_i}(CT_i)$ and updates (usk_i, mk_i) in $state_i$.

(Round 1 for Users) For $i \in \{1\} \cup [n, n+k]$, U_i generates $\tilde{r}_i \in_R \{0,1\}^\kappa$, $\tilde{r}'_i \in_R \mathsf{Kspace}_\kappa$, $\tilde{k}_i \in_R \{0,1\}^\kappa$, $\tilde{k}'_i \in_R \mathsf{Kspace}_\kappa$, $\tilde{s}_i \in_R \{0,1\}^\kappa$ and $\tilde{s}'_i \in_R \mathsf{Kspace}_\kappa$ as ESK_i, and computes $r_i = tPRF(\tilde{r}_i, \tilde{r}'_i, st_i, st'_i)$, $k_i = tPRF(\tilde{k}_i, \tilde{k}'_i, st_i, st'_i)$ and $s_i = tPRF(\tilde{s}_i, \tilde{s}'_i, st_i, st'_i)$. U_i computes $R_i = g^{r_i}$ and $c_i = g^{k_i} h^{s_i}$, and sends (R_i, c_i) to S.
For $i \in [2, n-1]$, U_i generates $\tilde{k}_i \in_R \{0,1\}^\kappa$, $\tilde{k}'_i \in_R \mathsf{Kspace}_\kappa$, $\tilde{s}_i \in_R \{0,1\}^\kappa$ and $\tilde{s}'_i \in_R \mathsf{Kspace}_\kappa$ as ESK_i, and computes $k_i = tPRF(\tilde{k}_i, \tilde{k}'_i, st_i, st'_i)$ and $s_i = tPRF(\tilde{s}_i, \tilde{s}'_i, st_i, st'_i)$. U_i computes $c_i = g^{k_i} h^{s_i}$, and sends c_i to S.

(Round 1 for Server) On receiving (R_i, c_i) for $i \in \{1\} \cup [n, n+k]$ and c_i for $i \in [2, n-1]$, S computes $sid = TCR(c_1, \ldots, c_{n+k})$, and chooses a representative user from $i \in \{1\} \cup [n, n+k]$. Here, w.l.o.g., we suppose that U_1 is the representative user. For $i \in [n+1, n+k]$, S sends (sid, R_{i-1}, R_{i+1}) to U_i where $R_{n+k+1} = R_1$. For $i \in \{1, 2\}$, S sends (sid, R_{i-1}) to U_i where $R_0 = R_{n+k}$. For $i \in [3, n-2]$, S sends sid to U_i. Also, S notices that U_1 is the representative user.

(Round 2 for Users) On receiving (sid, R_{n+k}), U_1 computes $K_1^{(l)} = F(sid, R_{n+k}^{r_1})$, $K_1^{(r)} = F(sid, g^{r_1 r})$, $T_1 = K_1^{(l)} \oplus K_1^{(r)}$ and $T' = K_1^{(l)} \oplus (k_1 \| s_1)$. U_1 computes $\sigma_1 \leftarrow \mathbf{Tag}_{mk_1}(R_1, c_1, R_{n+k}, T_1, T', U_1, sid)$, and sends (T_1, T', σ_1) to S.
On receiving (sid, R_1), U_2 computes $K_2^{(l)} = F(sid, R_1^r)$, $K_2^{(r)} = F(sid, g^r)$ and $T_2 = K_2^{(l)} \oplus K_2^{(r)}$. U_2 computes $\sigma_2 \leftarrow \mathbf{Tag}_{mk_2}(c_2, R_1, k_2, s_2, T_2, U_2, sid)$, and sends $(k_2, s_2, T_2, \sigma_2)$to S.
For $i \in [3, n-2]$, on receiving sid, U_i computes $\sigma_i \leftarrow \mathbf{Tag}_{mk_i}(c_i, k_i, s_i, U_i, sid)$, and sends (k_i, s_i, σ_i) to S.
On receiving (sid, R_n), U_{n-1} computes $K_{n-1}^{(l)} = F(sid, g^r)$, $K_{n-1}^{(r)} = F(sid, R_n^r)$ and $T_{n-1} = K_{n-1}^{(l)} \oplus K_{n-1}^{(r)}$. U_{n-1} computes $\sigma_{n-1} \leftarrow \mathbf{Tag}_{mk_{n-1}}(c_{n-1}, R_n, k_{n-1}, s_{n-1}, T_{n-1}, U_{n-1}, sid)$, and sends $(k_{n-1}, s_{n-1}, T_{n-1}, \sigma_{n-1})$ to S.

On receiving (sid, R_{n+1}), U_n computes $K_n^{(l)} = F(sid, R_n^{r})$, $K_n^{(r)} = F(sid, R_{n+1}^{r_n})$ and $T_n = K_n^{(l)} \oplus K_n^{(r)}$. U_n computes $\sigma_n \leftarrow \mathbf{Tag}_{mk_n}(R_n, c_n, R_{n+1}, k_n, s_n, T_n, U_n, sid)$, and sends $(k_n, s_n, T_n, \sigma_n)$ to S.

For $i \in [n+1, n+k]$, on receiving (sid, R_{i-1}, R_{i+1}), U_i computes $K_i^{(l)} = F(sid, R_{i-1}^{r_i})$, $K_i^{(r)} = F(sid, R_{i+1}^{r_i})$ and $T_i = K_i^{(l)} \oplus K_i^{(r)}$. U_i computes $\sigma_i \leftarrow \mathbf{Tag}_{mk_i}(R_i, c_i, R_{i-1}, R_{i+1}, k_i, s_i, T_i, U_i, sid)$, and sends $(k_i, s_i, T_i, \sigma_i)$ to S.

(Round 2 for Server) On receiving (T_1, T', σ_1) from U_1, $(k_i, s_i, T_i, \sigma_i)$ for $i \in \{2\} \cup [n-1, n+k]$ and (k_i, s_i, σ_i) for $i \in [3, n-2]$, S verifies authentication-tags, and if the verification fails, then aborts. Also, for $i \in [2, n+k]$, S checks if $c_i = g^{k_i} h^{s_i}$ holds, and if the verification fails, then aborts. S generates $\tilde{k}_S \in_R \{0,1\}^\kappa$, $\tilde{k}'_S \in_R \mathsf{Kspace}_\kappa$, $\tilde{K}_1 \in_R \{0,1\}^\kappa$ and $\tilde{K}'_1 \in_R Kspace_\kappa$ as ESK_S, and computes $k_S = tPRF(\tilde{k}_S, \tilde{k}'_S, st_S, st'_S)$, $K_1 = tPRF'(\tilde{K}_1, \tilde{K}'_1, st_S, st'_S)$ and $k' = (\bigoplus_{2 \le i \le n+k} k_i) \oplus k_S$. For $i \in [2, n+k]$, S computes $T'_i = \bigoplus_{1 \le j \le i-1} T_j$, where for $i \in [3, n-1]$, T_i is treated as empty (i.e., $T'_3 = \cdots = T'_{n-1}$). For $i \in [1, n+k]$, S computes $CT'_i \leftarrow \mathbf{AEnc}(Params, P_i, K_1)$ with access structure $P_i := (ID = U_i) \wedge (time \in TF)$.

S computes $\sigma'_1 \leftarrow \mathbf{Tag}_{mk_1}(R_1, c_1, R_{n+k}, T_1, T', U_1, sid, k', CT'_1)$, and sends (k', CT'_1, σ'_1) to U_1.

S computes $\sigma'_2 \leftarrow \mathbf{Tag}_{mk_2}(c_2, R_1, k_2, s_2, T_2, U_2, sid, c_1, k', T'_2, T', CT'_2)$, and sends $(c_1, k', T'_2, T', CT'_2, \sigma'_2)$ to U_2.

For $i \in [3, n-2]$, S computes $\sigma'_i \leftarrow \mathbf{Tag}_{mk_i}(c_i, k_i, s_i, U_i, sid, c_1, k', T'_i, T', CT'_i)$, and sends $(c_1, k', T'_i, T', CT'_i, \sigma'_i)$ to U_i.

S computes $\sigma'_{n-1} \leftarrow \mathbf{Tag}_{mk_{n-1}}(c_{n-1}, R_n, k_{n-1}, s_{n-1}, T_{n-1}, U_{n-1}, sid, c_1, k', T'_{n-1}, T', CT'_{n-1})$, and sends $(c_1, k', T'_{n-1}, T', CT'_{n-1}, \sigma'_{n-1})$to U_{n-1}.

S computes $\sigma'_n \leftarrow \mathbf{Tag}_{mk_n}(R_n, c_n, R_{n+1}, k_n, s_n, T_n, U_n, sid, c_1, k', T'_n, T', CT'_n)$, and sends $(c_1, k', T'_n, T', CT'_n, \sigma'_n)$ to U_n.

For $i \in [n+1, n+k]$, S computes $\sigma'_i \leftarrow \mathbf{Tag}_{mk_i}(R_i, c_i, R_{i-1}, R_{i+1}, k_i, s_i, T_i, U_i, sid, c_1, k', T'_i, T', CT'_i)$, and sends $(c_1, k', T'_i, T', CT'_i, \sigma'_i)$ to U_i.

(Session Key Generation and Post Computation) For $i \in [2, n+k]$, on receiving $(c_1, k', T'_i, T', CT'_i, \sigma'_i)$, U_i verifies the authentication-tag, and if the verification fails, then aborts. U_i computes $K_1^{(l)} = T'_i \oplus K_i^{(l)}$ where for $i \in [3, n-1]$ $K_1^{(l)} = T'_i \oplus g^r$ and $k_1 \| s_1 = T' \oplus K_1^{(l)}$, and checks if $c_1 = g^{k_1} h^{s_1}$ holds, and if the verification fails, then aborts. U_i decrypts $K_1 \leftarrow \mathbf{ADec}_{usk_i}(CT'_i, P_i)$, computes $K_2 = F'(sid, k' \oplus k_1)$, and outputs the session key $SK = F''(sid, K_1) \oplus F''(sid, K_2)$. As state information, U_i updates $r = F'''(sid, K_1) \oplus F'''(sid, K_2)$ in $state_i$. Also, U_n updates $H_n^{(r)} = R_{n+1}^{r_n}$ in $state_n$. For $i \in [n+1, n+k]$, U_i adds sid, $H_i^{(l)} = R_{i-1}^{r_i}$ and $H_i^{(r)} = R_{i+1}^{r_i}$ to $state_i$.

On receiving (k', CT'_1, σ'_1), U_1 verifies the authentication-tag, and if the verification fails, then aborts. U_1 decrypts $K_1 \leftarrow \mathbf{ADec}_{usk_1}(CT'_1, P_1)$, computes $K_2 = F'(sid, k' \oplus k_1)$, and outputs the session key $SK = F''(sid, K_1) \oplus F''(sid, K_2)$. As state information, U_1 updates sid, $H_1^{(l)} = R_{n+k}^{r_1}$ and $r = F'''(sid, K_1) \oplus F'''(sid, K_2)$ in $state_1$.

A.4 Leave Phase

A set of users $\mathcal{R} = (U_{j_1}, \ldots, U_{j_m})$ leave an established session by $U_1, \ldots, U_n$. Let $\mathcal{N} = (U_{j_1 - 1}, U_{j_1 + 1}, U_{j_2 - 1}, U_{j_2 + 1}, \ldots, U_{j_m - 1}, U_{j_m + 1})$ be a set of users neighbouring leaved users. W.l.o.g., we suppose that $U_1 \in \mathcal{N}$.

In the Leave phase, users $U_i \in \mathcal{I} \setminus (\mathcal{R} \cup \mathcal{N})$ can reduce computation than the Dist phase. They do not need to compute g^{r_i}. The ring structure to compute K_1 still works because $H_i^{(l)}$ and $H_i^{(r)}$ in $state_i$ are used to connect the ring instead of using $g^{r_{i-1}r_i}$ and $g^{r_i r_{i+1}}$.

(State Update at New Time Frame) If the session is the first session for U_i at the time frame TF', then for the current time $time$ S generates $usk_i \leftarrow \mathbf{Der}(Params, msk, A_i)$ with attribute $A_i = (U_i, time)$ and $mk_i \leftarrow \mathbf{MGen}$, and computes $CT_i \leftarrow \mathbf{Enc}_{pk_i}(usk_i, mk_i)$. Then, S sends CT_i to U_i, and U_i obtains $(usk_i, mk_i) \leftarrow \mathbf{Dec}_{sk_i}(CT_i)$ and updates (usk_i, mk_i) in $state_i$.

(Round 1 for Users) $U_i \in \mathcal{N}$ generates $\tilde{r}_i \in_R \{0,1\}^\kappa$, $\tilde{r}'_i \in_R$ Kspace$_\kappa$, $\tilde{k}_i \in_R \{0,1\}^\kappa$, $\tilde{k}'_i \in_R$ Kspace$_\kappa$, $\tilde{s}_i \in_R \{0,1\}^\kappa$ and $\tilde{s}'_i \in_R$ Kspace$_\kappa$ as ESK_i, and computes $r_i = tPRF(\tilde{r}_i, \tilde{r}'_i, st_i, st'_i)$, $k_i = tPRF(\tilde{k}_i, \tilde{k}'_i, st_i, st'_i)$ and $s_i = tPRF(\tilde{s}_i, \tilde{s}'_i, st_i, st'_i)$. Then, U_i computes $R_i = g^{r_i}$ and $c_i = g^{k_i} h^{s_i}$, and sends (R_i, c_i) to S.
$U_i \in \mathcal{I} \setminus (\mathcal{R} \cup \mathcal{N})$ generates $\tilde{k}_i \in_R \{0,1\}^\kappa$, $\tilde{k}'_i \in_R$ Kspace$_\kappa$, $\tilde{s}_i \in_R \{0,1\}^\kappa$ and $\tilde{s}'_i \in_R$ Kspace$_\kappa$ as ESK_i, and computes $k_i = tPRF(\tilde{k}_i, \tilde{k}'_i, st_i, st'_i)$ and $s_i = tPRF(\tilde{s}_i, \tilde{s}'_i, st_i, st'_i)$. Then, U_i computes $c_i = g^{k_i} h^{s_i}$, and sends c_i to S.

(Round 1 for Server) On receiving (R_i, c_i) from $U_i \in \mathcal{N}$ and c_i from $U_i \in \mathcal{I} \setminus (\mathcal{R} \cup \mathcal{N})$, for i such that $U_i \in \mathcal{I} \setminus \mathcal{R}$, S computes $sid = TCR(\{c_i\}_{\mathcal{I} \setminus \mathcal{R}})$, chooses a representative user from $U_i \in \mathcal{N}$. Here, w.l.o.g., we suppose that U_1 is the representative user. For i such that $U_i \in \mathcal{N}$ and $U_{i+1} \in \mathcal{R}$, S sends (sid, R_j) to U_i where j is the minimum index such that $U_j \in \mathcal{N}$ and $j > i$. For i such that $U_i \in \mathcal{N}$ and $U_{i-1} \in \mathcal{R}$, S sends $(sid, R_{j'})$ to U_i where j is the maximum index such that $U_{j'} \in \mathcal{N}$ and $j' < i$. Then, S sends sid to $U_i \in \mathcal{I} \setminus (\mathcal{R} \cup \mathcal{N})$. Also, S notices that U_1 is the representative user.

(Round 2 for Users) For U_1, if $U_j = U_3$ and $U_{j'} = U_{n-1}$ hold, then on receiving (sid, R_3, R_{n-1}), U_1 computes $K_1^{(l)} = F(sid, R_{n-1}^{r_1})$, $K_1^{(r)} = F(sid, R_3^{r_1})$, $T_1 = K_1^{(l)} \oplus K_1^{(r)}$ and $T' = K_1^{(l)} \oplus (k_1 \| s_1)$. For U_1, if $U_{j'} = U_{n-1}$ and $U_2 \in \mathcal{N}$ hold, then on receiving (sid, R_{n-1}), U_1 computes $K_1^{(l)} = F(sid, R_{n-1}^{r_1})$, $K_1^{(r)} = F(sid, H_1^{(r)})$, $T_1 = K_1^{(l)} \oplus K_1^{(r)}$ and $T' = K_1^{(l)} \oplus (k_1 \| s_1)$. For U_1, if $U_j = U_3$ and $U_n \in \mathcal{N}$ hold, then on receiving (sid, R_3), U_1 computes $K_1^{(l)} = F(sid, H_1^{(l)})$, $K_1^{(r)} = F(sid, R_3^{r_1})$, $T_1 = K_1^{(l)} \oplus K_1^{(r)}$ and $T' = K_1^{(l)} \oplus (k_1 \| s_1)$. U_1 computes $\sigma_1 \leftarrow \mathbf{Tag}_{mk_1}(R_1, c_1, (R_3, R_{n-1},)\ T_1, T', U_1, sid)$, and sends (T_1, T', σ_1) to S.
On receiving (sid, R_j), U_i such that $U_i \in \mathcal{N}$ and $U_{i+1} \in \mathcal{R}$ hold computes $K_i^{(l)} = F(sid, H_i^{(l)})$, $K_i^{(r)} = F(sid, R_j^{r_i})$ and $T_i = K_i^{(l)} \oplus K_i^{(r)}$. U_i computes $\sigma_i \leftarrow \mathbf{Tag}_{mk_i}(R_i, c_i, R_j, k_i, s_i, T_i, U_i, sid)$, and sends $(k_i, s_i, T_i, \sigma_i)$ to S.
On receiving $(sid, R_{j'})$, U_i such that $U_i \in \mathcal{N}$ and $U_{i-1} \in \mathcal{R}$ hold computes $K_i^{(l)} = F(sid, R_{j'}^{r_i})$, $K_i^{(r)} = F(sid, H_i^{(r)})$ and $T_i = K_i^{(l)} \oplus K_i^{(r)}$. U_i computes $\sigma_i \leftarrow \mathbf{Tag}_{mk_i}(R_i, c_i, R_{j'}, k_i, s_i, T_i, U_i, sid)$, and sends $(k_i, s_i, T_i, \sigma_i)$ to S.
On receiving sid, $U_i \in \mathcal{I} \setminus (\mathcal{R} \cup \mathcal{N})$ computes $K_i^{(l)} = F(sid, H_i^{(l)})$, $K_i^{(r)} = F(sid, H_i^{(r)})$ and $T_i = K_i^{(l)} \oplus K_i^{(r)}$. U_i computes $\sigma_i \leftarrow \mathbf{Tag}_{mk_i}(c_i, k_i, s_i, T_i, U_i, sid)$, and sends $(k_i, s_i, T_i, \sigma_i)$ to S.

(Round 2 for Server) On receiving (T_1, T', σ_1) from U_1 and $(k_i, s_i, T_i, \sigma_i)$ from other users, S verifies the authentication-tag, and if the verification fails, then aborts. Also, for $U_i \in \mathcal{I} \setminus (U_1 \cup \mathcal{R})$, S checks if $c_i = g^{k_i} h^{s_i}$ holds, and if the verification fails, then aborts. S generates $\tilde{k}_S \in_R \{0,1\}^\kappa$, $\tilde{k}'_S \in_R \mathsf{Kspace}_\kappa$, $\tilde{K}_1 \in_R \{0,1\}^\kappa$ and $\tilde{K}'_1 \in_R Kspace_\kappa$ as ESK_S, and computes $k_S = tPRF(\tilde{k}_S, \tilde{k}'_S, st_S, st'_S)$ and $K_1 = tPRF'(\tilde{K}_1, \tilde{K}'_1, st_S, st'_S)$. For i such that $U_i \in \mathcal{I} \setminus (U_1 \cup \mathcal{R})$, S computes $k' = (\bigoplus\{k_i\}) \oplus k_S$. For i such that $U_i \in \mathcal{I} \setminus \mathcal{R}$, S computes $T'_i = \bigoplus_{1 \le j \le i-1} T_j$, where for j such that $U_j \in \mathcal{R}$, T_j is empty. For $U_i \in \mathcal{I} \setminus \mathcal{R}$, S computes $CT'_i \leftarrow \mathbf{AEnc}(Params, P_i, K_1)$ with access structure $P_i := (ID = U_i) \wedge (time \in TF)$.

S computes $\sigma'_1 \leftarrow \mathbf{Tag}_{mk_1}(R_1, c_1, (R_3, R_{n-1},)\ T_1, T', U_1, sid, k', CT'_1)$, and sends (k', CT'_1, σ'_1) to U_1.

For i such that $U_i \in \mathcal{N}$ and $U_{i+1} \in \mathcal{R}$, S computes $\sigma'_i \leftarrow \mathbf{Tag}_{mk_i}(R_i, c_i, R_j, k_i, s_i, T_i, U_i, sid, c_1, k', T'_i, T', CT'_i)$, and sends $(c_1, k', T'_i, T', CT'_i, \sigma'_i)$ to U_i.

For i such that $U_i \in \mathcal{N}$ and $U_{i-1} \in \mathcal{R}$, S computes $\sigma'_i \leftarrow \mathbf{Tag}_{mk_i}(R_i, c_i, R_{j'}, k_i, s_i, T_i, U_i, sid, c_1, k', T'_i, T', CT'_i)$, and sends $(c_1, k', T'_i, T', CT'_i, \sigma'_i)$ to U_i.

For $U_i \in \mathcal{I} \setminus (\mathcal{R} \cup \mathcal{N})$, S computes $\sigma'_i \leftarrow \mathbf{Tag}_{mk_i}(c_i, k_i, s_i, T_i, U_i, sid, c_1, k', T'_i, T', CT'_i)$, and sends $(c_1, k', T'_i, T', CT'_i, \sigma'_i)$ to U_i.

(Session Key Generation and Post Computation) On receiving $(c_1, k', T'_i, T', CT'_i, \sigma'_i)$, $U_i \in \mathcal{I} \setminus (U_1 \cup \mathcal{R})$ verifies the authentication-tag, and if the verification fails, then aborts. U_i computes $K_1^{(l)} = T'_i \oplus K_i^{(l)}$ and $k_1 \| s_1 = T' \oplus K_1^{(l)}$, and checks if $c_1 = g^{k_1} h^{s_1}$hold, and if the verification fails, then aborts. U_i decrypts $K_1 \leftarrow \mathbf{ADec}_{usk_i}(CT'_i, P_i)$, computes $K_2 = F'(sid, k' \oplus k_1)$, and outputs the session key $SK = F''(sid, K_1) \oplus F''(sid, K_2)$. As state information, U_i updates sid, $r = F'''(sid, K_1) \oplus F'''(sid, K_2)$ in $state_i$. For i such that $U_i \in \mathcal{N}$ and $U_{i+1} \in \mathcal{R}$, U_i updates $H_i^{(r)} = R_j^{r_i}$ in $state_i$. For i such that $U_i \in \mathcal{N}$ and $U_{i-1} \in \mathcal{R}$, U_i updates $H_i^{(l)} = R_{j'}^{r_i}$ in $state_i$.

On receiving (k', CT'_1, σ'_1), U_1 verifies the authentication-tag, and if the verification fails, then aborts. U_1 decrypts $K_1 \leftarrow \mathbf{ADec}_{usk_1}(CT'_1, P_1)$, computes $K_2 = F'(sid, k' \oplus k_1)$, and outputs the session key $SK = F''(sid, K_1) \oplus F''(sid, K_2)$. As state information, U_1 updates sid, $r = F'''(sid, K_1) \oplus F'''(sid, K_2)$ in $state_1$. If $U_2 \in \mathcal{R}$, then U_1 updates $H_1^{(r)} = R_j^{r_1}$ in $state_1$. If $U_n \in \mathcal{R}$, then U_1 updates $H_1^{(l)} = R_{j'}^{r_1}$ in $state_1$.

A.5 Update Phase

The Update phase is the same as the simple case.