PAPER *Special Section on Cryptography and Information Security*

Strongly Secure Predicate-Based Authenticated Key Exchange: Definition and Constructions*

Atsushi FUJIOKA[†a)], **Koutarou SUZUKI**[†b)], *and* **Kazuki YONEYAMA**[†c)], ***Members***

SUMMARY This paper firstly provides the extended Canetti-Krawzcyk (eCK) security model for predicate-based authenticated key exchange (AKE) that guarantees resistance to leakage of ephemeral secret keys. Moreover, we propose two-pass key-policy (resp. session-policy) attribute-based AKE protocol secure in the proposed predicate-based eCK security model based on key-policy (resp. ciphertext-policy) attribute-based encryption. The proposed protocols have advantages in security against leakage of ephemeral secret keys and the round complexity compared to the previous predicate-based AKE protocols.

key words: *predicate-based AKE, eCK security, key-policy, session-policy*

1. Introduction

PKI-based authenticated key exchange (AKE) [3], [7] enables a user to authenticate a peer by the peer's public key and its certificate, and to establish a common session key secretly. In PKI-based AKE, each user has public information, called *static public key*, and the corresponding secret information, called *static secret key*, i.e., long-term secret key. The static public key is expected to be certified with the user's identity by a system such as a public key infrastructure (PKI). A user who wants to share a key with a peer exchanges information several times and then computes the shared session key. In two-pass AKE, the user generates *ephemeral public key* and the corresponding *ephemeral secret key*, i.e., session-wise short-term secret key that is defined as all randomness generated in the session, and sends the ephemeral public key to the peer. The receiving peer also generates ephemeral public key and the corresponding ephemeral secret key and returns the ephemeral public key to the sender. Then, both parties compute *shared values* from their static public keys, the corresponding static secret keys, the exchanged ephemeral public keys, and the corresponding ephemeral secret keys, and derive a *session key* from these values including the shared values. The session key is computed with a function called *key derivation function*, and in most cases, the key derivation function is a hash function regarded as a random oracle.

On the other hand, in ID-based AKE [6], [9], [14], [17], user is authenticated by user's ID, to reduce the costs of the management of certificates. In predicate-based AKE [4], [12] that is a natural extension of ID-based AKE, user is authenticated by *specifying the attributes*. For instance, user U wants to establish a session key with someone who offers a service, but U may not want to reveal any information except whether U has a right to receive the service or not. In another situation, U is not concerned with the peer's ID but with the peer's condition (i.e., attributes or policy) specified by U. For such situations where ID-based AKE is not applicable, predicate-based AKE is quite useful.

We consider a scenario that user U and U' try to exchange a session key using the predicate-based AKE. Each user U is first given a static secret key based on his condition γ_U by the key generation center (KGC). Next, U (U') specifies the condition δ_U ($\delta_{U'}$), which the peer U' (U) is expected to satisfy, and exchanges ephemeral public key based on the condition δ_U ($\delta_{U'}$) with peer U' (U), respectively. We represent whether γ satisfies δ with a predicate $\phi : \{0,1\}^l \times \{0,1\}^l \to \{0,1\}$, where l is a positive integer. When two users U and U' perform predicate-based AKE, both user can compute the session key if and only if $\phi(\gamma_U, \delta_{U'}) = 1$ and $\phi(\gamma_{U'}, \delta_U) = 1$.

As a special case of predicate-based AKE, we have attribute-based AKE, where these conditions γ_U and δ_U are implemented as an access policy and a set of attributes. We say an attribute-based AKE is *key-policy* attribute-based AKE (KP-AB-AKE), if an access policy is encoded in the static secret key. We say also an attribute-based AKE is *session-policy* attribute-based AKE (SP-AB-AKE), if an access policy is encoded in the ephemeral public key. In the case of $\phi(\gamma, \delta) = (\gamma \stackrel{?}{=} \delta)$, we have ID-based AKE as an special case of the predicate-based AKE.

1.1 Motivation

Wang, Xu and Ban [19], and Wang, Xu and Fu [20], [21] proposed simple variants of predicate-based AKE. In their protocols, attributes are regarded as identification strings, and there is no mechanism for evaluating predicate. Thus, their protocols are a kind of ID-based AKE rather than predicate-based AKE.

Gorantla, Boyd and González Nieto [12] proposed attribute-based group AKE protocol (BGGN10). Their protocol is generically constructed with a attribute-based KEM, and allows users access control based on the users' attributes. However, the condition is common for all users. Thus, the protocol does not fit in the our predicate-based

Manuscript received March 31, 2011.
Manuscript revised July 28, 2011.

[†]The authors are with NTT Information Sharing Platform Laboratories, NTT Corporation, Musashino-shi, 180-8585 Japan.

*Extended abstracts of this paper are appeared in [11] and [23].

a) E-mail: fujioka.atsushi@lab.ntt.co.jp
b) E-mail: suzuki.koutarou@lab.ntt.co.jp
c) E-mail: yoneyama.kazuki@lab.ntt.co.jp

DOI: 10.1587/transfun.E95.A.40

Table 1 Comparison with the existing predicate-based AKE protocols.

Protocol	Round complexity	Type	Access policy	Security model	Assumption
BGGN10 [12]	2 pass	group	SP	BR	AB-KEM
BS10 [4]	3 pass	2-party	SP or KP	BR, C-privacy	PB-signature
SSC10 [18]	4 pass	group	SP	BR	AB-signcryption
Proposed in Sec.3	2 pass	2-party	KP	eCK	gap BDH, ROM
Proposed in Sec.4	2 pass	2-party	SP	eCK	gap BDH, ROM

AKE scenario since each user cannot specify the condition in which the peers are expected to satisfy each other in the protocol. The protocol is constructed based on the attribute-based key encapsulation mechanism (AB-KEM) and security is proved in a security model based on the Bellare-Rogaway (BR) model [3].

Birkett and Stebila [4] proposed predicate-based 2-party AKE protocol (BS10). Their protocol is generically constructed with a predicate-based signature, i.e., their protocol is signed DH construction using a predicate-based signature (PB-signature). They prove security of their protocol without random oracles in a security model based on the BR model [3].

Steinwandt and Suárez Corona [18] proposed attribute-based group AKE protocol (SSC10). Their protocol is generically constructed with a attribute-based signcryption (AB-signcryption). They prove security of their protocol without random oracles in a security model based on the BR model [3].

However, security in [12], [4], and [18] cannot guarantee resistance to *leakage of ephemeral secret keys* since the BR model does not capture this property. Leakage of ephemeral secret keys will occur due to various factors, e.g., a pseudo-random generator implemented in the system is poor, the ephemeral secret key itself is leaked by physical attacks such as side channel attacks. Thus, if we consider such cases, resistance to leakage of ephemeral secret keys is important for AKE protocols, and the extended Canetti-Krawzcyk (eCK) security model [15], which captures leakage of ephemeral secret keys, is used for security proofs of some recent AKE protocols. As Birkett and Stebila stated in their conclusion [4], there is no predicate-based AKE protocol that satisfies resistance to leakage of ephemeral secret keys, and finding such a protocol has been an open problem.

To prove the security in the eCK model, it is known that the NAXOS technique [15] is effective. The NAXOS technique involves that each user applies a hash function to the static and ephemeral secret keys and computes an ephemeral public key by using the output of the hash function as the exponent of the ephemeral public key. An adversary cannot know the output of the function as long as the adversary cannot obtain the static secret key even if the ephemeral secret key is leaked. As Birkett and Stebila stated in their conclusion [4], the BS10 protocol does not become secure against leakage of ephemeral secret keys since we have no predicate-based signature which is secure against revealing the randomness used in signing.

Another problem of the BS10 protocol is in the communication round. Most eCK secure AKE protocols only need two-pass message transmission with no key confirmation, where key confirmation means that each user can explicitly verify that the session key is common with the peer. If necessary, key confirmation is easily obtained by sending confirmation information as the third pass. In the BS10 protocol, key confirmation is mandatory, and so the protocol always needs three passes. On some applications, predicate-based AKE is not necessarily required to have key confirmation. Since the construction of the BS10 protocol is based on the signed-DH paradigm [7], the third pass (for key confirmation) in the BS10 protocol cannot be removed for the security proof. We construct a two-pass eCK secure predicate-based AKE with no key confirmation and it has certain significance.

1.2 Contribution

In this paper, we have mainly two contributions: the first definition of predicate-based extended Canetti-Krawzcyk (eCK) security model, and the first constructions of key-policy and session-policy attribute-based AKE protocol, which needs only two-pass message transmission and is secure in the predicate-based eCK security model. Comparison of the proposed protocols with existing predicate-based AKE protocols is summarized in the Table 1. In the Table 1, the column of "Round complexity" shows the number of passes of the protocol. The column of "Type" shows the protocol is group or 2-party AKE protocol. The column of "Access policy" shows the protocol is key-policy (KP) or session-policy (SP). The column of "Security model" shows the security model in which the protocol is secure, where BR stands for predicate-based Bellare-Rogaway model [3], eCK stands for predicate-based extended Canetti-Krawzcyk model [15], and C-privacy stands for credential privacy [4]. The column of "Assumption" shows the assumption on which the protocol based, where gap BDH stands for the gap Bilinear Diffie-Hellman assumption and ROM stands for the random oracle model.

Predicate-based eCK Security Model

In contrast with the security model used by Birkett and Stebila [4] (based on the BR model), we allow the adversary to obtain static secret keys, the master key, and ephemeral secret keys individually to make the eCK model suitable to the predicate-based AKE setting. Obviously, if the adversary obtains the ephemeral secret key and the static secret key of an user (or the master key) together, the session key can be justly computed by the adversary. Thus, we have to consider the *freshness* of the session. Freshness is the condition of the session in which the adversary cannot trivially break the secrecy of the session key. Although the adversary is not allowed to reveal any secret information of the fresh session in the security model used in [4], we allow the adversary to take most malicious behaviors concerned with revealing secret information in the proposed security model. In the predicate-based AKE setting, especially, a static secret key corresponding to condition γ may also be usable as the static secret key corresponding to another condition γ' if γ' implies γ. Thus, our freshness definition is defined carefully in such an implication.

Two-pass Key-policy and Session-policy Attribute-based AKE Protocols

We proposes a two-pass key-policy attribute-based authenticated key exchange protocol (KP-AB-AKE) based on a key-policy attribute-based encryption [16], [13], and a two-pass session-policy attribute-based authenticated key exchange protocol (SP-AB-AKE) based on a ciphertext-policy attribute-based encryption [22]. In the proposed key-policy attribute-based AKE, a predicate $\phi(\gamma, \delta)$ is implemented using access tree. In the proposed session-policy attribute-based AKE, a predicate $\phi(\gamma, \delta)$ is implemented using linear secret sharing scheme (LSSS). And we prove that the proposed two-pass key-policy and session-policy attribute-based AKEs are secure in the predicate-based eCK security model as predicate-based AKE.

The proposed protocols achieve efficiency in two-pass message transmission, though the BS10 protocol needs three passes. Our protocol does not have key confirmation, but it is not essential for AKE. If key confirmation is required, it is easy to change the ordinary two-pass AKE protocol to have key confirmation by sending confirmation information as the third pass. Generally, since a session using the session key follows after the AKE session, we can simultaneously send confirmation information in the first pass of the following session.

1.3 Related Work

Models for key agreement were introduced by Bellare and Rogaway [3] and Blake-Wilson, Johnson and Menezes [5], in the shared- and public-key settings, respectively. Recent developments in two-party certificate-based AKE in the public-key infrastructure (PKI) setting have improved the security models and definitions, i.e., Canetti-Krawzcyk (CK) security model [7] and LaMacchia, Lauter and Mityagin's extended Canetti-Krawzcyk (eCK) security model [15].

On the other hand, some ID-based AKE protocols have been proposed, see for example [9], [17]. Boyd and Choo [6] state that many existing ID-based AKE protocols are not as secure as we expect them to be. Furthermore, security analysis for ID-based AKE protocols do not formally analyze ephemeral private key leakage. The only protocols that formally consider this scenario are proposed in [10], [11], [14].

In Sect. 2, we define the predicate-based eCK security model. In Sects. 3 and 4, we propose our two-pass key-policy and session-policy attribute-based AKE protocols, which are secure in the predicate-based eCK security model. In Sect. 5, we conclude the paper.

2. eCK Security Model for Predicate-Based AKE

In this section, we provide eCK-security model for two-pass predicate-based AKEs. The differences of predicate-based eCK model from PKI-based eCK model are as follows.

1. Owner U_i of a session is specified by string γ_i, which represents the condition that the static secret key of the owner satisfies, and string δ_i, which represents the condition that the static secret key of the peer should satisfy, instead of public key of users.
2. Session $(\gamma_A, \delta_A, X_A, X_B)$ and session $(\gamma_B, \delta_B, X_A, X_B)$ are matching if $\phi(\gamma_A, \delta_B) = 1$ and $\phi(\gamma_B, \delta_A) = 1$, where ϕ is a predicate and $\phi(\gamma, \delta) = 1$ means γ satisfies δ.
3. In the freshness condition, if a static secret key corresponding to γ s.t. $\phi(\gamma, \delta_B) = 1$ is revealed, we regard that the static secret key corresponding to γ_A s.t. $\phi(\gamma_A, \delta_B) = 1$ is also revealed.
4. The adversary additionally can reveal the master secret key.

In the case of $\phi(\gamma, \delta) = (\gamma \stackrel{?}{=} \delta)$, the predicate-based eCK model coincide with the ID-based eCK mode [14].

We denote a user by U_i, and user U_i and other parties are modeled as probabilistic polynomial-time Turing machines w.r.t. security parameter κ. Each user U_i has the static secret key corresponding to the string γ_i, which represents the condition that the static secret key of the user satisfies. In a session, user U_i sends to the peer the ephemeral public key corresponding to the string δ_i, which represents the condition that the static secret key of the peer should satisfy. Each user U_i obtains the static secret key corresponding to the string γ_i from a key generation center (KGC) via a secure and authentic channel. The center KGC uses a master secret key to generate individual secret keys.

Algorithms

Predicate-based AKE protocol Π consists of the following algorithms. We denote a user as U_i and the user's associated string as γ_i. User U_i and other parties are modeled as a probabilistic polynomial-time Turing machine. Predicate ϕ : $\{0,1\}^l \times \{0,1\}^l \to \{0,1\}$, where l is an integer, is given as a part of the public parameters. Predicate ϕ takes two input strings γ and δ and outputs a bit $\phi(\gamma, \delta) = 1$ or 0.

Key Generation:

The key generation algorithm KeyGen takes a security parameter 1^k as input, and outputs a master secret key msk and a master public key mpk, i.e.,

$$\mathsf{KeyGen}(1^k) \to (msk, mpk).$$

Key Extraction:

The key extraction algorithm KeyExt takes the master secret key msk, the master public key mpk, and a string γ_i given by user U_i, which represents the condition that the static secret key of the owner satisfies, and outputs a static secret key ssk_{γ_i} corresponding to the string γ_i, i.e.,

$$\mathsf{KeyExt}(msk, mpk, \gamma_i) \rightarrow ssk_{\gamma_i}.$$

Key Exchange:
Users U_A and U_B share a session key K by performing the following 2-pass protocol. User U_A (U_B) selects string δ_A (δ_B), which represents the condition that the static secret key of the peer should satisfy.

User U_A computes 1st message X_A by using the algorithm Message, which takes the master public key mpk, the string γ_A, the static secret key ssk_{γ_A}, and the string δ_A, and outputs 1st message X_A, i.e.,

$$\mathsf{Message}(mpk, \gamma_A, ssk_{\gamma_A}, \delta_A) \rightarrow X_A.$$

User U_A sends 1st message X_A to user U_B.

Upon receiving 1st message X_A, user U_B computes 2nd message X_B by using the algorithm Message, which takes the master public key mpk, the string γ_B, the static secret key ssk_{γ_B}, the string δ_B, and the received messages X_A, and outputs 2nd message X_B, i.e.,

$$\mathsf{Message}(mpk, \gamma_B, ssk_{\gamma_B}, \delta_B, X_A) \rightarrow X_B.$$

User U_B sends 2nd message X_B to user U_A.

User U_B computes a session key K by using the algorithm SessionKey, which takes the master public key mpk, the string γ_B, the static secret key ssk_{γ_B}, the string δ_B, and the received and sent messages X_A, X_B, and outputs session key K, i.e.,

$$\mathsf{SessionKey}(mpk, \gamma_B, ssk_{\gamma_B}, \delta_B, X_A, X_B) \rightarrow K.$$

Upon receiving 2nd message X_B, user U_A computes a session key K by using the algorithm SessionKey, which takes the master public key mpk, the string γ_A, the static secret key ssk_{γ_A}, the string δ_A, and the sent and received messages X_A, X_B, and outputs session key K, i.e.,

$$\mathsf{SessionKey}(mpk, \gamma_A, ssk_{\gamma_A}, \delta_A, X_A, X_B) \rightarrow K.$$

Both users U_A and U_B can compute the same session key if and only if $\phi(\gamma_A, \delta_B) = 1$ and $\phi(\gamma_B, \delta_A) = 1$.

Session
An invocation of a protocol is called a *session*. A session is activated via an incoming message of the forms $(\Pi, \mathcal{I}, \gamma_A, \delta_A)$ or $(\Pi, \mathcal{R}, \gamma_A, \delta_A, X_B)$, where Π is a protocol identifier and $\mathcal{I}/\mathcal{R}$ are initiator/responder identifier. If U_A was activated with $(\Pi, \mathcal{I}, \gamma_A, \delta_A)$, then U_A is the session *initiator*, otherwise the session *responder*. After activation, U_A appends an ephemeral public key X_A to the incoming message and sends it as an outgoing response. If U_A is the responder, U_A computes a session key. A party U_A that has been successfully activated via $(\Pi, \mathcal{I}, \gamma_A, \delta_A)$, can be further activated via $(\Pi, \mathcal{I}, \gamma_A, \delta_A, X_A, X_B)$ to compute a session key.

If U_A is the initiator, the session is identified via $\mathtt{sid} = (\Pi, \mathcal{I}, \gamma_A, \delta_A, X_A)$ or $\mathtt{sid} = (\Pi, \mathcal{I}, \gamma_A, \delta_A, X_A, X_B)$. If U_A is the responder, the session is identified via $\mathtt{sid} = (\Pi, \mathcal{R}, \gamma_A, \delta_A, X_B, X_A)$. We say that U_A is *owner* of session sid if the third coordinate of session sid is γ_A. We say that U_A is *peer* of session sid if U_A sends ephemeral public key X_A of session sid to owner of session sid. We say that a session is *completed* if its owner computes a session key.

The *matching session* of $(\Pi, \mathcal{I}, \gamma_A, \delta_A, X_A, X_B)$ is a session $(\Pi, \mathcal{R}, \gamma_B, \delta_B, X_A, X_B)$ s.t.

$$\phi(\gamma_A, \delta_B) = 1 \text{ and } \phi(\gamma_B, \delta_A) = 1,$$

and vice versa.

From now on, we omit protocol identifier Π, and we omit $\mathcal{I}$ and $\mathcal{R}$ since these "role markers" are implicitly defined by the order of X_A and X_B.

Adversary
The adversary $\mathcal{A}$, that is modeled as a probabilistic polynomial-time Turing machine, controls all communications between parties including session activation by performing the following adversary query.

- Send(message): The message has one of the following forms: (γ_A, δ_A), $(\gamma_A, \delta_A, X_B)$, or $(\gamma_A, \delta_A, X_A, X_B)$. The adversary $\mathcal{A}$ obtains the response from the user.

Note that the adversary does not control the communication between parties and the key generation center. For simplicity, we assume that strings and corresponding static keys are part of $\mathcal{A}$'s input.

A party's secret information is not accessible to the adversary. However, leakage of secret information is captured via the following adversary queries.

- SessionKeyReveal(sid) The adversary obtains the session key for the session sid, provided that the session holds a session key.
- EphemeralKeyReveal(sid) The adversary obtains the ephemeral secret key associated with the session sid.
- StaticKeyReveal(γ_i) The adversary learns the static secret key corresponding to string γ_i.
- MasterKeyReveal() The adversary learns the master secret key of the system.
- EstablishParty(γ_i) This query allows the adversary to establish a party with string γ_i, i.e., the adversary totally controls that party. If a party is established by an EstablishParty(γ_i) query issued by the adversary, then we call the party *dishonest*, otherwise we call the party *honest*. This query models malicious insiders.

Freshness
Our security definition requires the notion of "freshness".

Definition 1 (Freshness): Let $\mathtt{sid}^*$ be the session identifier of a completed session, owned by an honest party U_A with honest peer U_B. If the matching session exists, then let $\overline{\mathtt{sid}^*}$ be the session identifier of the matching session of $\mathtt{sid}^*$. Define $\mathtt{sid}^*$ to be fresh if none of the following conditions hold:

1. $\mathcal{A}$ issues SessionKeyReveal($\mathtt{sid}^*$) or SessionKeyReveal($\overline{\mathtt{sid}^*}$) (if $\overline{\mathtt{sid}^*}$ exists).

2. $\overline{\mathtt{sid}^*}$ exists and $\mathcal{A}$ makes either of the following queries
   - both StaticKeyReveal(γ) s.t. $\phi(\gamma, \delta_B) = 1$ and EphemeralKeyReveal($\mathtt{sid}^*$), or
   - both StaticKeyReveal(γ) s.t. $\phi(\gamma, \delta_A) = 1$ and EphemeralKeyReveal($\overline{\mathtt{sid}^*}$).
3. $\overline{\mathtt{sid}^*}$ does not exist and $\mathcal{A}$ makes either of the following queries
   - both StaticKeyReveal(γ) s.t. $\phi(\gamma, \delta_B) = 1$ and EphemeralKeyReveal($\mathtt{sid}^*$), or
   - StaticKeyReveal(γ) s.t. $\phi(\gamma, \delta_A) = 1$.

Note that if $\mathcal{A}$ issues a MasterKeyReveal() query, we regard $\mathcal{A}$ as having issued both a StaticKeyReveal(γ) query s.t. $\phi(\gamma, \delta_B) = 1$ and a StaticKeyReveal(γ) query s.t. $\phi(\gamma, \delta_A) = 1$.

Security Experiment

The adversary $\mathcal{A}$ starts with a set of honest parties U_i, for whom $\mathcal{A}$ adaptively selects strings γ_i. The adversary makes an arbitrary sequence of the queries described above. During the experiment, $\mathcal{A}$ makes a special query Test($\mathtt{sid}^*$) and is given with equal probability either the session key held by $\mathtt{sid}^*$ or a random key. The experiment continues until $\mathcal{A}$ makes a guess whether the key is random or not. The adversary *wins* the game if the test session $\mathtt{sid}^*$ is fresh at the end of $\mathcal{A}$'s execution and if $\mathcal{A}$'s guess was correct.

Definition 2 (Security): The advantage of the adversary $\mathcal{A}$ in the experiment with AKE protocols Π is defined as

$$\mathrm{Adv}_{\Pi}^{\mathrm{AKE}}(\mathcal{A}) = \Pr[\mathcal{A}\ wins] - \frac{1}{2}.$$

We say that Π is a secure AKE protocol if the following conditions hold:

1. If two honest parties complete matching sessions, then, except with negligible probability in security parameter κ, they both compute the same session key.
2. For any probabilistic polynomial-time bounded adversary $\mathcal{A}$, $\mathrm{Adv}_{\Pi}^{\mathrm{AKE}}(\mathcal{A})$ is negligible in security parameter κ.

Moreover, we say that predicate-based AKE protocol Π is *selective-condition secure in the predicate-based eCK model*, if the adversary $\mathcal{A}$ outputs target condition (δ_A, δ_B) at the beginning of the security experiment.

3. Proposed Two-Pass Key-Policy Attribute-Based AKE Protocol

We constructed a key-policy attribute-based AKE protocol (KP-AB-AKE) based on attribute-based encryption schemes [13], [16]. By applying the NAXOS technique [15], the proposed protocol can satisfy the predicate-based eCK security, under the gap Bilinear Diffie-Hellman (BDH) assumption [1] in the random oracle model.

3.1 Access Tree

In the proposed attribute-based AKE protocol, we use a predicate $\phi(\gamma, \delta)$, where the output of ϕ corresponds to whether a set of attributes δ satisfies an access tree γ, i.e., the first input γ of ϕ corresponds to the access tree and the second input δ of ϕ corresponds to the set of attributes. Thus, a static secret key of user U_A is represented as an access tree γ_A, and a ephemeral public key of user U_B is represented as a set of attributes δ_B. We show an explanation of the access tree and the set of attributes below.

Let γ be a tree and let $L(\gamma)$ be the set of all leaf nodes of γ. Let c_u be the number of child nodes of non-leaf node u, and we set $c_u = 1$ for leaf node u. Threshold value k_u $(1 \leq k_u \leq c_u)$ is assigned to each node u, and thus $k_u = c_u = 1$ for each node u. For each node u, we assign $index(u) \in \{1, \ldots, c_w\}$, where w is the parent of u and u is the $index(u)$-th child of w. Then, each node u of the tree is associated with the k_u-out-of-c_u threshold gate. Let $\mathcal{U} = \{1, 2, \ldots, n\}$ be the universe of attributes, and we assign $att(u) \in \mathcal{U}$ for each leaf node u. Then, each leaf node u of the tree is associated with an attribute. We call the tree γ with $(k_u, index(u), att(u))$ *access tree*.

We define that the set of attributes $\delta \subset \mathcal{U}$ satisfies access tree γ as follows: For each leaf node u, we say u is satisfied if and only if $att(u) \in \delta$. For non-leaf node u, we say u is satisfied if and only if the number of satisfied child nodes of u is equal to or greater than k_u. Then, we say the set of attributes satisfies the access tree if and only if the (non-leaf) root node u_r is satisfied.

By using the above access tree, we can describe any monotone circuit consists of OR and AND gates, since a threshold gate implies an OR gate when $k_u = 1$ and implies an AND gate when $k_u = c_u$. Moreover, the access tree enables us to describe any general circuit by adding attribute $\bar{a}$ representing NOT of each attribute a to the universe $\mathcal{U}$ of attributes, since a NOT gate can be moved to a leaf node of the access tree by using De Morgan's laws.

3.2 Proposed Two-Pass Key-Policy Attribute-Based AKE Protocol

In this section, we describe the proposed key-policy attribute-based AKE protocol (KP-AB-AKE). In the protocol, a predicate $\phi(\gamma, \delta)$ is implemented using access tree, i.e., the first input γ of predicate ϕ corresponds to an access tree on the attribute set, the second input δ of predicate ϕ corresponds to a subset of the attribute set, and the output of predicate ϕ corresponds to whether the subset δ of the attribute set satisfies the access tree γ on the attribute set.

Parameters:

Let k be the security parameter and G, G_T be bilinear groups with pairing $e : G \times G \rightarrow G_T$ of order k-bit prime p with generators g, g_T, respectively. Let $H : \{0,1\}^* \rightarrow \{0,1\}^k$ and $H' : \{0,1\}^* \rightarrow \mathbb{Z}_p$ be cryptographic hash functions modeled as random oracles. In addition, let $\Delta_{i,S}$ for $i \in \mathbb{Z}_p$ and a set

S of elements in $\mathbb{Z}_p$ be the Lagrange coefficient such that $\Delta_{i,S}(x) = \prod_{j\in S, j\neq i} \frac{x-j}{i-j}$.

Key Generation:
The key generator (algorithm) randomly selects a master secret key $z \in_U \mathbb{Z}_p$ and $\{t_i \in_U \mathbb{Z}_p\}_{i\in\mathcal{U}}$ for each attribute. Also, the key generator publishes the master public key $Z = g_T^z \in G_T$ and $\{T_i = g^{t_i}\}_{i\in\mathcal{U}}$.

Key Extraction:
For a given access tree γ_A of user U_A, the key extractor (algorithm) computes the static secret key $\{D_u\}_{u\in L(\gamma_A)}$ by choosing a polynomial q_u for each node u in γ_A as follows:

First, the key extractor sets the degree of the polynomial q_u to be $d_u = k_u - 1$. For the root node u_r, $q_{u_r}(0) = z$ and other d_{u_r} points of q_{u_r} are randomly chosen from $\mathbb{Z}_p$. Thus, q_{u_r} is fixed and other $c_{u_r} - d_{u_r}$ points are determined. For any other node u, set $q_u(0) = q_{u'}(index(u))$, where u' is the parent node of u and other d_u points of q_u are randomly chosen from $\mathbb{Z}_p$. Polynomials of all nodes are recursively determined with this procedure. Next, the key extractor computes a secret value for each leaf node u as $D_u = g^{\frac{q_u(0)}{t_i}}$, where $i = att(u)$. Finally, the key extractor returns the set $\{D_u\}_{u\in L(\gamma_A)}$ of the above secret values as the static secret key.

The static secret key $\{D_u\}_{u\in L(\gamma_B)}$ for an access tree γ_B of user U_B is derived from the same procedure.

Key Exchange:
In the following description, user U_A is the session initiator and user U_B is the session responder. User U_A has static secret keys $\{D_u\}_{u\in L(\gamma_A)}$ corresponding to an access tree γ_A, and user U_B has static secret keys $\{D_u\}_{u\in L(\gamma_B)}$ corresponding to an access tree γ_B. Then, user U_A sends to user U_B ephemeral public keys $(X, \{T_i^x\}_{i\in\delta_A})$ corresponding to the set of attributes δ_A, and user U_B sends to user U_A ephemeral public keys $(Y, \{T_i^y\}_{i\in\delta_B})$ corresponding to the set of attributes δ_B. Finally, both users U_A and U_B compute the shared key K if and only if $\phi(\gamma_A, \delta_B) = 1$ and $\phi(\gamma_B, \delta_A) = 1$, i.e., the set of attributes δ_B satisfies the access tree γ_A and the set of attributes δ_A satisfies access tree γ_B.

1. First, U_A determines a set $\delta_A \subset \mathcal{U}$ of attributes in which he hopes δ_A satisfies the access tree γ_B of U_B. U_A randomly chooses an ephemeral private key $\tilde{x} \in_U \mathbb{Z}_p$. Then, U_A computes $x = H'(\{D_u\}_{u\in L(\gamma_A)}, \tilde{x})$, and the ephemeral public key $X = g^x$ and $\{T_i^x\}_{i\in\delta_A}$. U_A sends $X, \{T_i^x\}_{i\in\delta_A}$ and the set of attributes δ_A to U_B.
2. Upon receiving $X = g^x$, $\{T_i^x\}_{i\in\delta_A}$, and δ_A, U_B determines a set of attributes δ_B in which he hopes δ_B satisfies the access tree γ_A of U_A. U_B randomly chooses an ephemeral private key $\tilde{y} \in_U \mathbb{Z}_p$. Then, U_B computes $y = H'(\{D_u\}_{u\in L(\gamma_B)}, \tilde{y})$, and the ephemeral public key $Y = g^y$ and $\{T_i^y\}_{i\in\delta_B}$. U_B sends $Y, \{T_i^y\}_{i\in\delta_B}$ and the set of attributes δ_B to U_A.
   U_B computes the shared secrets as follows: First, for each leaf node u in γ_B, U_B computes $e(D_u, T_j^x) = e(g^{\frac{q_u(0)}{t_j}}, g^{xt_j}) = e(g,g)^{xq_u(0)}$, where $j = att(u)$, if $att(u) \in \delta_A$. Next, for each non-leaf node u in γ_B, set $\tilde{S_u}' = \{u_c | u_c$ is a child node of u s.t. $e(g,g)^{xq_{u_c}(0)}$ is obtained.$\}$. If $|\tilde{S_u}'| \geq k_u$, U_B sets $\tilde{S_u} \subset \tilde{S_u}'$ s.t. $|\tilde{S_u}| = k_u$ and $S_u = \{index(u_c) | u_c \in \tilde{S_u}\}$, and computes
   $$\prod_{u_c\in\tilde{S_u}} (e(g,g)^{xq_{u_c}(0)})^{\Delta_{i,S_u}(0)} = \prod_{u_c\in\tilde{S_u}} e(g,g)^{xq_u(i)\cdot\Delta_{i,S_u}(0)} = e(g,g)^{xq_u(0)},$$
   where $i = index(u_c)$. This computation validly works by using polynomial interpolation. On the output of the root node u_r of γ_B, U_B obtains $e(g,g)^{xq_{u_r}(0)} = e(g,g)^{xz}$ if δ_A satisfies the access tree γ_B.
   Then, U_B sets the shared secrets
   $$\sigma_1 = e(g,g)^{xz},\ \sigma_2 = Z^y,\ \sigma_3 = X^y,$$
   and the session key $K = H(\sigma_1, \sigma_2, \sigma_3, \Pi, (\delta_A, X, \{T_i^x\}_{i\in\delta_A}), (\delta_B, Y, \{T_i^y\}_{i\in\delta_B}))$. U_B completes the session with session key K.
3. Upon receiving Y, $\{T_i^y\}_{i\in\delta_B}$ and δ_B, U_A computes the shared secrets as follows: First, for each leaf node u in γ_A, U_A computes $e(D_u, T_j^y) = e(g^{\frac{q_u(0)}{t_j}}, g^{yt_j}) = e(g,g)^{yq_u(0)}$, where $j = att(u)$, if $att(u) \in \delta_B$. Next, for each non-leaf node u in γ_A, U_B computes if $|\tilde{S_u}'| \geq k_u$.
   $$\prod_{u_c\in\tilde{S_u}} (e(g,g)^{yq_{u_c}(0)})^{\Delta_{i,S_u}(0)} = \prod_{u_c\in\tilde{S_u}} e(g,g)^{yq_u(i)\cdot\Delta_{i,S_u}(0)} = e(g,g)^{yq_u(0)},$$
   where $i = index(u_c)$. On the output of the root node u_r of γ_A, if δ_B satisfies the access tree γ_A, U_A obtains $e(g,g)^{yq_{u_r}(0)} = e(g,g)^{yz}$.
   Then, U_A sets the shared secrets
   $$\sigma_1 = Z^x,\ \sigma_2 = e(g,g)^{yz},\ \sigma_3 = Y^x,$$
   and the session key $K = H(\sigma_1, \sigma_2, \sigma_3, \Pi, (\delta_A, X, \{T_i^x\}_{i\in\delta_A}), (\delta_B, Y, \{T_i^y\}_{i\in\delta_B}))$. U_A completes the session with session key K.

   The shared secrets are
   $$\sigma_1 = g_T^{xz}, \sigma_2 = g_T^{yz}, \sigma_3 = g^{xy},$$

and therefore they can compute the same session key K if $\phi(\gamma_A, \delta_B) = 1$ and $\phi(\gamma_B, \delta_A) = 1$.

3.3 Security

We introduce the gap Bilinear Diffie-Hellman (BDH) problem [1] as follows. Let k be the security parameter and p be a k-bit prime. Let G be a cyclic group of a prime order p with a generator g, and G_T be a cyclic group of the prime order p with a generator g_T. Let $e : G \times G \to G_T$ be a polynomial-time computable bilinear non-degenerate map called pairing. We say that G, G_T are bilinear groups with the pairing e.

Define the computational BDH function BDH : $G^3 \rightarrow G_T$ as $\mathrm{BDH}(g^a, g^b, g^c) = e(g,g)^{abc}$ and the decisional BDH predicate DBDH : $G^4 \rightarrow \{0,1\}$ as a function which takes an input $(g^a, g^b, g^c, e(g,g)^d)$ and returns the bit one if $abc = d \bmod p$ or the bit zero otherwise. An adversary $\mathcal{A}$ is given input $g^a, g^b, g^c \in_U G$ selected uniformly randomly and can access the DBDH$(\cdot,\cdot,\cdot,\cdot)$ oracle, and tries to compute $\mathrm{BDH}(g^a, g^b, g^c)$. For adversary $\mathcal{A}$, we define advantage

$$Adv^{\mathrm{gap\ BDH}}(\mathcal{A}) = \Pr[g^a, g^b, g^c \in_U G,$$
$$\mathcal{A}^{\mathrm{DBDH}(\cdot,\cdot,\cdot,\cdot)}(g^a, g^b, g^c) = \mathrm{BDH}(g^a, g^b, g^c)],$$

where the probability is taken over the choices of g^a, g^b, g^c and the random tape of adversary $\mathcal{A}$.

Definition 3 (gap BDH assumption): We say that G satisfies the gap BDH assumption, if for any probabilistic polynomial-time adversary $\mathcal{A}$, advantage $Adv^{\mathrm{gapBDH}}(\mathcal{A})$ is negligible in security parameter k.

The proposed key-policy attribute-based AKE protocol is selective-condition secure in the predicate-based eCK security model under the gap BDH assumption and in the random oracle model.

Theorem 4: If G is a group, where the gap BDH assumption holds and H and H' are random oracles, the proposed attribute-based AKE protocol is selective-condition secure in the predicate-based eCK model described in Sect. 2.

Proof of Theorem 4 is provided in Appendix A. Here, we provide sketch of the proof. Adversary $\mathcal{A}$ can reveal the master secret key, static secret keys and ephemeral secret keys in the test session according to Definition 1.

First, when $\mathcal{A}$ poses an EphemeralKeyReveal query, $\tilde{x}$ and $\tilde{y}$ may be revealed. However, using the NAXOS technique (i.e. $H'(\{D_u\}_{u \in L(\gamma_A)}, \tilde{x})$ for x and $H'(\{D_u\}_{u \in L(\gamma_B)}, \tilde{y})$ for y), x and y are not revealed as long as $\{D_u\}_{u \in L(\gamma_A)}$ and $\{D_u\}_{u \in L(\gamma_B)}$ are not revealed respectively. Since $\mathcal{A}$ cannot pose EphemeralKeyReveal and StaticKeyReveal queries for the same user in the test session as in Definition 1, an EphemeralKeyReveal query gives no advantage to $\mathcal{A}$.

Next, we consider the case when the StaticKeyReveal query for party U_A is posed and there is no matching session. Then, the simulator cannot embed the BDH instances (g^u, g^v, g^w) into the static secret key of U_A and the ephemeral public key of the peer. However, the simulator can still embed $e(g^u, g^v)$ into the master public key $Z = e(g,g)^z$ and g^w into T_i^x for all i as $g^{w\beta_i}$, where $T_i = g^{\beta_i}$, because $\mathcal{A}$ cannot pose the MasterKeyReveal query and reveal x for U_A. Thus, the simulation successfully works, and the simulator can obtain $e(g,g)^{uvw}$ from $\sigma_1 = e(g,g)^{xz}$.

Finally, we consider the case when MasterKeyReveal query or both StaticKeyReveal queries for the test session and its matching session is posed. Then, the simulator cannot embed the BDH instances into the static secret keys of the test session owner and its peer, and the master secret key. However, the simulator can still embed g^u into the ephemeral public key X and g^v into the ephemeral public key Y because $\mathcal{A}$ cannot reveal x and y for U_A and U_B. Thus, the simulation successfully works, and the simulator can obtain $e(g,g)^{uvw}$ by computing $e(g^w, \sigma_3) = e(g^w, g^{xy})$. This is the reason why our protocol needs to exchange X and Y as well as $\{T_i^x\}_{i \in \delta_A}$ and $\{T_i^x\}_{i \in \delta_B}$ and σ_3 is needed.

4. Proposed Two-Pass Session-Policy Attribute-Based AKE Protocol

We constructed a session-policy attribute-based AKE protocol (SP-AB-AKE) based on attribute-based encryption schemes [22]. By applying the NAXOS technique [15], the proposed protocol can satisfy the predicate-based eCK security, under the gap Bilinear Diffie-Hellman (BDH) assumption [1] in the random oracle model.

4.1 Linear Secret Sharing

We introduce the notion of the access structure to represent the access control by the policy. We show the definition given in [2].

Definition 5 (Access Structure [2]): Let $\{P_1, P_2, \ldots, P_n\}$ be a set of parties. A collection $\mathbb{A} \subseteq 2^{\{P_1, P_2, \ldots, P_n\}}$ is monotone if $\forall Att_1, Att_2$: if $Att_1 \in \mathbb{A}$ and $Att_1 \subseteq Att_2$ then $Att_2 \in \mathbb{A}$. An access structure (resp. monotone access structure) is a collection (resp. monotone collection) $\mathbb{A}$ of non-empty subsets of $\{P_1, P_2, \ldots, P_n\}$, i.e., $\mathbb{A} \subseteq 2^{\{P_1, P_2, \ldots, P_n\}} \backslash \{\emptyset\}$. The sets in $\mathbb{A}$ are called the authorized sets, and the sets not in $\mathbb{A}$ are called the unauthorized sets.

Though this definition restricts monotone access structures, it is also possible to (inefficiently) realize general access structures by having the not of an attribute as a separate attribute altogether. Thus, the number of attributes in the system will be doubled.

We use linear secret sharing schemes (LSSSs) to obtain the fine-grained access control. The LSSS can provide arbitrary conditions for the reconstruction of the secret with monotone access structures. We show the definition given in [2].

Definition 6 (Linear Secret Sharing Schemes [2]): A secret sharing scheme Σ over a set of parties $\mathbb{P}$ is called linear (over $\mathbb{Z}_p$) if

1. The shares for each party form a vector over $\mathbb{Z}_p$.
2. There exists a matrix an M with ℓ rows and n columns called the share-generating matrix for Σ. For all $i = 1, \ldots, \ell$, the ith row of M we let the function ρ defined the party labeling row i as $\rho(i)$. When we consider the column vector $v = (s, r_2, \ldots, r_n)$, where $s \in \mathbb{Z}_p$ is the secret to be shared, and $r_2, \ldots, r_n \in \mathbb{Z}_p$ are randomly chosen, then Mv is the vector of ℓ shares of the secret s according to Σ. The share $(Mv)_i$ belongs to party $\rho(i)$.

The important property of LSSSs is the linear reconstruction property, defined as follows: Suppose that Σ is an LSSS for the access structure $\mathbb{A}$. Let $S \in \mathbb{A}$ be any authorized set, and let $I \subset \{1, 2, \ldots \ell\}$ be defined as $I = \{i :$

$\rho(i) \in S\}$. Then, there exist constants $\{w_i \in \mathbb{Z}_p\}_{i\in I}$ such that, if $\{\lambda_i\}$ are valid shares of any secret s according to Σ, then $\sum_{i\in I} w_i\lambda_i = s$. In [2], it is shown that these constants $\{w_i\}$ can be found in time polynomial in the size of the share generating matrix M.

We note that we use the convention that vector $(1, 0, 0, \dots, 0)$ is the "target" vector for any linear secret sharing scheme. For any satisfying set of rows I in M, we will have that the target vector is in the span of I. For any unauthorized set of rows I the target vector is not in the span of the rows of the set I. Moreover, there will exist a vector w such that $w \cdot (1, 0, 0, \dots, 0) = -1$ and $w \cdot M_i = 0$ for all $i \in I$.

4.2 Proposed Two-Pass Session-Policy Attribute-Based AKE Protocol

In this section, we describe the proposed key-policy attribute-based AKE protocol (SP-AB-AKE). In the protocol, a predicate $\phi(\gamma, \delta)$ is implemented using linear secret sharing scheme (LSSS), i.e., the first input γ of predicate ϕ corresponds to a subset of the attribute set, the second input δ of predicate ϕ corresponds to an access structure on the attribute set, and the output of predicate ϕ corresponds to whether the subset γ of the attribute set satisfies the access structure δ on the attribute set.

Our SP-AB-AKE scheme allows fine-grained access structure and large universe of attributes. Expressiveness of access structures is due to the direct application of LSSSs for the access control same as the Waters ABE. Our construction is also parameterized by n_{max} which specifies the maximum number of columns in share-generating matrices corresponding to access structures.

Parameters:
For input a security parameter κ, choose p, G, G_T, g and g_T such that bilinear groups with pairing $e : G \times G \to G_T$ of order κ-bit prime p with generators g and g_T, respectively. $H_1 : \{0,1\}^* \to G$, $H_2 : \{0,1\}^* \to \mathbb{Z}_p$ and $H_3 : \{0,1\}^* \to \{0,1\}^\kappa$ are hash functions modeled as random oracles.

Key Generation:
Output a master public key $MPK := (g, g^r, g_T^z)$ and a master secret key $MSK := g^z$ such that $r, z \in_R \mathbb{Z}_p$.

Key Extraction:
For input a set of attributes $\mathbb{S}_P$ from a party P, choose $t_{P_1}, \dots, t_{P_{n_{max}}} \in \mathbb{Z}_p$, and compute $S'_P = g^z g^{rt_{P_1}}$, $T_{P_j} = g^{t_{P_j}}$ for $1 \le j \le n_{max}$ (let $\{T_P\}$ denote the set of T_{P_j} for $1 \le j \le n_{max}$) and $S_{P_k} = \prod_{1\le j\le n_{max}} H_1(j,k)^{t_{P_j}}$ for $k \in \mathbb{S}_P$ (let $\{S_P\}$ denote the set of S_{P_k} for $k \in \mathbb{S}_P$). Then, output a static secret key $SK_P := (S'_P, \{T_P\}, \{S_P\})$.

Key Exchange:
We suppose that the party A is the session initiator and the party B is the session responder. A has the static secret key $SK_A = (S'_A, \{T_A\}, \{S_A\})$ corresponding to the set of his attributes $\mathbb{S}_A$ and B has the static secret key $SK_B = (S'_B, \{T_B\}, \{S_B\})$ corresponding to the set of his attributes $\mathbb{S}_B$. Then, A sends to B the ephemeral public key EPK_A corresponding to the access structure $\mathbb{A}_A$, and B sends to A the ephemeral public key EPK_B corresponding to the access structure $\mathbb{A}_B$. Finally, both parties A and B compute the shared key K if and only if the set of attributes $\mathbb{S}_A$ satisfies the access structure $\mathbb{A}_B$ and the set of attributes $\mathbb{S}_B$ satisfies the access structure $\mathbb{A}_A$.

1. First, A decides an access structure $\mathbb{A}_A$ which he hopes that the set of attributes $\mathbb{S}_B$ of B satisfies $\mathbb{A}_A$. Then, A derives the $\ell_A \times n_A$ share-generating matrix M_A and the injective labeling function ρ_A in a LSSS for $\mathbb{A}_A$. A chooses at random the ephemeral secret key $\tilde{u}_1, \dots, \tilde{u}_{n_A} \in_R \mathbb{Z}_p$. Then, A computes $u_j = H_2(S'_A, \{T_A\}, \{S_A\}, \tilde{u_j})$ for $1 \le j \le n_A$ and $X = g^{u_1}$. Also, A computes $U_{i,j} = g^{rM_{A_{i,j}}u_j} H_1(j, \rho_A(i))^{-u_1}$ for $1 \le i \le \ell_A$ and $1 \le j \le n_A$, and $U_{i,j} = H_1(j, \rho_A(i))^{-u_1}$ for $1 \le i \le \ell_A$ and $n_{A+1} \le j \le n_{max}$ (let $\{U\}$ denote the set of $U_{i,j}$ for $1 \le i \le \ell_A$ and $1 \le j \le n_{max}$). A sends $EPK_A := (X, \{U\})$, M_A and ρ_A to B, and erases $u_1, \dots, u_{n_A}$.
2. Upon receiving EPK_A, B checks whether the set of his attributes $\mathbb{S}_B$ satisfies the access structure M_A and ρ_A, and $X, \{U\} \in G$ holds. If not, B aborts. Otherwise, B decides an access structure $\mathbb{A}_B$ which he hopes that the set of attributes $\mathbb{S}_A$ of A satisfies $\mathbb{A}_B$. Then, B derives the $\ell_B \times n_B$ share-generating matrix M_B and the labeling function ρ_B in an LSSS for $\mathbb{A}_B$. B chooses at random the ephemeral secret key $\tilde{v}_1, \dots, \tilde{v}_{n_B} \in_R \mathbb{Z}_p$. Then, B computes $v_j = H_2(S'_B, \{T_B\}, \{S_B\}, \tilde{v_j})$ for $1 \le j \le n_B$ and $Y = g^{v_1}$. Also, B computes $V_{i,j} = g^{rM_{B_{i,j}}v_j} H_1(j, \rho_B(i))^{-v_1}$ for $1 \le i \le \ell_B$ and $1 \le j \le n_B$, and $V_{i,j} = H_1(j, \rho_B(i))^{-v_1}$ for $1 \le i \le \ell_B$ and $n_{B+1} \le j \le n_{max}$ (let $\{V\}$ denote the set of $V_{i,j}$ for $1 \le i \le \ell_B$ and $1 \le j \le n_{max}$). B sends $EPK_B := (Y, \{V\})$, M_B and ρ_B to A.
   B computes the shared secrets as follows: We suppose that $\mathbb{S}_B$ satisfies M_A and ρ_A, and let $I_B \subset \{1, 2, \dots, \ell_A\}$ be defined as $I_B = \{i : \rho_A(i) \in \mathbb{S}_B\}$. Then, B can efficiently find $\{w_{B_i} \in \mathbb{Z}_p\}_{i\in I_B}$ such that $\sum_{i\in I_B} w_{B_i}\lambda_i = s$ for valid shares $\{\lambda_i\}$ of any secret s according to M_A†. Note that, if $\mathbb{S}_B$ does not satisfy M_A and ρ_A, B cannot find all w_{B_i} for $i \in I_B$ from the property of LSSSs.
   Then, B sets the shared secrets
   $$\sigma_1 = e(X, S'_B) / \Big(\prod_{1\le j\le n_{max}} e(T_{B_j}, \prod_{i\in I_B} U_{i,j}^{w_{B_i}})\Big) \prod_{i\in I_B} e(X, S_{B_{\rho_A(i)}}^{w_{B_i}}),$$
   $$\sigma_2 = (g_T^z)^{v_1}, \quad \sigma_3 = X^{v_1}$$
   and the session key $K = H_3(\sigma_1, \sigma_2, \sigma_3, (X, \{U\}, M_A, \rho_A), (Y, \{V\}, M_B, \rho_B))$. B completes the session with the session key K, and erases $v_1, \dots, v_{n_B}$.
3. Upon receiving EPK_B, A checks whether the set of his attributes $\mathbb{S}_A$ satisfies the access structure M_B and ρ_B, and $Y, \{V\} \in G$ holds. If not, A aborts. Otherwise, A computes the shared secrets as follows: We suppose

†In this case, the secret corresponds to u_1 and shares correspond to $\{M_{A_{i,j}} u_j\}$.

that $\mathbb{S}_A$ satisfies M_B and ρ_B, and let $I_A \subset \{1, 2, \ldots, \ell_B\}$ be defined as $I_A = \{i : \rho_B(i) \in \mathbb{S}_A\}$. Then, A can efficiently find $\{w_{A_i} \in \mathbb{Z}_p\}_{i \in I_A}$ such that $\sum_{i \in I_A} w_{A_i} \lambda_i = s$ for valid shares $\{\lambda_i\}$ of any secret s according to M_B†. Note that, if $\mathbb{S}_A$ does not satisfy M_B and ρ_B, A cannot find all w_{A_i} for $i \in I_A$ from the property of LSSSs.
Then, A sets the shared secrets

$$\sigma_2 = e(Y, S'_A) / \left(\prod_{1 \le j \le n_{max}} e(T_{A_j}, \prod_{i \in I_A} V_{i,j}^{w_{A_i}}) \right) \prod_{i \in I_A} e(Y, S_{A_{\rho_B(i)}}^{w_{A_i}}),$$
$$\sigma_1 = (g_T^z)^{H_2(S'_A, \{T_A\}, \{S_A\}, \tilde{u_1})},$$
$$\sigma_3 = Y^{H_2(S'_A, \{T_A\}, \{S_A\}, \tilde{u_1})}$$

and the session key $K = H_3(\sigma_1, \sigma_2, \sigma_3, (X, \{U\}, M_A, \rho_A), (Y, \{V\}, M_B, \rho_B))$. A completes the session with the session key K.

The shared secrets that both parties compute are

$$\begin{aligned}
\sigma_1 &= e(X, S'_B) / \left(\prod_{1 \le j \le n_{max}} e(T_{B_j}, \prod_{i \in I_B} U_{i,j}^{w_{B_i}}) \right) \prod_{i \in I_B} e(X, S_{B_{\rho_A(i)}}^{w_{B_i}}) \\
&= e(X, S'_B) / \prod_{1 \le j \le n_A} e(g^{t_{B_j}}, g^{\sum_{i \in I_B} r M_{A_{i,j}} u_j w_{B_i}}) \\
&\quad \cdot \prod_{1 \le j \le n_{max}} e(g^{t_{B_j}}, \prod_{i \in I_B} H_1(j, \rho_A(i)^{-u_1 w_{B_i}})) \\
&\quad \cdot \prod_{i \in I_B} e(g^{u_1}, \prod_{1 \le j \le n_{max}} H_1(j, \rho_A(i))^{t_{B_j} w_{B_i}}) \\
&= e(X, S'_B) / \prod_{1 \le j \le n_A} e(g^{t_{B_j}}, g^{\sum_{i \in I_B} r M_{A_{i,j}} u_j w_{B_i}}) \\
&= e(g^{u_1}, g^z g^{r t_{B_1}}) / e(g^{t_{B_1}}, g^{\sum_{i \in I_B} r M_{A_{i,1}} u_1 w_{B_i}}) \\
&= g_T^{u_1(z + r t_{B_1})} / g_T^{u_1 r t_{B_1}} \\
&= g_T^{z u_1} (= (g_T^z)^{u_1}), \\
\sigma_2 &= e(Y, S'_A) / \left(\prod_{1 \le j \le n_{max}} e(T_{A_j}, \prod_{i \in I_A} V_{i,j}^{w_{A_i}}) \right) \prod_{i \in I_A} e(S_{A_{\rho_B(i)}}^{w_{A_i}}, Y), \\
&= e(Y, S'_A) / \prod_{1 \le j \le n_B} e(g^{t_{A_j}}, g^{\sum_{i \in I_A} r M_{B_{i,j}} v_j w_{A_i}}) \\
&\quad \cdot \prod_{1 \le j \le n_{max}} e(g^{t_{A_j}}, \prod_{i \in I_A} H_1(j, \rho_B(i)^{-v_1 w_{A_i}})) \\
&\quad \cdot \prod_{i \in I_A} e(g^{v_1}, \prod_{1 \le j \le n_{max}} H_1(j, \rho_B(i))^{t_{A_j} w_{A_i}}) \\
&= g_T^{z v_1} (= (g_T^z)^{v_1}), \\
\sigma_3 &= X^{v_1} = Y^{u_1} = g^{u_1 v_1},
\end{aligned}$$

and therefore they can compute the same session key K.

We construct our SP-AB-AKE scheme by combining modification of the Waters ABE scheme of Section 5 of [22] and the NAXOS technique [15]. From the structure of the ABE scheme, the ephemeral secret key needs to contain n elements. Thus, we apply the NAXOS technique to each element. Specifically, we convert $(\tilde{u}_1, \ldots, \tilde{u}_{n_A})$ and $(\tilde{v}_1, \ldots, \tilde{v}_{n_B})$ into $(u_1, \ldots, u_{n_A})$ and $(v_1, \ldots, v_{n_B})$, respectively, by using the random oracle H_2.

The decryption algorithm of the ABE scheme can be applied to the derivation of the session key. In the process of the decryption, the decryption algorithm computes $g_T^{t_1 z}$ where t_1 is the randomness in the encryption and z is the secret in the setup. In our SP-AB-AKE, we use $\sigma_1 = g_T^{u_1 z}$ and $\sigma_2 = g_T^{v_1 z}$ as a part of the seed of the session key where u_1 and v_1 are derived from the ephemeral secret keys of A and B respectively. However, only $g_T^{u_1 z}$ and $g_T^{v_1 z}$ are not enough to achieve the security in the predicate-based eCK model in Sect. 2. The predicate-based eCK model allows the adversary to reveal the master secret key. We cannot prove the security in such a case because the simulator cannot embed the BDH instance to the master secret key and cannot extract information to obtain the answer of the gap BDH problem from only $g_T^{u_1 z}$ and $g_T^{v_1 z}$. Thus, we add $\sigma_3 = g^{u_1 v_1}$ to the seed of the session key in order to simulate such a case.

Note that, it would be possible to modify our SP-AB-AKE scheme to be secure under the BDH assumption by using the twin DH technique [8]. However, this modification may bring about more keys, more shared values or much computation, and, thus, it would not be suitable to construct efficient schemes.

4.3 Security

The proposed session-policy attribute-based AKE protocol is selective-condition secure in the predicate-based eCK security model under the gap BDH assumption and in the random oracle model.

Theorem 7: If G is a group, where the gap BDH assumption holds and H_1, H_2 and H_3 are random oracles, the proposed session-policy attribute-based AKE protocol is selective-condition secure in the predicate-based eCK model described in Sect. 2.

Proof of Theorem 7 is provided in Appendix B.

5. Conclusion

We firstly defined the eCK security model, which captures leakage of ephemeral secret key, for predicate-based AKE by extending the eCK security model for (PKI-based) AKE. We also proposed eCK secure two-pass key-policy and session-policy attribute-based AKE protocols based on attribute-based encryptions, using the NAXOS technique. Our proposed protocols are selective-condition secure in the predicate-based eCK security model under the gap BDH assumption and in the random oracle model.

References

[1] J. Baek, R. Safavi-Naini, and W. Susilo, "Efficient multi-receiver identity-based encryption and its application to broadcast encryption," Public Key Cryptography 2005, pp.380–397, 2005.

[2] A. Beimel, "Secure schemes for secret sharing and key distribution," PhD thesis, Israel Institute of Technology, Technion, 1996.

† In this case, the secret corresponds to v_1 and shares correspond to $\{M_{B_{i,j}} v_j\}$.

[3] M. Bellare and P. Rogaway, "Entity authentication and key distribution," CRYPTO 1993, pp.232–249, 1993.
[4] J. Birkett and D. Stebila, "Predicate-based key exchange," ACISP 2010, pp.282–299, 2010.
[5] S. Blake-Wilson, D. Johnson, and A. Menezes, "Key agreement protocols and their security analysis," 6th IMA International Conference, pp.30–45, 1997.
[6] C. Boyd and K.-K.R. Choo, "Security of two-party identity-based key agreement," Mycrypt 2005, pp.229–243, 2005.
[7] R. Canetti and H. Krawczyk, "Analysis of key-exchange protocols and their use for building secure channels," EUROCRYPT 2001, pp.453–474, 2001.
[8] D. Cash, E. Kiltz, and V. Shoup, "The twin Diffie-Hellman problem and applications," J. Cryptology, vol.22, no.4, pp.470–504, 2009.
[9] L. Chen, Z. Cheng, and N.P. Smart, "Identity-based key agreement protocols from pairings," Int. J. Information Security, vol.6, no.4, pp.213–241, 2007.
[10] A. Fujioka, K. Suzuki, and B. Ustaoglu, "Ephemeral key leakage resilient and efficient ID-AKEs that can share identities, private and master keys," Pairing 2010, pp.187–205, 2010.
[11] A. Fujioka, K. Suzuki, and K. Yoneyama, "Predicate-based authenticated key exchange resilient to ephemeral key leakage," WISA 2010, pp.15–30, 2010.
[12] M.C. Gorantla, C. Boyd, and J.M. González Nieto, "Attribute-based authenticated key exchange," ACISP 2010, pp.300–317, 2010.
[13] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," ACM Conference on Computer and Communications Security 2006, pp.89–98, 2006.
[14] H. Huang and Z. Cao, "An ID-based authenticated key exchange protocol based on bilinear Diffie-Hellman problem," ASIACCS 2009, pp.333–342, 2009.
[15] B.A. LaMacchia, K. Lauter, and A. Mityagin, "Stronger security of authenticated key exchange," ProvSec 2007, pp.1–16, 2007.
[16] A. Sahai and B. Waters, "Fuzzy identity-based encryption," EUROCRYPT 2005, pp.457–473, 2005.
[17] N.P. Smart, "Identity-based authenticated key agreement protocol based on Weil pairing," Electron. Lett., vol.38, no.13, pp.630–632, 2002.
[18] R. Steinwandt and A. Suárez Corona, "Attribute-based group key establishment," Advances in Mathematics of Communications, vol.4, no.3, pp.381–398, 2010.
[19] H. Wang, Q. Xu, and T. Ban, "A provably secure two-party attribute-based key agreement protocol," Fifth International Conference on Intelligent Information Hiding and Multimedia Signal Processing, pp.1042–1045, 2009.
[20] H. Wang, Q. Xu, and X. Fu, Revocable attribute-based key agreement protocol without random oracles, J. Networks, vol.4, no.8, pp.787–794, 2009.
[21] H. Wang, Q. Xu, and X. Fu, "Two-party attribute-based key agreement protocol in the standard model," ISIP 2009, pp.325–328, 2009.
[22] B. Waters. "Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization," Public Key Cryptography 2011, pp.53–70, 2011.
[23] K. Yoneyama, "Strongly secure two-pass attribute-based authenticated key exchange," Pairing 2010, pp.147–166, 2010.

Appendix A: Proof of Theorem 4

We denote $\mathrm{BDH}(U, V, W) = e(g, g)^{\log U \log V \log W}$, and the DBDH oracle on input $(g^u, g^v, g^w, e(g, g)^x)$ returns bit 1 if $uvw = x$, and bit 0 otherwise. We need the gap Bilinear Diffie-Hellman (BDH) assumption, where one tries to compute $\mathrm{BDH}(U, V, W)$ accessing the DBDH oracle.

We show that if polynomially bounded adversary $\mathcal{A}$ can distinguish the session key of a fresh session from a randomly chosen session key, we can solve the gap BDH problem. Let κ denote the security parameter, and let $\mathcal{A}$ be a polynomially (in κ) bounded adversary. We use $\mathcal{A}$ to construct a gap BDH solver $\mathcal{S}$ that succeeds with non-negligible probability. Adversary $\mathcal{A}$ is said to be successful with non-negligible probability if $\mathcal{A}$ wins the distinguishing game with probability $\frac{1}{2}+f(\kappa)$, where $f(\kappa)$ is non-negligible, and the event M denotes a successful $\mathcal{A}$.

Let the test session be $\mathtt{sid}^* = (\Pi, \mathcal{I}, \gamma_A, \gamma_B, \hat{\delta}_A, \hat{\delta}_B)$ or $(\Pi, \mathcal{R}, \gamma_A, \gamma_B, \hat{\delta}_B, \hat{\delta}_A)$ that is a completed session between honest user U_A with string γ_A and U_B with string γ_B, where users U_A and U_B are the initiator and responder of the test session $\mathtt{sid}^*$. We denote the message sent from U_A to U_B by $\hat{\delta}_A$ and the message sent from U_B to U_A by $\hat{\delta}_B$. Let H^* be the event in which $\mathcal{A}$ queries $(\sigma_1, \sigma_2, \sigma_3, \Pi, \hat{\delta}_A, \hat{\delta}_B)$ to H. Let $\overline{H^*}$ be the complement of event H^*. Let $\mathtt{sid}$ be any completed session owned by an honest user such that $\mathtt{sid} \neq \mathtt{sid}^*$ and $\mathtt{sid}$ is non-matching to $\mathtt{sid}^*$. Since $\mathtt{sid}$ and $\mathtt{sid}^*$ are distinct and non-matching, the inputs to the key derivation function H are different for $\mathtt{sid}$ and $\mathtt{sid}^*$. Since H is a random oracle, $\mathcal{A}$ cannot obtain any information about the test session key from the session keys of non-matching sessions. Hence, $\Pr(M \wedge \overline{H^*}) \leq \frac{1}{2}$ and $\Pr(M) = \Pr(M \wedge H^*) + \Pr(M \wedge \overline{H^*}) \leq \Pr(M \wedge H^*) + \frac{1}{2}$, whence $f(\kappa) \leq \Pr(M \wedge H^*)$. Henceforth, the event $M \wedge H^*$ is denoted by M^*.

We denote the master secret and public keys by $z, Z = g^z$. For user U_i, we denote the string by γ_i, the static secret key by $D_i = \{D_u\}_{u \in L(\gamma_i)}$, the ephemeral secret key by $\tilde{x}_i$, and the exponent of the ephemeral public key by $x_i = H'(\{D_u\}_{u \in L(\gamma_i)}, \tilde{x}_i)$. We also denote the session key by K. Assume that $\mathcal{A}$ succeeds in an environment with n users and activates at most s sessions within a user.

We consider the following events.

- Let D be the event in which $\mathcal{A}$ queries the static secret key $\{D_u\}_{u \in L(\gamma)}$ to H', *before* asking StaticKeyReveal queries or the MasterKeyReveal query or *without* asking StaticKeyReveal queries or the MasterKeyReveal query.
- Let $\overline{D}$ be the complement of event D.

We consider the following events, which cover all cases of adversary $\mathcal{A}$'s behavior.

- Let E_1 be the event in which test session $\mathtt{sid}^*$ has no matching session $\overline{\mathtt{sid}^*}$ and $\mathcal{A}$ queries StaticKeyReveal(γ) s.t. $\phi(\gamma, \delta_B) = 1$.
- Let E_2 be the event in which test session $\mathtt{sid}^*$ has no matching session $\overline{\mathtt{sid}^*}$ and $\mathcal{A}$ queries EphemeralKeyReveal$(\mathtt{sid}^*)$.
- Let E_3 be the event in which test session $\mathtt{sid}^*$ has matching session $\overline{\mathtt{sid}^*}$ and $\mathcal{A}$ queries MasterKeyReveal() or queries StaticKeyReveal(γ) s.t. $\phi(\gamma, \delta_B) = 1$ and StaticKeyReveal(γ) s.t. $\phi(\gamma, \delta_A) = 1$.
- Let E_4 be the event in which test session $\mathtt{sid}^*$ has matching session $\overline{\mathtt{sid}^*}$ and $\mathcal{A}$ queries

EphemeralKeyReveal($\mathtt{sid}^*$) and EphemeralKeyReveal ($\overline{\mathtt{sid}^*}$).

- Let E_5 be the event in which test session $\mathtt{sid}^*$ has matching session $\overline{\mathtt{sid}^*}$ and $\mathcal{A}$ queries StaticKeyReveal(γ) s.t. $\phi(\gamma, \delta_B) = 1$ and EphemeralKeyReveal($\overline{\mathtt{sid}^*}$).
- Let E_6 be the event in which test session $\mathtt{sid}^*$ has matching session $\overline{\mathtt{sid}^*}$ and $\mathcal{A}$ queries EphemeralKeyReveal($\mathtt{sid}^*$) and StaticKeyReveal(γ) s.t. $\phi(\gamma, \delta_A) = 1$.

To finish the proof, we investigate events $D \wedge M^*$, $E_i \wedge \overline{D} \wedge M^*$ ($i = 1, ..., 6$), which cover all cases of event M^*, in the following.

A.1 Event $D \wedge M^*$

In event D, $\mathcal{A}$ queries static secret key $\{D_u\}_{u \in L(\gamma)}$ to H', before asking StaticKeyReveal queries or MasterKeyReveal query or without asking StaticKeyReveal queries or MasterKeyReveal query. We embed the instance as $Z = e(g, g)^z = e(U, V)$ and extract $g^z = g^{uv}$ from $\{D_u\}_{u \in L(\gamma_i)}$. Due to the page limitation, details of this case is provided in the full paper version of this parer.

In the case of event $D \wedge M^*$, $\mathcal{S}$ performs the following steps.

A.1.1 Setup

The gap BDH solver $\mathcal{S}$ begins by establishing n honest users that are assigned random static key pairs. In addition to the above steps, $\mathcal{S}$ embeds instance $(U = g^u, V = g^v, W = g^w)$ of gap BDH problem as follows.

Let δ_A, δ_B be the target conditions selected by $\mathcal{A}$. $\mathcal{S}$ sets $Z = e(U, V)$, selects $t_i \in_U \mathbb{Z}_q$ and sets $T_i = g^{t_i}$.

The algorithm $\mathcal{S}$ activates $\mathcal{A}$ on this set of users and awaits the actions of $\mathcal{A}$. We next describe the actions of $\mathcal{S}$ in response to user activations and oracle queries.

A.1.2 Simulation

The solver $\mathcal{S}$ simulate oracle queries as follows. $\mathcal{S}$ maintains list L_H that contains queries and answers of H oracle, and list L_S that contains queries and answers of SessionKeyReveal,

1. Send(Π, γ_i, γ_j): $\mathcal{S}$ selects a string δ_i, picks ephemeral secret key $x \in_U \mathbb{Z}_q$, computes ephemeral public key $\hat{\delta}_i$ honestly, records $(\Pi, \gamma_i, \gamma_j, \hat{\delta}_i)$, and returns it.
2. Send($\Pi, \gamma_i, \gamma_j, \hat{\delta}_i$): $\mathcal{S}$ selects a string δ_j, picks ephemeral secret key $y \in_U \mathbb{Z}_q$, computes ephemeral public key $\hat{\delta}_j$ honestly, records $(\Pi, \gamma_i, \gamma_j, \hat{\delta}_i, \hat{\delta}_j)$. and returns it.
3. Send($\Pi, \gamma_i, \gamma_j, \hat{\delta}_i, \hat{\delta}_j$): If $(\Pi, \gamma_i, \gamma_j, \hat{\delta}_i)$ is not recorded, $\mathcal{S}$ records the session $(\Pi, \gamma_i, \gamma_j, \hat{\delta}_i, \hat{\delta}_j)$ is not completed. Otherwise, $\mathcal{S}$ records the session is completed.
4. $H(\sigma_1, \sigma_2, \sigma_3, \Pi, \hat{\delta}_i, \hat{\delta}_j)$:
   a. If $(\sigma_1, \sigma_2, \sigma_3, \Pi, \hat{\delta}_i, \hat{\delta}_j)$ is recorded in list L_H, then return recorded value K.
   b. Else if $(\Pi, \hat{\delta}_i, \hat{\delta}_j)$ is recorded in list L_S, $\mathrm{DBDH}(X, U, V, \sigma_1) = 1$, $\mathrm{DBDH}(Y, U, V, \sigma_2) = 1$, and $e(X, Y) = e(g, \sigma_3)$, then return recorded value K and record it in list L_H.
   c. Otherwise, $\mathcal{S}$ returns random value K and record it in list L_H.
5. SessionKeyReveal($\mathtt{sid}$):
   a. If the session $\mathtt{sid}$ is not completed, return error.
   b. Else if $\mathtt{sid}$ is recorded in list L_S, then return recorded value K.
   c. Else if $(\sigma_1, \sigma_2, \sigma_3, \Pi, \hat{\delta}_i, \hat{\delta}_j)$ is recorded in list L_H, $\mathrm{DBDH}(X, U, V, \sigma_1) = 1$, $\mathrm{DBDH}(Y, U, V, \sigma_2) = 1$, and $e(X, Y) = e(g, \sigma_3)$, then return recorded value K and record it in list L_S.
   d. Otherwise, $\mathcal{S}$ returns random value K and record it in list L_S.
6. $H'(\{D_u\}_{u \in L(\gamma_i)}, \tilde{x})$: $\mathcal{S}$ computes $g^z = g^{uv}$ from $\{D_u\}_{u \in L(\gamma_i)}$ by iterated polynomial interpolation, then $\mathcal{S}$ stops and is successful by outputting answer of gap BDH problem $e(g^z, W) = \mathrm{BDH}(U, V, W)$.
7. EphemeralKeyReveal($\mathtt{sid}$): $\mathcal{S}$ returns random value $\tilde{x}$ and record it.
8. StaticKeyReveal(γ_i): $\mathcal{S}$ aborts with failure.
9. MasterKeyReveal(): $\mathcal{S}$ aborts with failure.
10. EstablishParty(U_i, γ_i): $\mathcal{S}$ responds to the query faithfully.
11. Test($\mathtt{sid}$): $\mathcal{S}$ responds to the query faithfully.
12. If $\mathcal{A}$ outputs a guess γ, $\mathcal{S}$ aborts with failure.

A.1.3 Analysis

The simulation of $\mathcal{A}$ environment is perfect except with negligible probability.

Suppose event D occurs, $\mathcal{S}$ does not abort in Step 8 and Step 9 since StaticKeyReveal and MasterKeyReveal are not queried.

Under event M^*, $\mathcal{A}$ queries correctly formed $\sigma_1, \sigma_2, \sigma_3$ to H. Therefore, $\mathcal{S}$ is successful as described in Step 6 and does not abort as in Step 12.

Hence, $\mathcal{S}$ is successful with probability $Pr(S) \geq p_D$, where p_D is probability that $D \wedge M^*$ occurs.

A.2 Event $E_1 \wedge \overline{D} \wedge M^*$

Here, we only consider the event $E_1 \wedge \overline{D} \wedge M^*$, due to the page limitation. Details of the other cases is provided in the full paper version of this parer.

In event E_1, test session $\mathtt{sid}^*$ has no matching session $\overline{\mathtt{sid}^*}$, and $\mathcal{A}$ queries StaticKeyReveal(γ) s.t. $\phi(\gamma, \delta_B) = 1$, and $\mathcal{A}$ does not query EphemeralKeyReveal($\mathtt{sid}^*$) and StaticKeyReveal(γ) s.t. $\phi(\gamma, \delta_A) = 1$ by the condition of freshness. We embed the instance as $Z = e(g, g)^z = e(U, V)$, $T_i^x = W^{\beta_i}$ where $T_i = g^{\beta_i}$, and extract $e(g, g)^{uvw}$ from

$\sigma_1 = e(g, g)^{xz}$. In event $E_1 \wedge \overline{D} \wedge M^*$, $\mathcal{S}$ performs the following steps.

A.2.1 Setup

The gap BDH solver $\mathcal{S}$ begins by establishing n honest users that are assigned random static key pairs. In addition to the above steps, $\mathcal{S}$ embeds instance $(U = g^u, V = g^v, W = g^w)$ of the gap BDH problem as follows.

Let δ_A and δ_B be the target conditions selected by $\mathcal{A}$. $\mathcal{S}$ sets the public master key as $Z = e(U, V) = g_T^{uv}$, $T_i = g^{\beta_i}$ for $i \in \delta_A$, and $T_i = V^{\beta_i} = g^{v\beta_i}$ for $i \notin \delta_A$, where $\mathcal{S}$ selects $\beta_i \in_U \mathbb{Z}_q$ randomly.

$\mathcal{S}$ randomly selects two users U_A and U_B and integers $j_A \in_R [1, s]$, that becomes a guess of the test session with probability $1/n^2 s$. $\mathcal{S}$ sets the ephemeral public key of j_A-th session of user U_A as $T_i^x = W^{\beta_i} = g^{w\beta_i}$ for $i \in \delta_A$ and $X = W$.

The solver $\mathcal{S}$ activates $\mathcal{A}$ on this set of users and awaits the actions of $\mathcal{A}$. We next describe the actions of $\mathcal{S}$ in response to user activations and oracle queries.

A.2.2 Simulation

The solver $\mathcal{S}$ simulates oracle queries as follows. $\mathcal{S}$ maintains list L_H that contains queries and answers of H oracle, and list L_S that contains queries and answers of SessionKeyReveal,

1. Send$(\Pi, \gamma_i, \gamma_j)$: $\mathcal{S}$ selects a string δ_i, picks ephemeral secret key $x \in_U \mathbb{Z}_q$, honestly computes ephemeral public key $\hat{\delta}_i$, records $(\Pi, \gamma_i, \gamma_j, \hat{\delta}_i)$, and returns it.
2. Send$(\Pi, \gamma_i, \gamma_j, \hat{\delta}_i)$: $\mathcal{S}$ selects a string δ_j, picks ephemeral secret key $y \in_U \mathbb{Z}_q$, honestly computes ephemeral public key $\hat{\delta}_j$, records $(\Pi, \gamma_i, \gamma_j, \hat{\delta}_i, \hat{\delta}_j)$, and returns it.
3. Send$(\Pi, \gamma_i, \gamma_j, \hat{\delta}_i, \hat{\delta}_j)$: If $(\Pi, \gamma_i, \gamma_j, \hat{\delta}_i)$ is not recorded, $\mathcal{S}$ records the session $(\Pi, \gamma_i, \gamma_j, \hat{\delta}_i, \hat{\delta}_j)$ as not completed. Otherwise, $\mathcal{S}$ records the session as completed.
4. $H(\sigma_1, \sigma_2, \sigma_3, \Pi, \hat{\delta}_i, \hat{\delta}_j)$:
   a. If $(\sigma_1, \sigma_2, \sigma_3, \Pi, \hat{\delta}_i, \hat{\delta}_j)$ is recorded in list L_H, then return recorded value K.
   b. Else if $(\Pi, \hat{\delta}_i, \hat{\delta}_j)$ is recorded in list L_S, DBDH$(X, U, V, \sigma_1) = 1$, DBDH$(Y, U, V, \sigma_2) = 1$, and $e(X, Y) = e(g, \sigma_3)$, then return recorded value K and record it in list L_H.
   c. Else if DBDH$(X, U, V, \sigma_1) = 1$, DBDH$(Y, U, V, \sigma_2) = 1$, $e(X, Y) = e(g, \sigma_3)$, $i = A, j = B$, and the session is the j_A-th session of user U_A, then $\mathcal{S}$ stops and is successful by outputting the answer of the gap BDH problem instance $\sigma_1 =$ BDH(U, V, W).
   d. Otherwise, $\mathcal{S}$ returns random value K and records it in list L_H.
5. SessionKeyReveal(`sid`):
   a. If the session `sid` is not completed, return error.
   b. Else if `sid` is recorded in list L_S, then return recorded value K.
   c. Else if $(\sigma_1, \sigma_2, \sigma_3, \Pi, \hat{\delta}_i, \hat{\delta}_j)$ is recorded in list L_H, DBDH$(X, U, V, \sigma_1) = 1$, DBDH$(Y, U, V, \sigma_2) = 1$, and $e(X, Y) = e(g, \sigma_3)$, then return recorded value K and record it in list L_S.
   d. Otherwise, $\mathcal{S}$ returns random value K and records it in list L_S.
6. $H'(\{D_u\}_{u \in L(\gamma_i)}, \tilde{x})$: If $\{D_u\}_{u \in L(\gamma_i)}, \tilde{x}$ is used in the j_A-th session of user U_A, $\mathcal{S}$ aborts with failure. Otherwise, simulate this random oracle in the usual way.
7. EphemeralKeyReveal(`sid`): $\mathcal{S}$ returns random value $\tilde{x}$ and records it.
8. StaticKeyReveal(γ_i): First, we consider the case that γ_i is an access tree with height 1. By the condition $\phi(\gamma_i, \delta_A) \neq 1$, we have $|\gamma_i \cap \delta_A| < k$, where k is the threshold. Thus, we have a set Γ s.t. $\gamma_i \cap \delta_A \subset \Gamma \subset \delta_A$ and $|\Gamma| = k - 1$.
   For $u \in \Gamma$, $\mathcal{S}$ selects random $q(i)$ and sets $D_u = V^{q(i)/\beta_i} = g^{vq(i)/\beta_i} = g^{Q(i)/\beta_i}$, where $i = index(u)$. $\mathcal{S}$ also sets $g^{vq(0)} = g^{uv}$, and this implicitly defines degree $k - 1$ polynomial $q(x)$ and $Q(x) = vq(x)$.
   For $u \notin \delta_A$, $\mathcal{S}$ sets $D_u = [\prod_{j=index(v),\, v\in\Gamma} g^{q(j)\Delta_{j,\Gamma}(i)} U^{\Delta_{0,\Gamma}(i)}]^{1/\beta_i} = g^{q(i)/\beta_i} = g^{Q(i)/v\beta_i}$, where $i = index(u)$.
   Finally, $\mathcal{S}$ returns $\{D_u\}_{u \in L(\gamma_i)}$.
   If the height is more than 1, $\mathcal{S}$ creates $\{D_u\}_{u \in L(\gamma_i)}$ as follows. By the condition $\phi(\gamma_i, \delta_A) \neq 1$, attributes δ_A do not satisfy access tree γ_i, so the root node u_r is not satisfied, i.e., the number of satisfied nodes is less than threshold k_{u_r} of the root node u_r. Thus, we have a set Γ of nodes that contains all satisfied nodes and $|\Gamma| = k_{u_r} - 1$.
   For $u \in \Gamma$, $\mathcal{S}$ selects random $q(index(u))$ and assigns it to the node $u \in \Gamma$. $\mathcal{S}$ also sets $g^{q(0)} = D_{u_r} = g^{uv}$, and this implicitly defines degree $k_{u_r} - 1$ polynomial $q(x)$. For each node $u \in \Gamma$, $\mathcal{S}$ performs secret sharing iteratively, i.e., makes shares of $q(index(u))$ then makes shares of the share and so on, until reaching a leaf node.
   For $u \notin \Gamma$, $\mathcal{S}$ computes $D_u = g^{q(index(u))}$ and assigns it to the node $u \notin \Gamma$, which is unsatisfied by the definition of Γ.
   For each unsatisfied node $u \notin \Gamma$, $\mathcal{S}$ applies the above procedure recursively. Finally, $\mathcal{S}$ reaches unsatisfied node u that has leaf nodes as its children. Then $\mathcal{S}$ computes $\{D_u\}_{u \in L(\gamma_i)}$ the same as in the case that γ_i is an access tree with height 1.
9. MasterKeyReveal(): $\mathcal{S}$ aborts with failure.
10. EstablishParty(U_i, γ_i): $\mathcal{S}$ responds to the query faithfully.
11. Test(`sid`): If ephemeral public key X is not W in session `sid`, then $\mathcal{S}$ aborts with failure. Otherwise, respond to the query faithfully.
12. If $\mathcal{A}$ outputs a guess γ, $\mathcal{S}$ aborts with failure.

A.2.3 Analysis

The simulation of the adversary environment is perfect except with negligible probability. The probability that $\mathcal{A}$ selects the session, where ephemeral public key X is W, as the test session $\mathtt{sid}^*$ is at least $\frac{1}{n^2 s}$. Suppose this is indeed the case, $\mathcal{S}$ does not abort in Step 11.

Suppose event E_1 occurs, $\mathcal{S}$ does not abort in Step 9, since MasterKeyReveal() is not queried. Suppose event E_1 occurs, $\mathcal{S}$ does not abort in Step 6 except with negligible probability, since EphemeralKeyReveal($\mathtt{sid}^*$) is not queried.

Under event M^*, $\mathcal{A}$ queries correctly formed $\sigma_1, \sigma_2, \sigma_3$ to H. Therefore, $\mathcal{S}$ is successful as described in Step 4c and does not abort as in Step 12.

Hence, $\mathcal{S}$ is successful with probability $Pr(S) \geq \frac{p_1}{n^2 s}$, where p_1 is the probability that $E_1 \wedge \overline{D} \wedge M^*$ occurs. □

A.3 Event $E_3 \wedge \overline{D} \wedge M^*$

In event E_3, test session $\mathtt{sid}^*$ has matching session $\overline{\mathtt{sid}^*}$, and $\mathcal{A}$ queries MasterKeyReveal() or queries StaticKeyReveal(γ) s.t. $\phi(\gamma, \delta_B) = 1$ and StaticKeyReveal(γ) s.t. $\phi(\gamma, \delta_A) = 1$ and does not query EphemeralKeyReveal($\mathtt{sid}^*$) and EphemeralKeyReveal($\overline{\mathtt{sid}^*}$) due to the condition of freshness. We embed the instance as $X = g^x = U$, $Y = g^y = V$ and extract g^{uv} from $\sigma_3 = g^{xy}$. Due to the page limitation, details of this case is provided in the full paper version of this parer.

In the case of event $E_3 \wedge \overline{D} \wedge M^*$, $\mathcal{S}$ performs the following steps.

A.3.1 Setup

The gap BDH solver $\mathcal{S}$ begins by establishing n honest users that are assigned random static key pairs. In addition to the above steps, $\mathcal{S}$ embeds instance $(U = g^u, V = g^v, W = g^w)$ of gap BDH problem as follows.

Let δ_A, δ_B be the target conditions selected by $\mathcal{A}$. $\mathcal{S}$ selects $z \in_U \mathbb{Z}_q$ and sets $Z = g_T^z$, selects $t_i \in_U \mathbb{Z}_q$ and sets $T_i = g^{t_i}$.

$\mathcal{S}$ randomly selects two users U_A, U_B and integers $j_A, j_B \in_R [1, s]$, that becomes a guess of the test session with probability $1/n^2 s^2$. $\mathcal{S}$ sets ephemeral public key of j_A-th session of user U_A as $X = g^x = U$, $T_i^x = U^{t_i}$. $\mathcal{S}$ sets ephemeral public key of j_B-th session of user U_B as $Y = g^y = V$, $T_i^y = V^{t_i}$.

The algorithm $\mathcal{S}$ activates $\mathcal{A}$ on this set of users and awaits the actions of $\mathcal{A}$. We next describe the actions of $\mathcal{S}$ in response to user activations and oracle queries.

A.3.2 Simulation

The solver $\mathcal{S}$ simulate oracle queries as follows. $\mathcal{S}$ maintains list L_H that contains queries and answers of H oracle, and list L_S that contains queries and answers of SessionKeyReveal,

1. Send(Π, γ_i, γ_j): $\mathcal{S}$ selects a string δ_i, picks ephemeral secret key $x \in_U \mathbb{Z}_q$, computes ephemeral public key $\hat{\delta}_i$ honestly, records $(\Pi, \gamma_i, \gamma_j, \hat{\delta}_i)$, and returns it.
2. Send($\Pi, \gamma_i, \gamma_j, \hat{\delta}_i$): $\mathcal{S}$ selects a string δ_j, picks ephemeral secret key $y \in_U \mathbb{Z}_q$, computes ephemeral public key $\hat{\delta}_j$ honestly, records $(\Pi, \gamma_i, \gamma_j, \hat{\delta}_i, \hat{\delta}_j)$. and returns it.
3. Send($\Pi, \gamma_i, \gamma_j, \hat{\delta}_i, \hat{\delta}_j$): If $(\Pi, \gamma_i, \gamma_j, \hat{\delta}_i)$ is not recorded, $\mathcal{S}$ records the session $(\Pi, \gamma_i, \gamma_j, \hat{\delta}_i, \hat{\delta}_j)$ is not completed. Otherwise, $\mathcal{S}$ records the session is completed.
4. $H(\sigma_1, \sigma_2, \sigma_3, \Pi, \hat{\delta}_i, \hat{\delta}_j)$:
   a. If $(\sigma_1, \sigma_2, \sigma_3, \Pi, \hat{\delta}_i, \hat{\delta}_j)$ is recorded in list L_H, then return recorded value K.
   b. Else if $(\Pi, \hat{\delta}_i, \hat{\delta}_j)$ is recorded in list L_S, $X^z = \sigma_1$, $Y^z = \sigma_2$, and $e(X, Y) = e(g, \sigma_3)$, then return recorded value K and record it in list L_H.
   c. Else if $X^z = \sigma_1$, $Y^z = \sigma_2$, and $e(X, Y) = e(g, \sigma_3)$, $i = A, j = B$, and the session is j_A-th session of user U_A and the session is j_B-th session of user U_B, then $\mathcal{S}$ stops and is successful by outputting answer of gap BDH problem $e(\sigma_3, W) = \mathrm{BDH}(U, V, W)$.
   d. Otherwise, $\mathcal{S}$ returns random value K and record it in list L_H.
5. SessionKeyReveal($\mathtt{sid}$):
   a. If the session $\mathtt{sid}$ is not completed, return error.
   b. Else if $\mathtt{sid}$ is recorded in list L_S, then return recorded value K.
   c. Else if $(\sigma_1, \sigma_2, \sigma_3, \Pi, \hat{\delta}_i, \hat{\delta}_j)$ is recorded in list L_H, $X^z = \sigma_1$, $Y^z = \sigma_2$, and $e(X, Y) = e(g, \sigma_3)$, then return recorded value K and record it in list L_S.
   d. Otherwise, $\mathcal{S}$ returns random value K and record it in list L_S.
6. $H'(\{D_u\}_{u \in L(\gamma_i)}, \tilde{x})$: If $\{D_u\}_{u \in L(\gamma_i)}, \tilde{x}$ is used in j_A-th session of user U_A or used in j_B-th session of user U_B $\mathcal{S}$ aborts with failure. Otherwise, simulates random oracle in the usual way.
7. EphemeralKeyReveal($\mathtt{sid}$): $\mathcal{S}$ returns random value $\tilde{x}$ and record it.
8. StaticKeyReveal(γ_i): First, we consider γ_i is access tree with height 1 and threshold k. $\mathcal{S}$ selects random degree $k-1$ polynomial $q(x)$ s.t. $q(0) = z$, computes $D_i = g^{q(i)/t_i}$, and returns $\{D_u\}_{u \in L(\gamma_i)}$.
   In the case of the height is more than 1, $\mathcal{S}$ performs secret sharing iteratively, i.e., makes shares of z then makes shares of the share and so on, until reaching leaf node.
9. MasterKeyReveal(): $\mathcal{S}$ returns z.
10. EstablishParty(U_i, γ_i): $\mathcal{S}$ responds to the query faithfully.
11. Test($\mathtt{sid}$): If ephemeral public key X is not U or Y is not V in session $\mathtt{sid}$, then $\mathcal{S}$ aborts with failure. Otherwise, responds to the query faithfully.

12. If $\mathcal{A}$ outputs a guess γ, $\mathcal{S}$ aborts with failure.

A.3.3 Analysis

The simulation of $\mathcal{A}$ environment is perfect except with negligible probability. The probability that $\mathcal{A}$ selects the session, where ephemeral public key X is W, as the test session $\mathtt{sid}^*$ is at least $\frac{1}{n^2 s}$. Suppose this is indeed the case, $\mathcal{S}$ does not abort in Step 11.

Suppose event E_3 occurs, $\mathcal{S}$ does not abort in Step 6 except negligible probability, since EphemeralKeyReveal($\mathtt{sid}^*$) and EphemeralKeyReveal($\overline{\mathtt{sid}^*}$) are not queried.

Under event M^*, $\mathcal{A}$ queries correctly formed $\sigma_1, \sigma_2, \sigma_3$ to H. Therefore, $\mathcal{S}$ is successful as described in Step 4c and does not abort as in Step 12.

Hence, $\mathcal{S}$ is successful with probability $Pr(S) \geq \frac{p_3}{n^2 s^2}$, where p_3 is probability that $E_3 \wedge \overline{D} \wedge M^*$ occurs.

A.4 Other Events

A.4.1 Event $E_2 \wedge \overline{D} \wedge M^*$

In event E_2, test session $\mathtt{sid}^*$ has no matching session $\overline{\mathtt{sid}^*}$, and $\mathcal{A}$ queries EphemeralKeyReveal($\mathtt{sid}^*$) and does not query StaticKeyReveal(γ) s.t. $\phi(\gamma, \delta_B) = 1$ and StaticKeyReveal(γ) s.t. $\phi(\gamma, \delta_A) = 1$ due to the condition of freshness. Thus, $\mathcal{A}$ cannot obtain information about x, except with negligible guessing probability, since H' is a random oracle. Therefore, $\mathcal{S}$ performs the same reduction as in event $E_1 \wedge \overline{D} \wedge M^*$.

A.4.2 Event $E_4 \wedge \overline{D} \wedge M^*$

In event E_4, test session $\mathtt{sid}^*$ has matching session $\overline{\mathtt{sid}^*}$, and $\mathcal{A}$ queries EphemeralKeyReveal($\mathtt{sid}^*$) and EphemeralKeyReveal($\overline{\mathtt{sid}^*}$) and does not query StaticKeyReveal(γ) s.t. $\phi(\gamma, \delta_B) = 1$ and StaticKeyReveal(γ) s.t. $\phi(\gamma, \delta_A) = 1$ due to the condition of freshness. Moreover, $\mathcal{A}$ cannot obtain information about x and y, except with negligible guessing probability, since H' is a random oracle and output of StaticKeyReveal is randomized. Therefore, $\mathcal{S}$ performs the same reduction as in event $E_3 \wedge \overline{D} \wedge M^*$.

A.4.3 Event $E_5 \wedge \overline{D} \wedge M^*$

In event E_4, test session $\mathtt{sid}^*$ has matching session $\overline{\mathtt{sid}^*}$, and $\mathcal{A}$ queries StaticKeyReveal(γ) s.t. $\phi(\gamma, \delta_B) = 1$ and EphemeralKeyReveal($\overline{\mathtt{sid}^*}$) and does not query EphemeralKeyReveal($\mathtt{sid}^*$) and StaticKeyReveal(γ) s.t. $\phi(\gamma, \delta_A) = 1$ due to the condition of freshness. Moreover, $\mathcal{A}$ cannot obtain information about y, except with negligible guessing probability, since H' is a random oracle and output of StaticKeyReveal is randomized. Therefore, $\mathcal{S}$ performs the same reduction as in event $E_3 \wedge \overline{D} \wedge M^*$.

A.4.4 Event $E_6 \wedge \overline{D} \wedge M^*$

In event E_4, test session $\mathtt{sid}^*$ has matching session $\overline{\mathtt{sid}^*}$, and $\mathcal{A}$ queries EphemeralKeyReveal($\mathtt{sid}^*$) and StaticKeyReveal(γ) s.t. $\phi(\gamma, \delta_A) = 1$ and does not query StaticKeyReveal(γ) s.t. $\phi(\gamma, \delta_B) = 1$ and EphemeralKeyReveal($\overline{\mathtt{sid}^*}$) due to the condition of freshness. Moreover, $\mathcal{A}$ cannot obtain information about x, except with negligible guessing probability, since H' is a random oracle and output of StaticKeyReveal is randomized. Therefore, $\mathcal{S}$ performs the same reduction as in event $E_3 \wedge \overline{D} \wedge M^*$.

Appendix B: Proof of Theorem 7

We will show that if a polynomially bounded adversary $\mathcal{A}$ can distinguish the session key of a fresh session from a randomly chosen session key, we can solve the GBDH problem. Let κ be the security parameter, and let $\mathcal{A}$ be a polynomially (in κ) bounded adversary. We use adversary $\mathcal{A}$ to construct a GBDH solver $\mathcal{S}$ that succeeds with non-negligible probability. *Suc* denotes the event that $\mathcal{A}$ wins. Let *AskH* be the event that adversary $\mathcal{A}$ poses $(\sigma_1, \sigma_2, \sigma_3, (X, \{U\}, M_A, \rho_A), (Y, \{V\}, M_B, \rho_B))$ to H_3. Let $\overline{AskH}$ be the complement of event *AskH*. Let *sid* be any completed session owned by an honest party such that $sid \neq sid^*$ and *sid* is non-matching to sid^*. Since *sid* and sid^* are distinct and non-matching, the inputs to the key derivation function H_3 are different for *sid* and sid^*. Since H_3 is a random oracle, $\mathcal{A}$ cannot obtain any information about the test session key from the session keys of non-matching sessions. Hence, $\Pr[Suc \wedge \overline{AskH}] \leq \frac{1}{2}$ and $\Pr[Suc] = \Pr[Suc \wedge AskH] + \Pr[Suc \wedge \overline{AskH}] \leq \Pr[Suc \wedge AskH] + \frac{1}{2}$ whence $f(\kappa) \leq \Pr[Suc \wedge AskH]$. Henceforth, the event $Suc \wedge AskH$ is denoted by Suc^*.

We denote the master secret and public keys by g^z and (g, g^r, g_T^z) respectively. For party P, we denote the set of attributes by $\mathbb{S}_P$, the static secret key by $(S'_P, \{T_P\}, \{S_P\})$, the ephemeral secret key by $\tilde{u}_1, \ldots, \tilde{u}_{n_P}$, and the exponent of the ephemeral public keys by $u_j = H_2(S'_P, \{T_P\}, \{S_P\}, \tilde{u_j})$ for $1 \leq j \leq n_P$. We also denote the session key by K. Assume that $\mathcal{A}$ succeeds in an environment with N users, activates at most L sessions within a party.

We consider the following events.

- Let *AskS* be the event $\mathcal{A}$ poses the static secret key $(S'_P, \{T_P\}, \{S_P\})$ to H_2, *before* asking StaticKeyReveal queries or MasterKeyReveal query, or *without* asking StaticKeyReveal queries or MasterKeyReveal query.
- Let $\overline{AskS}$ be the complement of event *AskS*.

We consider the following events that cover all cases of the behavior of $\mathcal{A}$.

- Let E_1 be the event that the test session sid^* has no matching session $\overline{sid}^*$ and $\mathcal{A}$ poses StaticKeyReveal($\mathbb{S}$) s.t. $\mathbb{S} \in \mathbb{A}_B$.
- Let E_2 be the event that the test session sid^* has no

matching session $\overline{sid}^*$ and $\mathcal{A}$ poses $\mathsf{EphemeralKeyReveal}(sid^*)$.

- Let E_3 be the event that the test session sid^* has matching session $\overline{sid}^*$ and $\mathcal{A}$ poses MasterKeyReveal or poses $\mathsf{StaticKeyReveal}(\mathbb{S})$ s.t. $\mathbb{S} \in \mathbb{A}_B$ and $\mathsf{StaticKeyReveal}(\mathbb{S})$ s.t. $\mathbb{S} \in \mathbb{A}_A$.
- Let E_4 be the event that the test session sid^* has matching session $\overline{sid}^*$ and $\mathcal{A}$ poses $\mathsf{EphemeralKeyReveal}(sid^*)$ and $\mathsf{EphemeralKeyReveal}(\overline{sid}^*)$.
- Let E_5 be the event that the test session sid^* has matching session $\overline{sid}^*$ and $\mathcal{A}$ poses $\mathsf{StaticKeyReveal}(\mathbb{S})$ s.t. $\mathbb{S} \in \mathbb{A}_B$ and $\mathsf{EphemeralKeyReveal}(\overline{sid}^*)$.
- Let E_6 be the event that the test session sid^* has matching session $\overline{sid}^*$ and $\mathcal{A}$ poses $\mathsf{EphemeralKeyReveal}(sid^*)$ and $\mathsf{StaticKeyReveal}(\mathbb{S})$ s.t. $\mathbb{S} \in \mathbb{A}_A$.

To finish the proof, we investigate events $AskS \wedge Suc^*$ and $E_i \wedge \overline{AskS} \wedge Suc^*$ $(i = 1, \ldots, 6)$ that cover all cases of event Suc^*.

B.1 Event $AskS \wedge Suc^*$

In the event $AskS$, $\mathcal{A}$ poses the static secret key $(S'_A, \{T_A\}, \{S_A\})$ to H_2 before posing StaticKeyReveal queries or MasterKeyReveal query, or without posing StaticKeyReveal queries or MasterKeyReveal query. The solver $\mathcal{S}$ embeds instance as $g_T^z = e(\alpha, \beta)$ and extract $g^z = g^{ab}$ from S'_A and T_{A_1} because $\mathcal{S}$ knows r and can compute $S'_A / T^r_{A_1} = g^z$. Then, $\mathcal{S}$ obtains $\mathsf{BDH}(\alpha, \beta, \gamma) = e(g^z, \gamma)$.

B.2 Event $E_1 \wedge \overline{AskS} \wedge Suc^*$

In the event E_1, the test session sid^* has no matching session $\overline{sid}^*$, $\mathcal{A}$ poses $\mathsf{StaticKeyReveal}(\mathbb{S})$ s.t. $\mathbb{S} \in \mathbb{A}_B$, and does not pose $\mathsf{EphemeralKeyReveal}(sid^*)$, MasterKeyReveal or $\mathsf{StaticKeyReveal}(\mathbb{S})$ s.t. $\mathbb{S} \in \mathbb{A}_A$ by the condition of freshness. In the case of event $E_1 \wedge \overline{AskS} \wedge M^*$, $\mathcal{S}$ performs the following steps.

B.2.1 Setup

The GBDH solver $\mathcal{S}$ receives a BDH tuple $(g, \alpha, \beta, \gamma)$ as a challenge. Also, $\mathcal{S}$ receives (M_A^*, ρ_A^*) for $\mathbb{A}_A^*$ and (M_B^*, ρ_B^*) for $\mathbb{A}_B^*$ as a challenge access structure from $\mathcal{A}$. M_A^* is $\ell_A^* \times n_A^*$ matrix and M_B^* is $\ell_B^* \times n_B^*$ matrix.

$\mathcal{S}$ chooses $z' \in_R \mathbb{Z}_p$ and lets $g_T^z := e(\alpha, \beta) g_T^{z'}$ (i.e., $z = ab + z'$ implicitly). $\mathcal{S}$ embeds $g^r := \alpha$ and outputs the master public key $MPK = (g, g^r, g_T^z)$.

$\mathcal{S}$ randomly selects two parties A, B and integers $i_A \in_R [1, L]$ that becomes a guess of the test session with probability $1/n^2 L$. $\mathcal{S}$ sets the ephemeral public key of i_Ath session of A as follows: First, $\mathcal{S}$ programs the random oracle H_1 by building a table. For each j, k pair where $1 \leq j \leq n_{max}$ and k such that $\rho_A^*(i) = k$ for $1 \leq i \leq \ell_A^*$, $\mathcal{S}$ chooses a random value $h_{j,k} \in_R \mathbb{Z}_p$. Then, let $H_1(j,k) = g^{h_{j,k}} \alpha^{M^*_{A i,j}}$ for $1 \leq j \leq n_A^*$ and k such that $\rho_A^*(i) = k$ for $1 \leq i \leq \ell_A^*$. Otherwise, let $H_1(j,k) = g^{h_{j,k}}$[†]. Next, $\mathcal{S}$ lets $X := \gamma$ and chooses random values $x_2, \ldots, x_{n_A^*} \in \mathbb{Z}_p$ and sets $x_1 = 0$. Then, $\mathcal{S}$ computes $U_{i,j} = \alpha^{M^*_{A i,j} x_j} \gamma^{-h_{j,\rho_A^*(i)}}$ for $1 \leq i \leq \ell_A^*$ and $1 \leq j \leq n_A^*$ (i.e., $(u_1, \ldots, u_{n_A^*}) = (c, c + x_2, \ldots, c + x_{n_A^*})$ implicitly), and $U_{i,j} = \gamma^{-h_{j,\rho^*(i)}}$ for $1 \leq i \leq \ell_A^*$ and $n_A^* + 1 \leq j \leq n_{max}$. Finally, $\mathcal{S}$ sets the ephemeral public key $EPK_A = (X, \{U\})$ of i_Ath session of A.

B.2.2 Simulation

$\mathcal{S}$ simulates oracle queries by $\mathcal{A}$ as follows. $\mathcal{S}$ maintains the lists $\mathcal{L}_{H_1}$, $\mathcal{L}_{H_2}$ and $\mathcal{L}_{H_3}$ that contains queries and answers of the H_1, H_2 and H_3 oracles respectively, and the list $\mathcal{L}_K$ that contains queries and answers of SessionKeyReveal.

1. $H_1(j,k)$: If there exists a tuple $(j, k, *) \in \mathcal{L}_{H_1}$, $\mathcal{S}$ returns the registered value. Otherwise, $\mathcal{S}$ chooses $h_{j,k} \in_R \mathbb{Z}_p$, returns $g^{h_{j,k}}$ and records it to $\mathcal{L}_{H_1}$.
2. $H_2(S', \{T\}, \{S\}, \tilde{u}_j)$: If there exists a tuple $(S', \{T\}, \{S\}, \tilde{u}_j, *) \in \mathcal{L}_{H_2}$, $\mathcal{S}$ returns the registered value[††]. Otherwise, $\mathcal{S}$ chooses $u_j \in_R \mathbb{Z}_p$, returns u_j and records it to $\mathcal{L}_{H_2}$.
3. $H_3(\sigma_1, \sigma_2, \sigma_3, (X, \{U\}, M_P, \rho_P), (Y, \{V\}, M_{\bar{P}}, \rho_{\bar{P}}))$:
   a. If there exists a tuple $(\sigma_1, \sigma_2, \sigma_3, (X, \{U\}, M_P, \rho_P), (Y, \{V\}, M_{\bar{P}}, \rho_{\bar{P}}), *) \in \mathcal{L}_{H_3}$, $\mathcal{S}$ returns the registered value.
   b. Else if there exists a tuple $(\mathcal{I}, \mathbb{S}_P, \mathbb{S}_{\bar{P}}, (X, \{U\}, M_P, \rho_P), (Y, \{V\}, M_{\bar{P}}, \rho_{\bar{P}}), *) \in \mathcal{L}_K$ or $(\mathcal{R}, \mathbb{S}_{\bar{P}}, \mathbb{S}_P, (X, \{U\}, M_P, \rho_P), (Y, \{V\}, M_{\bar{P}}, \rho_{\bar{P}}), *) \in \mathcal{L}_K$, $\mathsf{DBDH}(X, \alpha, \beta, \sigma_1) = 1$, $\mathsf{DBDH}(Y, \alpha, \beta, \sigma_2) = 1$ and $e(X, Y) = e(g, \sigma_3)$, then $\mathcal{S}$ returns the recorded value and record it in the list $\mathcal{L}_{H_3}$.
   c. Else if $\mathsf{DBDH}(X, \alpha, \beta, \sigma_1) = 1$, $\mathsf{DBDH}(Y, \alpha, \beta, \sigma_2) = 1$, $e(X, Y) = e(g, \sigma_3)$, $P = A$, $\tilde{P} = B$ and the session is i_A-th session of A, then $\mathcal{S}$ stops and is successful by outputting the answer of the GBDH problem $\sigma_1 = \mathsf{BDH}(\alpha, \beta, \gamma)$.
   d. Otherwise, $\mathcal{S}$ returns a random value $K \in_R \{0,1\}^\kappa$ and records it in the list $\mathcal{L}_{H_3}$.
4. $\mathsf{Send}(\mathcal{I}, \mathbb{S}_P, \mathbb{S}_{\bar{P}})$: If $P = A$ and the session is i_A-th session of A, $\mathcal{S}$ returns the ephemeral public key EPK_A computed in the setup. Otherwise, $\mathcal{S}$ computes the ephemeral public key EPK_P obeying the protocol, returns it and records $(\mathbb{S}_P, \mathbb{S}_{\bar{P}}, (EPK_P, M_P, \rho_P))$.
5. $\mathsf{Send}(\mathcal{R}, \mathbb{S}_{\bar{P}}, \mathbb{S}_P, (EPK_P, M_P, \rho_P))$: $\mathcal{S}$ computes the ephemeral public key $EPK_{\bar{P}}$ obeying the protocol, returns it and records $(\mathbb{S}_P, \mathbb{S}_{\bar{P}}, (EPK_P, M_P, \rho_P),$

[†] All $H_1(j,k)$ are distributed randomly due to the $g^{h_{j,k}}$. Also, since ρ_A^* is injective, for any k there is at most one i such that $\rho_A^*(i) = k$.

[††] $c, x_2, \ldots, x_{n_A^*}$ are not registered in $\mathcal{L}_{H_2}$. However, $\mathcal{A}$ does not pose $\mathsf{EphemeralKeyReveal}(sid^*)$ and so cannot know information about $\tilde{c}, \tilde{x}_2, \ldots, \tilde{x}_{n_A^*}$ corresponding to $c, x_2, \ldots, x_{n_A^*}$. Thus, $\mathcal{A}$ cannot distinguish the real experiment from the simulation by such queries.

$(EPK_{\bar{P}}, M_{\bar{P}}, \rho_{\bar{P}}))$ as the completed session.

6. $\mathsf{Send}(\mathcal{I}, \mathbb{S}_{\bar{P}}, \mathbb{S}_P, (EPK_P, M_P, \rho_P), (EPK_{\bar{P}}, M_{\bar{P}}, \rho_{\bar{P}}))$: If $(\mathbb{S}_P, \mathbb{S}_{\bar{P}}, (EPK_P, M_P, \rho_P))$ is not recorded, $\mathcal{S}$ records the session $(\mathbb{S}_P, \mathbb{S}_{\bar{P}}, (EPK_P, M_P, \rho_P))$ is not completed. Otherwise, $\mathcal{S}$ records the session is completed.
7. $\mathsf{SessionKeyReveal}(sid)$:
   a. If the session sid is not completed, $\mathcal{S}$ returns an error message.
   b. Else if sid is recorded in the list $\mathcal{L}_K$, then $\mathcal{S}$ returns the recorded value K.
   c. Else if $(\sigma_1, \sigma_2, \sigma_3, (X, \{U\}, M_P, \rho_P), (Y, \{V\}, M_{\bar{P}}, \rho_{\bar{P}}))$ is recorded in the list $\mathcal{L}_{H_3}$, $\mathsf{DBDH}(X, \alpha, \beta, \sigma_1) = 1$, $\mathsf{DBDH}(Y, \alpha, \beta, \sigma_2) = 1$ and $e(X, Y) = e(g, \sigma_3)$, then $\mathcal{S}$ returns the recorded value K and records it in the list $\mathcal{L}_K$.
   d. Otherwise, $\mathcal{S}$ returns a random value $K \in_R \{0,1\}^\kappa$ and records it in the list $\mathcal{L}_K$.
8. $\mathsf{EphemeralKeyReveal}(sid)$: $\mathcal{S}$ returns a random value $\tilde{u}_1, \ldots, \tilde{u}_n$ where n is the size of the column of M in sid and records it.
9. $\mathsf{StaticKeyReveal}(\mathbb{S}_P)$: In the event E_1, $\mathbb{S}_P$ does not satisfy M^*_A. Without loss of generality, we can suppose that $M^*_{Ai,j} = 0$ for $n^*_A + 1 \leq j \leq n_{max}$. By the definition of LSSSs, $\mathcal{S}$ can efficiently find a vector $\vec{w} = (w_1, \ldots, w_{n_{max}}) \in \mathbb{Z}_p^{n_{max}}$ such that $w_1 = -1$ and for all i where $\rho^*_A(i) \in \mathbb{S}_P$ we have that $\vec{w} \cdot M^*_{A_{i,j}} = 0$. Note that, we can simply let $w_j = 0$ and consider $M^*_{Ai,j} = 0$ for $n^*_A + 1 \leq j \leq n_{max}$.
   $\mathcal{S}$ sets the static secret key SK_P as follows: $\mathcal{S}$ chooses random values $y_1, \ldots, y_{n_{max}} \in_R \mathbb{Z}_p$ and computes $S'_P := g^{z'}\alpha^{y_1}$ and $T_{P_j} = g^{y_j} \cdot \beta^{w_j}$ (i.e., $t_j = y_j + w_j b$ implicitly). Also, $\mathcal{S}$ sets S_{P_k} for $k \in \mathbb{S}_P$ as $S_{P_k} := \prod_{i \leq j \leq n_{max}} T_{P_j}^{h_{j,k}}$ for $k \in \mathbb{S}_P$ where there is no i such that $\rho^*_A(i) = k$. For $k \in \mathbb{S}_P$ where there is i such that $\rho^*_A(i) = k$, $\mathcal{S}$ sets $S_{P_k} := \prod_{i \leq j \leq n_{max}} g^{h_{j,k} y_j} \cdot \beta^{h_{j,k}} \cdot \gamma^{M_{A^*_{i,j}} y_j}$.
10. $\mathsf{MasterKeyReveal}$: $\mathcal{S}$ aborts with failure[†].
11. $\mathsf{Establish}(P, \mathbb{S}_P)$: $\mathcal{S}$ responds to the query as the definition.
12. $\mathsf{Test}(sid)$: If the ephemeral public key in the session sid is not EPK_A, then $\mathcal{S}$ aborts with failure. Otherwise, responds to the query as the definition.
13. If $\mathcal{A}$ outputs a guess b', $\mathcal{S}$ aborts with failure.

B.2.3 Analysis

The simulation for $\mathcal{S}$ is perfect except with negligible probability. The probability that $\mathcal{A}$ selects the session, where the ephemeral public key is EPK_A, as the test session sid^* is at least $\frac{1}{N^2 L}$.

Under the event Suc^*, $\mathcal{A}$ poses correctly formed $\sigma_1, \sigma_2, \sigma_3$ to H_3. Therefore, $\mathcal{S}$ is successful and does not abort.

Hence, $\mathcal{S}$ is successful with probability $Pr[\mathcal{S}$ solves the GBDH problem$] \geq \frac{p_1}{n^2 s}$, where p_1 is probability that $E_1 \wedge \overline{AskS} \wedge M^*$ occurs.

B.3 Other Events

B.3.1 Event $E_2 \wedge \overline{AskS} \wedge Suc^*$

In the event E_2, the test session sid^* has no matching session $\overline{sid}^*$, $\mathcal{A}$ poses $\mathsf{EphemeralKeyReveal}(sid^*)$, and $\mathcal{A}$ does not pose $\mathsf{StaticKeyReveal}(\mathbb{S})$ s.t. $\mathbb{S} \in \mathbb{A}_A$, $\mathsf{StaticKeyReveal}(\mathbb{S})$ s.t. $\mathbb{S} \in \mathbb{A}_B$ or $\mathsf{MasterKeyReveal}$ by the condition of freshness. Thus, $\mathcal{A}$ cannot obtain no information about $u_1, \ldots, u_{n_{max}}$ except negligible guessing probability, since H_2 is the random oracle. Hence, $\mathcal{S}$ performs the reduction same as in the case of event $E_1 \wedge \overline{AskS} \wedge Suc^*$.

B.3.2 Event $E_3 \wedge \overline{AskS} \wedge Suc^*$

In the event E_3, the test session sid^* has the matching session $\overline{sid}^*$, $\mathcal{A}$ poses $\mathsf{MasterKeyReveal}$ or poses both $\mathsf{StaticKeyReveal}(\mathbb{S})$ s.t. $\mathbb{S} \in \mathbb{A}_A$ and $\mathsf{StaticKeyReveal}(\mathbb{S})$ s.t. $\mathbb{S} \in \mathbb{A}_B$, and $\mathcal{A}$ does not pose $\mathsf{EphemeralKeyReveal}(sid^*)$ and $\mathsf{EphemeralKeyReveal}(\overline{sid}^*)$ by the condition of freshness. $\mathcal{S}$ simulates the setup and key generations obeying the scheme. $\mathcal{S}$ embeds the BDH instance as $X = g^x = \alpha$, $Y = g^y = \beta$ in sid^*, and extracts g^{ab} from $\sigma_3 = g^{xy}$. Then, $\mathcal{S}$ obtains $\mathsf{BDH}(\alpha, \beta, \gamma)$ by $e(\sigma_3, \gamma)$.

B.3.3 Event $E_4 \wedge \overline{AskS} \wedge Suc^*$

In the event E_4, the test session sid^* has the matching session $\overline{sid}^*$, $\mathcal{A}$ poses $\mathsf{EphemeralKeyReveal}(sid^*)$ and $\mathsf{EphemeralKeyReveal}(\overline{sid}^*)$, and does not pose $\mathsf{StaticKeyReveal}(\mathbb{S})$ s.t. $\mathbb{S} \in \mathbb{A}_A$, $\mathsf{StaticKeyReveal}(\mathbb{S})$ s.t. $\mathbb{S} \in \mathbb{A}_B$ or $\mathsf{MasterKeyReveal}$ by the condition of freshness. Then, $\mathcal{A}$ cannot obtain no information about $u_1, \ldots, u_{n_{max}}$ and $v_1, \ldots, v_{n_{max}}$ except negligible guessing probability because H_2 is the random oracle and outputs of $\mathsf{StaticKeyReveal}$ are randomized. Hence, $\mathcal{S}$ performs the reduction same as in the case of event $E_3 \wedge \overline{AskS} \wedge Suc^*$.

B.3.4 Event $E_5 \wedge \overline{AskS} \wedge Suc^*$

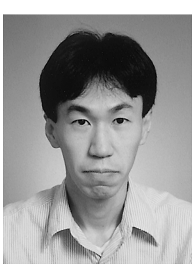
In the event E_5, the test session sid^* has the matching session $\overline{sid}^*$, $\mathcal{A}$ poses $\mathsf{StaticKeyReveal}(\mathbb{S})$ s.t. $\mathbb{S} \in \mathbb{A}_B$ and $\mathsf{EphemeralKeyReveal}(\overline{sid}^*)$, and does not pose $\mathsf{EphemeralKeyReveal}(sid^*)$, $\mathsf{StaticKeyReveal}(\mathbb{S})$ s.t. $\mathbb{S} \in \mathbb{A}_A$ or $\mathsf{MasterKeyReveal}$ by the condition of freshness. Then, $\mathcal{A}$ cannot obtain no information about $v_1, \ldots, v_{n_{max}}$ except negligible guessing probability because H_2 is the random oracle and outputs of $\mathsf{StaticKeyReveal}$ are randomized. Hence, $\mathcal{S}$ performs the reduction same as in the case of event $E_3 \wedge \overline{AskS} \wedge Suc^*$.

[†]In the event E_1, $\mathcal{A}$ does not pose $\mathsf{MasterKeyReveal}$ query.

B.3.5 Event $E_6 \wedge \overline{AskS} \wedge Suc^*$

In the event E_6, the test session sid^* has the matching session $\overline{sid}^*$, $\mathcal{A}$ poses StaticKeyReveal($\mathbb{S}$) s.t. $\mathbb{S} \in \mathbb{A}_A$ and EphemeralKeyReveal(sid^*), and does not pose EphemeralKeyReveal($\overline{sid}^*$), StaticKeyReveal($\mathbb{S}$) s.t. $\mathbb{S} \in \mathbb{A}_B$ or MasterKeyReveal by the condition of freshness. Then, $\mathcal{A}$ cannot obtain no information about $u_1, \ldots, u_{n_{max}}$ except negligible guessing probability because H_2 is the random oracle and outputs of StaticKeyReveal are randomized. Hence, $\mathcal{S}$ performs the reduction same as in the case of event $E_3 \wedge \overline{AskS} \wedge Suc^*$.

Atsushi Fujioka received his B.E., M.E., and D.E. degrees in electrical and electronic engineering from Tokyo Institute of Technology in 1985, 1987 and 1990, respectively. Dr. Fujioka joined NTT, Nippon Telegraph and Telephone Corporation, in 1990, and was an academic guest in Swiss Federal Institute of Technology in 1993–1994. He received the 30th Best Achievement Award (the 9th Kobayashi Memorial Award) from IEICE. Dr. Fujioka is presently engaged in research on cryptography and information security as the research group leader in NTT Information Sharing Platform Laboratories. He is a member of IPSJ and IACR.



Koutarou Suzuki received the B.Sc., M.Sc., and Ph.D. degrees from the University of Tokyo in 1994, 1996, and 1999, respectively. He joined NTT, Nippon Telegraph and Telephone Corporation, in 1999. He is engaged in research on public key cryptography at NTT Information Sharing Platform Laboratories. He received the SCIS Paper Award in 2002. He is a member of the International Association for Cryptologic Research (IACR) and Information Processing Society of Japan (IPSJ).

Kazuki Yoneyama received the B.E., M.E. and Ph.D. degrees from the University of Electro-Communications, Tokyo, Japan, in 2004, 2006 and 2008, respectively. He is presently engaged in research on cryptography at NTT Information Sharing Platform Laboratories, since 2009. He is a member of the International Association for Cryptologic Research, IPSJ and JSIAM.