Efficient, Pairing-Free, One Round Attribute-Based Authenticated Key Exchange

Suvradip Chakraborty[1] , Srinivasan Raghuraman[1], C. Pandu Rangan[1]

1 – Theoretical Computer Science Lab, Department of Computer Science and Engineering, Indian Institute of Technology Madras, Chennai, India
suvradip1111@gmail.com, srini131293@gmail.com, prangan55@gmail.com

Abstract. In this paper, we present a single round two-party attribute-based authenticated key exchange protocol. Since pairing is a costly operation and the composite order groups must be very large to ensure security, we focus on pairing free protocols in prime order groups. We propose a new protocol that is pairing free, working in prime order group and having tight reduction to Strong Diffie Hellman (SDH) problem under the Attribute-based CK model which is a natural extension of the CK model for the public key setting. Our proposed attribute based authenticated key exchange protocol (ABAKE) also does not depend on any underlying attribute based encryption schemes unlike the previous solutions for ABAKE. Ours is the first scheme that removes this restriction. Thus, the first major advantage is that smaller key sizes are sufficient to achieve comparable security. Our scheme has several other advantages. The major one being the capability to handle active adversaries. Most of the previous Attribute-Based authenticated key exchange protocols can offer security only under passive adversaries. Our protocol recognizes the corruption by an active adversary and aborts the process. In addition to this property, our scheme satisfies other security properties that are not covered by CK model such as forward secrecy, key compromise impersonation attacks and ephemeral key compromise impersonation attacks.

Keywords: Authenticated Key Exchange, attribute based authenticated key exchange, CK model, ABCK model, Random Oracle Model, Forward Secrecy, Key Compromise Impersonation attacks.

1 Introduction

Attribute based encryption (ABE) was introduced by Sahai and Waters [SW05] which allows for fine-grained access control on encrypted data and reduces bulk encryptions to a number of people who have several common characteristics. After that a lot of other ABE schemes were proposed [GPSW06], [BSW07], [GJPS08], [OSW07], [LOS$^+$10], [OT10]. The goal of an Authenticated Key Exchange (AKE) protocol is for two parties to establish a common shared session key which they can later use to securely communicate with each other. *Attribute-based* AKE (ABAKE) is a new variant of the AKE that allows users to authenticate each other using their *attributes* unlike in the PKI settings where the users authenticate each other using their identities. ABAKE can hide the identity information of an individual, which allows users to achieve mutual authentication and establish a secret session key by their attributes and some fine grained access control policy. Attribute based key exchange finds its application in distributed collaborative systems where it is more convenient for users to communicate with other users using their roles or responsibilities which can be described by attributes, interactive chat rooms , online forums where a user can have read/write access to threads only if they have desired attributes etc. Hence an authenticated key exchange protocol that facilitates attribute usage can be employed in this setting. Besides it may also be used to securely transfer some sensitive information such as medical history which is established by some AKE scheme.

1.1 Related Works

In the recent literature some ABAKE are proposed. Ateniese *et al.* [AKB07] proposed a fuzzy handshake technique that is closely related to the ABAKE model. However there are some differences between the two

as their scheme can only handle simple authentication condition by allowing only a single threshold gates as opposed to several threshold gates in AB-AKE settings. Gorantla *et al.* [GBN10] proposed the first ABAKE scheme based on key encapsulation mechanism which provides parties with the fine-grained access control based on parties attributes. However it does not provide the flexibility of each user to select their access structures which they want their peers to satisfy. So in that sense their scheme is not an ABAKE scheme. Besides the security of their scheme is based on the BR model [BR94].
Birkett and Stebila [BS10] introduced the concept of predicate based key exchange with fine-grained access control with a predicate-based signature and here the parties can specify the condition the peer is expected to satisfy. However their scheme is proven secure in the random oracle model in BR model. The BR model does not allow the adversary to reveal the session specific informations and ephemral keys. Yoneyama [Yon10] proposed a two-pass attribute based key exchange secure in the random oracle model under the Gap Bilinear Diffie Hellman assumption in the attribute based eCK [LLM07] model. But it does not achieve full security (i.e. adaptive security) as it relies on Waters CP-ABE which is selectively secure.

The previous works on attribute based key agreement do not consider an active adversary. An active adversary is one which can extract the messages that are exchanged during key agreement and modify them arbitrarily during transit. In the scheme presented in [Yon10], the adversary can extract the ephemeral component $(X, \{U\})$ and change it to $(X', \{U'\})$ and chooses an access structure by itself that is trivially satisfied by the attributes of user B and send it to B. Similarly he can extract the ephemeral component $(Y, \{V\})$ and change it to $(Y', \{V'\})$ and chooses an access structure by itself that is trivially satisfied by the attributes of user A and send it to A. Thus the final shared secret key of A and B will not be in agreement. Our protocol avoids this kind of an attack by a signature on the ephemeral components. In our scheme, we use a Schnorr group and hence the exponentiation operations are cheaper even though it involves more exponentiation operations. This is because in a Schnorr group the exponent is from a group ${Z_p}^*$ where size of p is 224 bits according to *http://www.keylength.com/en/4/*.

1.2 Our Contribution

- In this paper, we present an attribute based key agreement protocol which can be proved secure under the Strong-Diffie Hellman (SDH) assumption [AKO09] in the random oracle model. We extend the techniques used in [VSVR13] for attribute based settings. Doing this is not trivial since in an attribute based system the keys and the ciphertexts have much more richer structure than identity based encryption schemes. We are able to achieve a tight reduction to the Strong Diffie Hellman problem based on the random oracle model.

- All the previous known attribute based key agreement protocols use well known existing attribute based encryption schemes to get a key agreement among the users. Hence the security of the key agreement were implicitly relying on the security guarantees provided by the underlying encryption schemes. Ours is the first scheme that removes this restriction and we get a key agreement protocol that does not rely on any attribute based encryption scheme. Moreover our construction is also efficient as it does not involve any pairing computations.

- Our scheme is also resistant to a dynamic active adversary which is allowed to modify the components exchanged during the key agreement. The scheme performs a check which will detect any tampering done on the components. In this way, a fully authenticated key agreement protocol (both the parties are mutually authenticated to each other) is achieved.

- The protocol also satisfies additional security properties like forward secrecy, key compromise impersonation attacks.

- In a practical sense, we can use any string as an attribute in our protocol because the setup algorithm of our protocol does not depend on the number of attribute candidates (i.e., the setup algorithm outputs constant size parameters).

- Finally, we prove the security of our ABAKE system in the attribute based CK (ABCK) model which is a natural extension of the CK model for public key settings. In the ABCK model the adversary is allowed to pose queries that allows him to reveal the static secret key, master secret key and the ephemeral secret key. Also the freshness conditions are a little different than the CK model and the parties are identified by a set of attribute $\mathbb{S}_P$. We prove the security of our ABAKE in this model under the SDH assumption. From the relation between hard problem and the instance of the protocol, it is clear that the key size be just same as the problem size that makes the SDH problem hard. Such tight reductions imply stronger security even with smaller keys. Thus, in practice, we may obtain a decent degree of security with reasonable sized keys.

1.3 Organization

In section 2, we present the preliminaries required for our ABAKE protocol. In more details, subsections 2.1- 2.4 provide the required notations and the necessary details on access structure,linear secret sharing schemes and the security assumptions. In section 3, we describe ABCK model and define the security of ABAKE schemes in this ABCK model. In section 4 we present our ABAKE scheme and in section 5 we prove the security of our ABAKE scheme. Section 6 describes the additional security guarantees that our scheme achieves. Finally section 7 concludes the paper.

2 Preliminaries

2.1 Notation

Throughout this work, we denote the security parameter by κ.

We denote by $x \in_R X$ the fact that the value of the variable x is chosen uniformly at random from the set of values X.

We set up notations for vectors which are used extensively. We denote by $\overrightarrow{a}$ a vector, which is the tuple of values $(a_1, \ldots, a_n)$, where n is the length of the vector $\overrightarrow{a}$. We define the outcome of various operations on a vector as used in this work.

1. For any function f, we denote by $f(\overrightarrow{a})$ the tuple of values $(f(a_1), \ldots, f(a_n))$.
2. For any function f and scalar c, we denote by $f(c, \overrightarrow{a})$ the tuple of values $(f(c, a_1), \ldots, f(c, a_n))$.
3. For any function f, we denote by $f\left(\overrightarrow{a}, \overrightarrow{b}\right)$ the tuple of values $(f(a_1, b_1), \ldots, f(a_n, b_n))$, where n is the length of the vectors $\overrightarrow{a}$ and $\overrightarrow{b}$.

As illustrations to each of the aforementioned,

1. Suppose H is a hash function with domain $\{0,1\}^*$. This could be a keyed-hash function, and for simplicity, we ignore mentioning the key repeatedly. We denote by $H(\overrightarrow{a})$ the tuple of values $(H(a_1), \ldots, H(a_n))$.
2. Suppose $\mathbb{G}$ is a multiplicative group and $g \in \mathbb{G}$. We denote by $g^{\overrightarrow{a}}$ the tuple of values $(g^{a_1}, \ldots, g^{a_n})$.
3. We denote by $\overrightarrow{a} + \overrightarrow{b}$ and $\overrightarrow{a} \cdot \overrightarrow{b}$, respectively, the tuple of values $(a_1 + b_1, \ldots, a_n + b_n)$ and the tuple of values $(a_1 \cdot b_1, \ldots, a_n \cdot b_n)$, respectively, where n is the length of the vectors $\overrightarrow{a}$ and $\overrightarrow{b}$.

2.2 Access Structure

Definition 1. *Access Structure [Bei96] Let $\{P_1, \cdots P_n\}$ be a set of parties. A collection $\mathbb{X} \subseteq 2^{\{P_1, \cdots P_n\}}$ is monotone if $\forall B, C$: if $B \in \mathbb{X}$ and $B \subseteq C$, then $C \in \mathbb{X}$. An access structure (respectively, monotone access structure) is a collection (respectively, monotone collection) $\mathbb{X}$ of non-empty subsets of $P_1, \cdots P_n$, i.e $\mathbb{A} \subseteq 2^{\{P_1, \cdots P_n\}} \backslash \{\emptyset\}$. The sets in $\mathbb{A}$ are called the authorized sets, and the sets not in $\mathbb{X}$ are called the unauthorized sets.*

In our setting, attributes will play the role of parties and we will only deal with monotone access structures. We note that it is possible to (inefficiently) realize general access structures with our techniques by having the negation of an attribute be a separate attribute (so the total number of attributes will be doubled).

2.3 Linear Secret Sharing

Our construction will employ linear secret sharing schemes (LSSS). We use the definition adapted from [Wat11].

Definition 2. *(Linear Secret Sharing (LSSS)) A secret sharing scheme π over a set of parties $\mathcal{P}$ 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 A called the share-generating matrix for Π. The matrix A has l rows and n columns. For all $i = 1, \ldots, l$, the i^{th} row of A is labeled by a party $\rho(i)$ (ρ is a function from $\{i = 1, \ldots, l\}$ to $\mathcal{P}$). When we consider the column vector $v = (s, r_2 \cdots r_n)$, where $s \in \mathbb{Z}_p$ is the secret to be shared and $r2, \cdots r_n \in \mathbb{Z}_p$ are randomly chosen, then Av is the vector of l shares of the secret s according to Π. The share $(Av)_i$ belongs to party $\rho(i)$.*

We note the *linear reconstruction* property: we suppose that Π is an LSSS for access structure $\mathbb{A}$. We let S denote an authorized set, and define $I \subseteq \{i = 1, \cdots, l,\}$ as $I = \{i|\rho(i) \in S\}$. Then the vector $(1, 0, \cdots, 0)$ is in the span of rows of A indexed by I, and there exist constants $\{w_i \in \mathbb{Z}_p\}_{i \in I}$ such that for any valid shares $\{\lambda_i\}$ of a secret s according to Π, we have: $\sum_{i \in I} w_i \lambda_i = s$. These constants $\{w_i\}$ can be found in time polynomial in the size of the share-generating matrix A [Wat11]. We note that for unauthorized sets, no such constants $\{w_i\}$ exist.

Boolean Formulas Access policies might also be described in terms of monotonic boolean formulas. LSSS access structures are more general and can be derived from such representations. More precisely, one can use standard techniques to convert any monotonic boolean formula into a corresponding LSSS matrix. We can represent the boolean formula as an access tree, where the interior nodes are AND and OR gates, and the leaf nodes correspond to attributes. The number of rows in the corresponding LSSS matrix will be same as the number of leaf nodes in the access tree.

2.4 Complexity Assumptions

In this section, we present a brief overview of the hard problem assumptions.

Definition 3. ***Computation Diffie-Hellman Problem (CDH) -*** *Given $(g, g^a, g^b) \in_R \mathbb{G}^3$ for unknown $a, b \in \mathbb{Z}_q^*$, where $\mathbb{G}$ is a cyclic prime order multiplicative group with g as a generator and q the order of the group, the CDH problem in $\mathbb{G}$ is to compute g^{ab}.*

The advantage of any probabilistic polynomial time algorithm $\mathcal{A}$ in solving the CDH problem in $\mathbb{G}$ is defined as

$$Adv_{\mathcal{A}}^{CDH} = Pr\left[\mathcal{A}(g, g^a, g^b) = g^{ab} \mid a, b \in \mathbb{Z}_q^*\right]$$

The CDH Assumption is that, for any probabilistic polynomial time algorithm $\mathcal{A}$, the advantage $Adv_{\mathcal{A}}^{CDH}$ is negligibly small.

Definition 4. ***Decisional Diffie-Hellman Problem (DDH) -*** *Given $(g, g^a,\ g^b, h) \in \mathbb{G}^4$ for unknown $a, b \in \mathbb{Z}_q^*$, where $\mathbb{G}$ is a cyclic prime order multiplicative group with g as a generator and q the order of the group, the DDH problem in $\mathbb{G}$ is to check whether $h \stackrel{?}{=} g^{ab}$.*

The advantage of any probabilistic polynomial time algorithm $\mathcal{A}$ in solving the DDH problem in $\mathbb{G}$ is defined as

$$Adv_{\mathcal{A}}^{DDH} = |Pr\left[\mathcal{A}(g, g^a, g^b, g^{ab}) = 1\right] - Pr\left[\mathcal{A}(g, g^a, g^b, h) = 1\right]| \mid a, b \in \mathbb{Z}_q^*$$

The CDH Assumption is that, for any probabilistic polynomial time algorithm $\mathcal{A}$, the advantage $Adv_{\mathcal{A}}^{CDH}$ is negligibly small.

Definition 5. ***Strong Diffie Hellman Problem (SDH) [AKO09]):*** *Let κ be the security parameter and $\mathbb{G}$ be a multiplicative group of order q, where $|q| = \kappa$. Given $(g, g^a, g^b) \in_R \mathbb{G}^3$ and access to a Decision Diffie Hellman (DDH) oracle $\mathcal{DDH}_{g,a}(.,.)$ which on input g^b and g^c outputs* `True` *if and only if $g^{ab} = g^c$, the strong Diffie Hellman problem is to compute $g^{ab} \in \mathbb{G}$ (i.e., the problem of solving Computational Diffie Hellman Problem (CDH) using a DDH oracle)*

The advantage of an adversary $\mathcal{A}$ in solving the strong Diffie Hellman problem is defined as the probability with which $\mathcal{A}$ solves the above strong Diffie Hellman problem.

$$Adv_{\mathcal{A}}^{SDHP} = Pr[\mathcal{A}(g, g^a, g^b) = g^{ab} | \mathcal{DDH}_{g,a}(.,.)]$$

The strong Diffie Hellman assumption holds in $\mathbb{G}$ if for all polynomial time adversaries $\mathcal{A}$, the advantage $Adv_{\mathcal{A}}^{SDHP}$ is negligible.

Note: In pairing groups (also known as gap groups), the DDH oracle can be efficiently instantiated and hence the strong Diffie Hellman problem is equivalent to the Gap Diffie Hellman problem [OP01].

3 ABCK security model

In this section we describe the ABCK model which is a natural extension of the CK model for PKI settings. An ABAKE consists of three polynomial time algorithms – Setup, KeyGen and KeyExchange. These algorithms are discussed below.

Setup: The setup algorithm takes as input the implicit security parameter κ and the attribute universe U and outputs the master public key MPK and master secret key MSK.

KeyGen: The key generation algorithm takes in the master secret key MSK, the master public key MPK, and a set of attributes $\mathbb{S}_{\mathbb{P}}$ given by a party P, and outputs a static secret key $SK_{\mathbb{S}_{\mathbb{P}}}$ corresponding to $\mathbb{S}_{\mathbb{P}}$.

KeyExchange: This algorithm is run between two or more users or parties in the system(in our case the number of users is two as it is two-party setting). Each party in the AB-AKE protocol executes the KeyExchange algorithm which initially takes as input the master public key MPK, an access structure $\mathbb{X}$ and a private key for a set of attributes $\mathbb{S}$. If $\mathbb{S}$ satisfies $\mathbb{X}$, KeyExchange proceeds as per specification and may generate outgoing messages and also accept incoming messages from other parties as inputs. The output of KeyExchange is either a session key Z or $\perp$. Both parties compute the same session key Z if and only if $\mathbb{S}_{\mathbb{A}} \in \mathbb{X}_B$ and $\mathbb{S}_{\mathbb{B}} \in \mathbb{X}_A$ (i.e, the attributes of one party satisfies the access structure of its peer).

Session. An instance of the protocol as described above when run at a party is called a session. The user/entity that initiates a session is called the *owner* and the other user is called the *peer.*A session is activated with an incoming message of the forms $(\mathcal{I}, \mathbb{S}_A, \mathbb{S}_B)$ or $(\mathcal{R}, \mathbb{S}_B, \mathbb{S}_A, m_1)$, where $\mathcal{I}$ and $\mathcal{R}$ with role identifiers, and A and B with user identifiers. If A was activated with $(\mathcal{I}, \mathbb{S}_A, \mathbb{S}_B)$, then A is called the session initiator. If B was activated with $(\mathcal{R}, \mathbb{S}_B, \mathbb{S}_A, m_1)$, then B is called the session responder. The components exchanged between the owner and the peer constitute the *session state*. The shared secret key obtained after exchange of components among both the parties is called the *session key*. On successful completion of a session, each entity outputs the session key and deletes the session state. Otherwise, the session is said to be in abort state and no session key is generated in this case. Each entity participating in a session assigns a unique identifier to that session. If A is the initiator of the session it sets the session id sid as $(\mathcal{I}, \mathbb{S}_A, \mathbb{S}_B, out, in)$ where out and in are respectively the components sent to B and received by A. If B is the responder of a session initiated by A, it sets the sid as $(\mathcal{R}, \mathbb{S}_B, \mathbb{S}_A, out', in')$.

Adversary. The adversary $\mathcal{A}$ is also modelled as a probabilistic polynomial time turing machine which has full control on the communication network over which protocol messages can be altered, injected or eavesdropped at any time. There are three types of adversary:

– Type I : The adversary of this type does not belong to the system and hence has access only to the PKG's parameters. It is not given access to the private keys of users and does not impersonate anyone. This is the weakest adversary.
– Type II : The adversary belongs to the attribute based system and can query for the private keys of polynomial number of users. It is not allowed to impersonate as any user.
– Type III : The adversary of this type belongs to the attribute based system and it is given access to the private keys of polynomial number of users. It can also impersonate as any other user. This is the strongest adversary and we prove our scheme secure against this type of adversary.

Since we prove our scheme secure against the Type III adversary, it is also secure against Type I and Type II because they are weaker adversaries compared to Type III. We allow the adversary to access some of the parties secret information, via the following oracle queries:

$\mathsf{Send(Message)}$: The ability of the adversary to control the communication network is modelled by the Send query. Here the adversary can send a message of the form $(\mathcal{I}, \mathbb{S}_A, \mathbb{S}_B, m)$. It sends a message m to the party A on behalf of party B and return A's response to this message to the adversary. If $m = 0$, this query makes party A to start an AKE session with B and to provide communication from B to A. Else it will send the message m from party A to party B and makes B respond to the supposed session $(\mathcal{I}, \mathbb{S}_A, \mathbb{S}_B, m, \star)$
$\mathsf{SessionStateReveal}(sid)$: The adversary $\mathcal{A}$ is given all the ephemeral secrets or the session state corresponding to the session *sid*. This could be possible if the session-specific secret information is stored in insecure memory, or if the random number generator of the party be guessed.
$\mathsf{SessionKeyReveal}(sid)$: $\mathcal{A}$ is given the session key for *sid*, provided that the session holds a session key.

$\mathsf{Party\ Corruption}(\mathbb{S}_P)$: The adversary learns the static secret key corresponding to the set of attributes $\mathbb{S}_P$.

$\mathsf{Establish}(P, \mathbb{S}_P)$: This query allows the adversary to register a set of attributes $\mathbb{S}_P$ on behalf of the party P; the adversary totally controls that party. If a party is established by $\mathsf{Establish}(P, \mathbb{S}_P)$ query issued by the adversary, then we call the party P dishonest or corrupt. If not, we call the party honest.

We now give the definition for a matching session and what it means for a session to be fresh.

Definition 6. ***(Matching Sessions).*** *Let Π be a protocol and $sid = (\zeta, \mathbb{S}_A, \mathbb{S}_B, out, in)$ and $sid' = (\zeta', \mathbb{S}_B, \mathbb{S}_A, in', out')$ be the identifier of two sessions. Then sid and sid′ are called matching (or partnered) sessions if:*

– *The attributes of user B satisfy the access structure of user A i.e. $\mathbb{S}_{\mathbb{B}} \in \mathbb{X}_A$.*
– *The attributes of user A satisfy the access structure of user B i.e. $\mathbb{S}_{\mathbb{A}} \in \mathbb{X}_B$.*
– *$out = in'$ and $in = out'$ and*
– *$\zeta \neq \zeta'$*

Definition 7. ***(Freshness).*** *A session with identifier sid is called fresh if none of the following queries by an adversary are allowed on that session sid or it's matching session sid′ (if it exists)*

– *The adversary $\mathcal{A}$ issues a $\mathsf{SessionKeyReveal}$ query on sid.*
– *The adversary $\mathcal{A}$ issues a $\mathsf{SessionStateReveal}$ query on sid.*
– *The adversary $\mathcal{A}$ issues a $\mathsf{Party\ Corruption}(\mathbb{S}_P)$ query on the party P holding the session sid or a $\mathsf{Establish}(P, \mathbb{S}_P)$ query on P.*

The adversary begins the second phase of the game by choosing a fresh session *sid** and issuing a $\mathsf{Test}(sid*)$ query, where the fresh session and test query are defined as follows:

$\mathsf{Test}(sid*)$: Here the session *sid** must be a fresh session. On the Test query, a bit $b \in \{0, 1\}$ is randomly chosen.The session key is given to the adversary $\mathcal{A}$, if $b = 0$, otherwise a uniformly chosen random value from the distribution of valid session keys is returned to $\mathcal{A}$. Only one query of this form is allowed for the adversary. Of course, after the Test query has been issued, the adversary can continue querying the oracles provided that the test session is fresh. $\mathcal{A}$ outputs his guess b' in the test session. An adversary wins the game if the selected test session is fresh and if he guesses the challenge correctly i.e., $b' = b$. The advantage of $\mathcal{A}$ in the ABAKE scheme Π is defined as

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

We now define the ABCK security definition as follows:

Definition 8. ***(ABCK security)***. *We say that an ABAKE scheme Π is secure in the ABCK model, if the following conditions hold:*

1. *If two honest parties complete matching sessions and $\mathbb{S}_A \in \mathbb{X}_B$ and $\mathbb{S}_B \in \mathbb{X}_A$, then, except with negligible probability, they both compute the same session key.*
2. *For any probabilistic polynomial-time adversary $\mathcal{A}$, $\mathsf{Adv}_{\Pi}^{ABCK}(\mathcal{A})$ is negligible.*

4 Our Construction

We now give the description of the attribute based key agreement protocol and formally prove its security in the next section.

Setup: It chooses a group $\mathbb{G}$ of prime order q. Let g be the generator of group $\mathbb{G}$. The PKG picks $s_1, s_2 \in_R {Z_p}^*$, where p divides $q-1$, sets $y_1 = g^{s_1}$ and $y_2 = g^{s_2}$. The master secret key is $\langle s_1, s_2 \rangle$ and the master public key is $\langle y_1, y_2 \rangle$. It also defines the following hash functions: $H_1 : \{0,1\}^* \to \mathbb{G}$, $H_2 : \{0,1\}^* \times \mathbb{G} \to {Z_p}^*$, $H_3 : \{0,1\}^* \times \mathbb{G} \times \mathbb{G} \times \mathbb{G} \times \mathbb{G} \to {Z_p}^*$, $H_4 : \{0,1\}^* \times \mathbb{G} \times \mathbb{G} \times \mathbb{G} \times \mathbb{G} \to {Z_p}^*$, $H_5 : \mathbb{G} \times \mathbb{G} \times \{0,1\}^* \times \{0,1\}^* \to {Z_p}^*$ and $H_6 : \mathbb{G} \times \mathbb{G} \times \mathbb{G} \to {Z_p}^*$. It then makes *params* public and keeps *msk* to itself, where *params* and *msk* are defined as follows:

$params = \langle \mathbb{G}, g, q, p, y_1, y_2, H_1, H_2, H_3, H_4, H_5, H_6 \rangle$ and $msk = \langle s_1, s_2 \rangle$.

Key Generation: On input an attribute vector $\overrightarrow{\mathbb{S}_i} = (\mathbb{S}_i^1, \mathbb{S}_i^2, \cdots, \mathbb{S}_i^{m_i})$ corresponding to an user i the PKG does the following to generate the private key of the user i :

- Chooses $\overrightarrow{x_i} \in_R {{Z_p}^*}^{m_i}$.
- Computes $\overrightarrow{u_{i1}} = g^{\overrightarrow{x_i}}$ and sets $\overrightarrow{h_i} = H_1\left(\overrightarrow{\mathbb{S}_i}\right)$.
- Computes $\overrightarrow{v_{i1}} = \overrightarrow{h_i}^{\overrightarrow{x_i}}$.
- Chooses $\overrightarrow{r_i} \in_R {{Z_p}^*}^{m_i}$, computes $\overrightarrow{u_{i2}} = g^{\overrightarrow{r_i}}$ and $\overrightarrow{v_{i2}} = \overrightarrow{h_i}^{\overrightarrow{r_i}}$.
- Sets $\overrightarrow{c_i} = H_2(\overrightarrow{u_{i1}})$, $\overrightarrow{b_i} = H_3(\overrightarrow{u_{i1}}, \overrightarrow{v_{i1}}, \overrightarrow{u_{i2}}, \overrightarrow{v_{i2}})$ and $\overrightarrow{e_i} = H_4(\overrightarrow{u_{i1}}, \overrightarrow{v_{i1}}, \overrightarrow{u_{i2}}, \overrightarrow{v_{i2}})$.
- Computes $\overrightarrow{d_{i1}} = \overrightarrow{x_i} + s_1 \overrightarrow{c_i}$ where s_1 is the master secret key. It also calculates $\overrightarrow{d_{i2}} = \overrightarrow{x_i} + \overrightarrow{r_i}\,\overrightarrow{b_i} + s_2 \overrightarrow{e_i}$.
- Finally it sends $\langle \overrightarrow{u_{i1}}, \overrightarrow{v_{i1}}, \overrightarrow{u_{i2}}, \overrightarrow{v_{i2}}, \overrightarrow{d_{i1}}, \overrightarrow{d_{i2}}, \overrightarrow{h_i}^{s_2} \rangle$ to the user i.

The user after receiving the private key components from the PKG performs the checks as described (Key Sanity Check) to ensure the correctness of the components.

Secret Key Sanity Check: After receiving the private key from the PKG in the key extract phase, the user performs the following check to ensure the correctness of the components of the private key. The user first computes the following and then performs three checks as follows:

a. $\overrightarrow{c_i} = H_2\left(\overrightarrow{u_{i1}}\right)$

b. $\overrightarrow{b_i} = H_3\left(\overrightarrow{u_{i1}}, \overrightarrow{v_{i1}}, \overrightarrow{u_{i2}}, \overrightarrow{v_{i2}}\right)$

c. $\overrightarrow{e_i} = H_4\left(\overrightarrow{u_{i1}}, \overrightarrow{v_{i1}}, \overrightarrow{u_{i2}}, \overrightarrow{v_{i2}}\right)$

Test 1: Check if $\dfrac{g^{\overrightarrow{d_{i1}}}}{{y_1}^{H_2\left(\overrightarrow{u_{i1}}\right)}} \stackrel{?}{=} \overrightarrow{u_{i1}}$.

This can be verified as $\dfrac{g^{\overrightarrow{x_i}+s_1\cdot\overrightarrow{c_i}}}{g^{s_1\cdot H_2\left(\overrightarrow{u_{i1}}\right)}}$ where $\overrightarrow{c_i} = H_2\left(\overrightarrow{u_{i1}}\right)$. This is equal to $g^{\overrightarrow{x_i}} = \overrightarrow{u_{i1}}$. This check ensures the correctness of $\overrightarrow{d_{i1}}$ and $\overrightarrow{u_{i1}}$.

Test 2: Check if $\dfrac{g^{\overrightarrow{d_{i2}}}}{\overrightarrow{u_{i2}}^{H_3\left(\overrightarrow{u_{i1}},\overrightarrow{v_{i1}},\overrightarrow{u_{i2}},\overrightarrow{v_{i2}}\right)}.{y_2}^{H_4\left(\overrightarrow{u_{i1}},\overrightarrow{v_{i1}},\overrightarrow{u_{i2}},\overrightarrow{v_{i2}}\right)}} \stackrel{?}{=} \overrightarrow{u_{i1}}$.

This can be verified as $\dfrac{g^{\left(\overrightarrow{x_i}+\overrightarrow{r_i}\cdot\overrightarrow{b_i}+s_2\cdot\overrightarrow{e_i}\right)}}{g^{\overrightarrow{r_i}\cdot H_3\left(\overrightarrow{u_{i1}},\overrightarrow{v_{i1}},\overrightarrow{u_{i2}},\overrightarrow{v_{i2}}\right)}\cdot g^{s_2\cdot H_4\left(\overrightarrow{u_{i1}},\overrightarrow{v_{i1}},\overrightarrow{u_{i2}},\overrightarrow{v_{i2}}\right)}} \stackrel{?}{=} g^{\overrightarrow{x_i}} = \overrightarrow{u_{i1}}$, as $\overrightarrow{b_i} = H_3\left(\overrightarrow{u_{i1}}, \overrightarrow{v_{i1}}, \overrightarrow{u_{i2}}, \overrightarrow{v_{i2}}\right)$ and $\overrightarrow{e_i} = H_4\left(\overrightarrow{u_{i1}}, \overrightarrow{v_{i1}}, \overrightarrow{u_{i2}}, \overrightarrow{v_{i2}}\right)$.

This check ensures the correctness of $\overrightarrow{d_{i2}}, \overrightarrow{u_{i2}}, \overrightarrow{v_{i1}}, \overrightarrow{v_{i2}}$.

Test 3 : Check if $\dfrac{\overrightarrow{h_i}^{\overrightarrow{d_{i2}}}}{\overrightarrow{v_{i2}}^{H_3\left(\overrightarrow{u_{i1}},\overrightarrow{v_{i1}},\overrightarrow{u_{i2}},\overrightarrow{v_{i2}}\right)}.\left(\overrightarrow{h_i}^{s_2}\right)^{H_4\left(\overrightarrow{u_{i1}},\overrightarrow{v_{i1}},\overrightarrow{u_{i2}},\overrightarrow{v_{i2}}\right)}} = \overrightarrow{v_{i1}}$.

This can be verified as $\dfrac{\overrightarrow{h_i}^{\overrightarrow{x_i}+\overrightarrow{r_i}.\overrightarrow{b_i}+s_2.\overrightarrow{e_i}}}{\left(\overrightarrow{h_i}^{\overrightarrow{r_i}}\right)^{H_3\left(\overrightarrow{u_{i1}},\overrightarrow{v_{i1}},\overrightarrow{u_{i2}},v_{i2}\right)}.\left(\overrightarrow{h_i}^{s_2}\right)^{H_4\left(\overrightarrow{u_{i1}},\overrightarrow{v_{i1}},\overrightarrow{u_{i2}},\overrightarrow{v_{i2}}\right)}} = \overrightarrow{h_i}^{\overrightarrow{x_i}} = \overrightarrow{v_{i1}}$ where $\overrightarrow{b_i} = H_3(\overrightarrow{u_{i1}}, \overrightarrow{v_{i1}}, \overrightarrow{u_{i2}}, \overrightarrow{v_{i2}})$ and $\overrightarrow{e_i} = H_4\left(\overrightarrow{u_{i1}}, \overrightarrow{v_{i1}}, \overrightarrow{u_{i2}}, \overrightarrow{v_{i2}}\right)$.

Test 3 ensures the correctness of $\overrightarrow{h_i}^{s_2}$. Test 2 and Test 3 ensures that g and $\overrightarrow{h_i}$ are raised to the same exponent $\overrightarrow{x_i}$ in $\overrightarrow{u_{i1}}$ and $\overrightarrow{v_{i1}}$ respectively.

If the received private key satisfies all the tests then it is valid.

Key Agreement: The two users i and j with attribute vectors $\overrightarrow{\mathbb{S}}_i$ and $\overrightarrow{\mathbb{S}}_j$ get their respective private keys from the PKG. First, i decides an access structure $\mathbb{X}_i$ and he hopes that the set of attributes $\overrightarrow{\mathbb{S}}_j$ of j satisfies $\mathbb{X}_i$. Then, i derives the $l_i \times n_i$ share generating matrix M_i and the injective labeling function ρ_i in a LSSS for $\mathbb{X}_i$. Similarly, j also decides an access structure $\mathbb{X}_j$ and he hopes that the set of attributes $\mathbb{S}_i$ of i satisfies $\mathbb{X}_j$. It then derives the $l_j \times n_j$ share generating matrix M_j and the injective labeling function ρ_j in a LSSS for $\mathbb{X}_j$. i and j then choose ephemeral secret components $\overrightarrow{t_i}, \overrightarrow{w_i} \in_R ({Z_p}^*)^{l_i}$ and $\overrightarrow{t_j}, \overrightarrow{w_j} \in_R ({Z_p}^*)^{l_j}$ respectively. i (respectively j) also chooses a random vector $\overrightarrow{\sigma_i} \in_R {Z_p}^{*l_i}$ (respectively $\overrightarrow{\sigma_j} \in_R {Z_p}^{*l_j}$) and engage in a session as described in Table 1. User k ($k \in i, j$) also computes the values $\overrightarrow{\overline{c}_k} = H_2\left(\overrightarrow{\mathbb{A}_k}, \overrightarrow{u_{k1}}\right)$, $\overrightarrow{\overline{b}_k} = H_3\left(\overrightarrow{\mathbb{A}_k}, , \overrightarrow{u_{k1}}, \overrightarrow{v_{k1}}, \overrightarrow{u_{k2}}, \overrightarrow{v_{k2}}\right)$ and $\overrightarrow{\overline{e}_k} = H_4\left(\overrightarrow{\mathbb{A}_k}, \overrightarrow{u_{k1}}, \overrightarrow{v_{k1}}, \overrightarrow{u_{k2}}, \overrightarrow{v_{k2}}\right)$, where $\overrightarrow{\mathbb{A}_k}$ is the attribute vector that user k wants its peer k' to satisfy.

Note that user j (respectively i) does not know the corresponding attributes of i (j),i.e., $\overrightarrow{\mathbb{S}}_i$ (respectively $\overrightarrow{\mathbb{S}}_j$), but he can apply the function ρ_i (respectively ρ_j) to the rows of the share generating matrix M_i (respectively M_j) sent by i (respectively j) to get back the corresponding attribute vector $\overrightarrow{\mathbb{A}_i}$ corresponding to the access structure $\mathbb{X}_i$ (respectively $\mathbb{X}_j$) LSSS matrix if the attributes of user j satisfies $\mathbb{X}_i$. It can then proceed with the computation of our key agreement protocol with the attributes $\overrightarrow{\mathbb{A}_i}$ (respectively $\overrightarrow{\mathbb{A}_j}$) corresponding to the rows of M_i (respectively M_j).

Remark 1 : The values in $\overrightarrow{V_i}$ are freshly generated for every session in the following manner. In a preprocessing or a setup stage, the user i generates a large number of $\left(\beta, g^{\beta}\right)$ pairs and stores them in a table T_i.

User i	User j
1. Send $\overrightarrow{F_i} = \left\langle \overrightarrow{u_{i1}}, \overrightarrow{v_{i1}}, \overrightarrow{d_{i2}}, \overrightarrow{b_i}, \overrightarrow{e_i}, \overrightarrow{\widetilde{c_i}}, \overrightarrow{\widetilde{b_i}}, \overrightarrow{\widetilde{e_i}}, \overrightarrow{h_i}^{s_2}, M_i, \rho_i \right\rangle$, $\overrightarrow{V_i} = \left\langle \overrightarrow{w_i} + \overrightarrow{d_{i1}} \cdot H_5\left(g^{\overrightarrow{t_i}}, g^{\overrightarrow{w_i}}, M_i, \rho_i\right), g^{\overrightarrow{t_i}}, g^{\overrightarrow{w_i}} \right\rangle$, $X_i^{(1)} = g^{\sigma_i^{(1)}}$, $U_i^{(ab)} = g^{M_{i_{ab}} \sigma_i^a}$ to j.	1. Send $\overrightarrow{F_j} = \left\langle \overrightarrow{u_{j1}}, \overrightarrow{v_{j1}}, \overrightarrow{d_{j2}}, \overrightarrow{b_j}, \overrightarrow{e_j}, \overrightarrow{\widetilde{c_i}}, \overrightarrow{\widetilde{b_j}}, \overrightarrow{\widetilde{e_j}}, \overrightarrow{h_j}^{s_2}, M_j, \rho_j \right\rangle$, $\overrightarrow{V_j} = \left\langle \overrightarrow{w_j} + \overrightarrow{d_{j1}} \cdot H_5\left(g^{\overrightarrow{t_j}}, g^{\overrightarrow{w_j}}, M_j, \rho_j\right), g^{\overrightarrow{t_j}}, g^{\overrightarrow{w_j}} \right\rangle$, $X_j^{(1)} = g^{\sigma_j^{(1)}}$, $U_j^{(ab)} = g^{M_{j_{ab}} \sigma_j^a}$ to i.
2. (a) ***Check for correctness of*** $\overrightarrow{F_j}$***:*** Compute $\overrightarrow{u_{j2}} = \left(\frac{g^{\overrightarrow{d_{j2}}}}{\overrightarrow{u_{j1}}.{y_2}^{\overrightarrow{e_j}}} \right)^{\overrightarrow{b_j}^{-1}}$ Compute $\overrightarrow{v_{j2}} = \left(\frac{\overrightarrow{h_j}^{\overrightarrow{d_{j2}}}}{\overrightarrow{v_{j1}} \cdot \left(\overrightarrow{h_j}^{s_2}\right)^{\overrightarrow{e_j}}} \right)^{\overrightarrow{b_j}^{-1}}$ ***Check 1 :*** Check if $\overrightarrow{\widetilde{c_j}} = H_2\left(\overrightarrow{\mathbb{A}_j}, \overrightarrow{u_{j1}}\right)$ $\overrightarrow{\widetilde{b_j}} \stackrel{?}{=} H_3(\overrightarrow{\mathbb{A}}_j, \overrightarrow{u_{j1}}, \overrightarrow{v_{j1}}, \overrightarrow{u_{j2}}, \overrightarrow{v_{j2}})$ $\overrightarrow{\widetilde{e_j}} \stackrel{?}{=} H_4(\overrightarrow{\mathbb{A}}_j, \overrightarrow{u_{j1}}, \overrightarrow{v_{j1}}, \overrightarrow{u_{j2}}, \overrightarrow{v_{j2}})$ If not equal abort, else proceed. (b) ***Check for correctness of*** $\overrightarrow{V_j}$***:*** ***Check 2 :*** Check if $\left[\frac{g^{\left(\overrightarrow{w_j} + \overrightarrow{d_{j1}} \cdot H_5\left(g^{\overrightarrow{t_j}}, g^{\overrightarrow{w_j}}, M_j, \rho_j\right)\right)}}{\left(g^{\overrightarrow{x_j}}\right)^{H_5\left(g^{\overrightarrow{t_j}}, g^{\overrightarrow{w_j}}, M_j, \rho_j\right)} (y_1)^{\overrightarrow{c_j} \cdot H_5\left(g^{\overrightarrow{t_j}}, g^{\overrightarrow{w_j}}, M_j, \rho_j\right)}} \right] \stackrel{?}{=} g^{\overrightarrow{w_j}}$ where $\overrightarrow{c_j} = H_2\left(\overrightarrow{u_{j1}}\right)$. If not equal abort, else proceed. ***Check 3 :*** Check if $\prod_{a,b:\rho_j(a) \in \mathbb{S}_i} (U_j^{(ab)})^{\lambda_i^a} \stackrel{?}{=} X_j^{(1)}$. If equal proceed to step 3, else abort.	2. (a) ***Check for correctness of*** $\overrightarrow{F_i}$***:*** Compute $\overrightarrow{u_{i2}} = \left(\frac{g^{\overrightarrow{d_{i2}}}}{\overrightarrow{u_{i1}} \cdot {y_2}^{\overrightarrow{e_i}}} \right)^{\overrightarrow{b_i}^{-1}}$ Compute $\overrightarrow{v_{i2}} = \left(\frac{\overrightarrow{h_i}^{\overrightarrow{d_{i2}}}}{\overrightarrow{v_{i1}} \cdot \left(\overrightarrow{h_i}^{s_2}\right)^{\overrightarrow{e_i}}} \right)^{\overrightarrow{b_i}^{-1}}$ ***Check 1 :*** Check if $\overrightarrow{\widetilde{c_i}} = H_2\left(\overrightarrow{\mathbb{A}_i}, \overrightarrow{u_{i1}}\right)$ $\overrightarrow{\widetilde{b_i}} \stackrel{?}{=} H_3\left(\overrightarrow{\mathbb{A}}_i, \overrightarrow{u_{i1}}, \overrightarrow{v_{i1}}, \overrightarrow{u_{i2}}, \overrightarrow{v_{i2}}\right)$ $\overrightarrow{\widetilde{e_i}} \stackrel{?}{=} H_4\left(\overrightarrow{\mathbb{A}}_i, \overrightarrow{u_{i1}}, \overrightarrow{v_{i1}}, \overrightarrow{u_{i2}}, \overrightarrow{v_{i2}}\right)$ If not equal abort, else proceed. (b) ***Check for correctness of*** $\overrightarrow{V_i}$***:*** ***Check 2 :*** Check if $\left[\frac{g^{\left(\overrightarrow{w_i} + \overrightarrow{d_{i1}} . H_5\left(g^{\overrightarrow{t_i}}, g^{\overrightarrow{w_i}}, M_i, \rho_i\right)\right)}}{\left(g^{\overrightarrow{x_i}}\right)^{H_5\left(g^{\overrightarrow{t_i}}, g^{\overrightarrow{w_i}}, M_i, \rho_i\right)} (y_1)^{\overrightarrow{c_i} \cdot H_5\left(g^{\overrightarrow{t_i}}, g^{\overrightarrow{w_i}}, M_i, \rho_i\right)}} \right] \stackrel{?}{=} g^{\overrightarrow{w_i}}$ where $\overrightarrow{c_i} = H_2\left(\overrightarrow{u_{i1}}\right)$. If not equal abort, else proceed. ***Check 3 :*** Check if $\prod_{a,b:\rho_i(a) \in \mathbb{S}_j} (U_i^{(ab)})^{\lambda_j^a} \stackrel{?}{=} X_i^{(1)}$. If equal proceed to step 3, else abort.
3. ***Shared secret key generation:*** Compute $\overrightarrow{Z_1} = \left(\overrightarrow{u_{j1}} {y_1}^{\overrightarrow{c_j}} g^{\overrightarrow{t_j}}\right)^{\overrightarrow{d_{i1}} + \overrightarrow{t_i}}$ $\overrightarrow{Z_2} = \overrightarrow{v_{i1}} \overrightarrow{v_{j1}}$ $\overrightarrow{Z_3} = \left(g^{\overrightarrow{t_j}}\right)^{\overrightarrow{t_i}}$. $\overrightarrow{Z} = H_6\left(\overrightarrow{Z_1}, \overrightarrow{Z_2}, \overrightarrow{Z_3}\right)$.	3. ***Shared secret key generation:*** Compute $\overrightarrow{Z_1} = \left(\overrightarrow{u_{i1}} {y_1}^{\overrightarrow{c_i}} g^{\overrightarrow{t_i}}\right)^{\overrightarrow{d_{j1}} + \overrightarrow{t_j}}$ $\overrightarrow{Z_2} = \overrightarrow{v_{j1}} \overrightarrow{v_{i1}}$ $\overrightarrow{Z_3} = \left(g^{\overrightarrow{t_i}}\right)^{\overrightarrow{t_j}}$. $\overrightarrow{Z} = H_6\left(\overrightarrow{Z_1}, \overrightarrow{Z_2}, \overrightarrow{Z_3}\right)$.

$\overrightarrow{Z}$ is the shared secret key that is established between User i and User j.

Table 1. Description of the Key Agreement protocol

For each session, user i extracts 2 fresh vectors each of length l_i from the table T_i and uses them to generate components of $\overrightarrow{V_i}$. For security reasons, we assume that

(a) immediately after generating the components of $\overrightarrow{V_i}$, $\overrightarrow{w_i}$ is erased from the system.

(b) $\overrightarrow{w_i} + \overrightarrow{d_{i1}} \cdot H_5\left(g^{\overrightarrow{t_i}}, g^{\overrightarrow{w_i}}, M_i, \rho_i\right)$ is computed in a secured way so that $\overrightarrow{w_i}$ and $\overrightarrow{d_{i1}}$ are not leaked to the adversary and only $\overrightarrow{w_i} + \overrightarrow{d_{i1}} \cdot H_5\left(g^{\overrightarrow{t_i}}, g^{\overrightarrow{w_i}}, M_i, \rho_i\right)$ is available to the adversary.

Remark 2 : The components in $\overrightarrow{F_i}, \overrightarrow{V_i}$ and $\overrightarrow{F_j}, \overrightarrow{V_j}$ is required to be sent only for the first time key establishment between users i and j. For subsequent key establishments between i and j, only $\overrightarrow{\overrightarrow{c}_i}$, $\overrightarrow{\overrightarrow{b}_i}$, $\overrightarrow{\overrightarrow{e}_i}$, $\overrightarrow{V_i}$ and $\overrightarrow{\overrightarrow{c}_j}$, $\overrightarrow{\overrightarrow{b}_j}$, $\overrightarrow{\overrightarrow{e}_j}$,$\overrightarrow{V_j}$ need to be exchanged since all other values in $\overrightarrow{F_i}$ are same for all sessions between a pair of users and is independent of the session.

Remark 3 : The intuition behind using the component $\overrightarrow{Z_3}$ is to eliminate $g^{\overrightarrow{t_i}\cdot\overrightarrow{t_j}}$ from $\overrightarrow{Z_1}$ in the security proof to obtain the solution to the hard problem.

Remark 4 : Note that none of the users need to know the attributes of its peer for running the key agreement protocol. If the attributes of a user satisfies the access structure of its peer, then it can extract those attributes that its peer wants it to be get satisfied namely the $\overrightarrow{\mathbb{A}_k}$ values where $k \in \{i, j\}$ (applying the function ρ_k on the rows of the access matrix M_k) and then it perform ***Check 1*** and ***Check 2*** in our construction. If the attributes of the peer indeed satisfies the access structure it can extract the $\overrightarrow{\mathbb{A}_k}$ vector and proceed with the checks.

Check 1 is done to ensure that g and $\overrightarrow{h_i}$ are raised to the same exponent $\overrightarrow{x_i}$. This is a crucial security requirement.

For valid components this check holds good. We prove it here. For the ease of understanding we will prove it for one component of the corresponding vectors. The same check goes through for all the components of the corresponding vectors.

$$\left(\frac{g^{\overrightarrow{d_{i2}}}}{\overrightarrow{u_{i1}}.y_2{}^{\overrightarrow{e_i}}}\right)^{\overrightarrow{b_i}^{-1}} = \left(\frac{g^{\overrightarrow{x_i}+\overrightarrow{r_i}.\overrightarrow{b_i}+s_2.\overrightarrow{e_i}}}{g^{\overrightarrow{x_i}}\cdot g^{s_2\cdot\overrightarrow{e_i}}}\right)^{\overrightarrow{b_i}^{-1}} = \left(g^{\overrightarrow{r_i}\cdot\overrightarrow{b_i}}\right)^{\overrightarrow{b_i}^{-1}} = g^{\overrightarrow{r_i}} = \overrightarrow{u_{i2}}.$$

$$\left(\frac{\overrightarrow{h_i}^{\overrightarrow{d_{i2}}}}{\overrightarrow{v_{i1}}\cdot\left(\overrightarrow{h_i}^{s_2}\right)^{\overrightarrow{e_i}}}\right)^{\overrightarrow{b_i}^{-1}} = \left(\frac{\overrightarrow{h_i}^{\overrightarrow{x_i}+\overrightarrow{r_i}\cdot\overrightarrow{b_i}+s_2\cdot\overrightarrow{e_i}}}{\overrightarrow{h_i}^{\overrightarrow{x_i}}\cdot\left(\overrightarrow{h_i}^{s_2}\right)^{\overrightarrow{e_i}}}\right)^{\overrightarrow{b_i}^{-1}} = \left(\overrightarrow{h_i}^{\overrightarrow{r_i}\cdot\overrightarrow{b_i}}\right)^{\overrightarrow{b_i}^{-1}} = \overrightarrow{h_i}^{\overrightarrow{r_i}} = \overrightarrow{v_{i2}}$$

The components that are recomputed are valid and if the attributes of user j satisfies the access structure of user i, it can get the attribute vector $\mathbb{A}_i$ and hence the computation of $\overrightarrow{\overrightarrow{b}_i} = H_3\left(\overrightarrow{\mathbb{A}}_i, \overrightarrow{u_{i1}}, \overrightarrow{v_{i1}}, \overrightarrow{u_{i2}}, \overrightarrow{v_{i2}}\right)$ and $\overrightarrow{\overrightarrow{e}_i} = H_3\left(\overrightarrow{\mathbb{A}}_i, \overrightarrow{u_{i1}}, \overrightarrow{v_{i1}}, \overrightarrow{u_{i2}}, \overrightarrow{v_{i2}}\right)$ will match the one obtained if not for any tampering during transfer.

Check 2 is done to ensure that a dynamic adversary cannot tamper the components exchanged and affect the shared secret key generation. It verifies the signature $w_i + d_{i1} \cdot H_5\left(g^{t_i}, g^{w_i}, M_i, \rho_i\right)$ on g^{t_i}.

$$\frac{g^{\left(\overrightarrow{w_i}+\overrightarrow{d_{i1}}\cdot H_5\left(g^{\overrightarrow{t_i}},g^{\overrightarrow{w_i}},M_i,\rho_i\right)\right)}}{\left(g^{\overrightarrow{x_i}}\right)^{H_5\left(g^{\overrightarrow{t_i}},g^{\overrightarrow{w_i}},M_i,\rho_i\right)}.(y_1)^{\overrightarrow{c_i}\cdot H_5\left(g^{\overrightarrow{t_i}},g^{\overrightarrow{w_i}},M_i,\rho_i\right)}} = \frac{g^{\left(\overrightarrow{w_i}+\left(\overrightarrow{x_i}+s_1\cdot\overrightarrow{c_i}\right)\cdot H_5\left(g^{\overrightarrow{t_i}},g^{\overrightarrow{w_i}},M_i,\rho_i\right)\right)}}{\left(g^{\overrightarrow{x_i}}\right)^{H_5\left(g^{\overrightarrow{t_i}},g^{\overrightarrow{w_i}},\right)}\cdot(g)^{s_1\cdot\overrightarrow{c_i}\cdot H_5\left(g^{\overrightarrow{t_i}},g^{\overrightarrow{w_i}},M_i,\rho_i\right)}} = g^{\overrightarrow{w_i}}$$

Check 3 is done to ensure that a dynamic adversary cannot tamper with the access structure. If the adversary tries to tamper with the acess structure, he will be caught since the share generating matrix and the function ρ_i is bound to the hash function H_5. If the attributes of user j i.e. $\mathbb{S}_j$ satisfies the access structure of i, it can generate the constants λ_j's $\in \mathbb{Z}_p$ such that for any valid shares $\{\omega_i\}_{i\in I}$ of a secret $\sigma_i^{(1)}$ according to the LSSS scheme Π, we have: $\sum_{i\in I}\omega_i\lambda_i=\sigma_i^{(1)}$ where $I=\{i|\rho_i(a)\in S\}$. So:

$$\prod_{a,b:\rho_i(a)\in\mathbb{S}_j}(U_i^{(ab)})^{\lambda_j^a}=\prod_{a,b:\rho_i(a)\in\mathbb{S}_j}\left(g^{M_{i_{ab}}\sigma_i^a}\right)^{\lambda_j^a}=g^{\sum_{a,b:\rho_i(a)\in\mathbb{S}_j}M_{i_{ab}}\sigma_i^a\lambda_j^a}=g^{\sigma_i^{(1)}}=X_i^{(1)}.$$

Lemma 1: The shared secret key computed by both the parties are identical.

Proof: User i computes :
$\overrightarrow{Z_1}=\left(\overrightarrow{u_{j1}}\overrightarrow{y_1}^{\overrightarrow{c_j}}g^{\overrightarrow{t_j}}\right)^{\overrightarrow{d_{i1}}+\overrightarrow{t_i}}=\left(g^{\left(\overrightarrow{x_j}+s_1\overrightarrow{c_j}+\overrightarrow{t_j}\right)}\right)^{\left(\overrightarrow{d_{i1}}+\overrightarrow{t_i}\right)}=g^{\left(\overrightarrow{d_{j1}}+\overrightarrow{t_j}\right)\left(\overrightarrow{d_{i1}}+\overrightarrow{t_i}\right)}$, since $\overrightarrow{u_{j1}}=g^{\overrightarrow{x_j}}$ and $\overrightarrow{x_j}+s_1\overrightarrow{c_j}=\overrightarrow{d_{j1}}$.
User j computes:
$\overrightarrow{Z_1}=\left(\overrightarrow{u_{i1}}y_1^{\overrightarrow{c_i}}g^{t_i}\right)^{d_{j1}+t_j}=\left(g^{\left(\overrightarrow{x_i}+s_1\overrightarrow{c_i}+\overrightarrow{t_i}\right)}\right)^{\left(\overrightarrow{d_{j1}}+\overrightarrow{t_j}\right)}=g^{\left(\overrightarrow{d_{i1}}+\overrightarrow{t_i}\right)\left(\overrightarrow{d_{j1}}+\overrightarrow{t_j}\right)}$, since $\overrightarrow{u_{i1}}=g^{\overrightarrow{x_i}}$ and $\overrightarrow{x_i}+s_1\overrightarrow{c_i}=\overrightarrow{d_{i1}}$.
Thus $\overrightarrow{Z_1}$ computed by both the parties are identical. $\overrightarrow{Z_2}$ and $\overrightarrow{Z_3}$ are also consistent. Thus the final shared secret key computed by both the parties are consistent. □

5 Security Proof

In this section, we present a formal security proof for the protocol described in the previous section. The proof is based on the ABCK security model described in section 3. The scheme is proved secure under the Gap Diffie-Hellman (GDH) assumption in the random oracle model. The security proof is modeled as a game between the challenger and the adversary.

Theorem 1. *Under the GDH assumption in $\mathbb{G}$ and the RO model,the protocol in section 4 is ABCK-secure.*

Proof of Theorem 1.
The proof of Theorem 1 is shown in the Appendix.

6 Additional Security Properties

The proposed protocol offers additional security properties which we discuss informally. Formal details of these properties can be found in the full version of the paper.

Forward Secrecy: A key agreement protocol has forward secrecy, if after a session is completed and its shared secret key is erased, the adversary cannot learn it even if it corrupts the parties involved in that session. In other words, learning the private keys of parties should not affect the security of the shared secret key. Relaxing the definition of forward secrecy, we assume that the past sessions with passive adversary are the ones whose shared secret keys are not compromised. The proposed scheme offers forward secrecy.

Resistance to Key Compromise Impersonation attacks: Whenever a user i's private key is learned by the adversary, it can impersonate as i. A key compromise impersonation (KCI) attack can be carried out when the knowledge of i's private key allows the adversary to impersonate another party to i. Our scheme is resistant to KCI attacks. This is because in the proof, when the adversary tries to impersonate i to user j, the challenger is able to answer private key queries from the adversary corresponding to user j. Thus the resistance to KCI attacks is inbuilt in the security proof.

Resistance to Ephemeral Key Compromise Impersonation: Generally the users pick the ephemeral keys $\left(\overrightarrow{t_i},g^{\overrightarrow{t_i}}\right)$ from a pre-computed list in order to minimize online computation cost. But the problem with this approach is that the ephemeral components may be subjected to leakage. This attack considers the case when the adversary can make state-reveal queries even in the test session. But our scheme is resistant to that type of an attack because when an adversary tries to impersonate a user j without knowing the private key of j, it cannot generate the components $\overrightarrow{d_{j2}}$ and the signature on $g^{\overrightarrow{t_j}}$ (We assume that $\overrightarrow{w_i}$ is erased immediately after the signature on $g^{\overrightarrow{t_i}}$ is computed and hence is not available to the adversary during state-reveal queries). Thus it is secure and resists ephemeral key compromise impersonation attack.

7 Conclusion

The main advantages of our protocol is that it requires only one round of communication among the users and the messages can be scheduled arbitarily. Moreover our scheme also provides protection against active adversaries and also does not rely on any underlying attribute based encryption scheme as a key exchange problem should be fundamentally more simpler than any encryption scheme. Also our scheme enjoys the property of having constant size public parameters. Moreover our proof techniques can be easily modified to achieve security in attribute based eCK model. We leave open the problem of designing ABAKE scheme in more stronger model that allows arbitraly leakages of intermediate values as in seCK [SEVB10] model and also designing ABAKE schemes in standard model.

References

[AKB07] Giuseppe Ateniese, Jonathan Kirsch, and Marina Blanton. Secret handshakes with dynamic and fuzzy matching. In *NDSS*, volume 7, pages 1–19, 2007.

[AKO09] Masayuki Abe, Eike Kiltz, and Tatsuaki Okamoto. Compact cca-secure encryption for messages of arbitrary length. In *Public Key Cryptography–PKC 2009*, pages 377–392. Springer, 2009.

[Bei96] Amos Beimel. *Secure schemes for secret sharing and key distribution.* PhD thesis, Technion-Israel Institute of technology, Faculty of computer science, 1996.

[BR94] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In *Advances in CryptologyCRYPTO93*, pages 232–249. Springer, 1994.

[BS10] James Birkett and Douglas Stebila. Predicate-based key exchange. In *Information Security and Privacy*, pages 282–299. Springer, 2010.

[BSW07] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*, pages 321–334. IEEE, 2007.

[GBN10] M Choudary Gorantla, Colin Boyd, and Juan Manuel González Nieto. Attribute-based authenticated key exchange. In *Information Security and Privacy*, pages 300–317. Springer, 2010.

[GJPS08] Vipul Goyal, Abhishek Jain, Omkant Pandey, and Amit Sahai. Bounded ciphertext policy attribute based encryption. In *Automata, languages and programming*, pages 579–591. Springer, 2008.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 89–98. Acm, 2006.

[LLM07] Brian LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. In *Provable Security*, pages 1–16. Springer, 2007.

[LOS+10] Allison Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In *Advances in Cryptology–EUROCRYPT 2010*, pages 62–91. Springer, 2010.

[OP01] Tatsuaki Okamoto and David Pointcheval. The gap-problems: A new class of problems for the security of cryptographic schemes. In *Public Key Cryptography*, pages 104–118. Springer, 2001.

[OSW07] Rafail Ostrovsky, Amit Sahai, and Brent Waters. Attribute-based encryption with non-monotonic access structures. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 195–203. ACM, 2007.

[OT10] Tatsuaki Okamoto and Katsuyuki Takashima. Fully secure functional encryption with general relations from the decisional linear assumption. In *Advances in Cryptology–CRYPTO 2010*, pages 191–208. Springer, 2010.

[SEVB10] Augustin P Sarr, Philippe Elbaz-Vincent, and Jean-Claude Bajard. A new security model for authenticated key agreement. In *Security and Cryptography for Networks*, pages 219–234. Springer, 2010.

[SW05] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *Advances in Cryptology–EUROCRYPT 2005*, pages 457–473. Springer, 2005.

[VSVR13] S Sree Vivek, S Sharmila Deva Selvi, Layamrudhaa Renganathan Venkatesan, and C Pandu Rangan. Efficient, pairing-free, authenticated identity based key agreement in a single round. In *Provable Security*, pages 38–58. Springer, 2013.

[Wat11] Brent Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In *Public Key Cryptography–PKC 2011*, pages 53–70. Springer, 2011.

[Yon10] Kazuki Yoneyama. Strongly secure two-pass attribute-based authenticated key exchange. In *Pairing-Based Cryptography-Pairing 2010*, pages 147–166. Springer, 2010.

Appendix

Security Proof.
We now give the security proof for our protocol from section 4.

Setup: The challenger is given the SDH problem instance $\langle \mathbb{G}, g, q, p, C = g^a, D = g^b \rangle$ and access to the Diffie Hellman Oracle $DH(y_1, ., .)$. The challenger sets the master public key $y_1 = C$ and hence the master secret key s_1 is implicitly set as a. The challenger chooses $s_2 \in_R Z_p{}^*$ and sets $y_2 = g^{s_2}$. The challenger gives the tuple $\langle \mathbb{G}, g, q, p, y_1, y_2 \rangle$ to the adversary. The challenger simulates the hash oracles in the following way:

H_1 *Oracle:* The challenger is queried by the adversary for the hash value of the attribute vector $\overrightarrow{\mathbb{S}_i}$ corresponding to user i. If the H_1 *Oracle* was already queried with $\overrightarrow{\mathbb{S}_i}$ as input, the challenger returns the value computed before which is stored in the hash list L_{h1} described below. Otherwise the challenger tosses a *coin* $\tau_i^{(\gamma)}$ where the $Pr\left(\tau_i^{(\gamma)} = 0\right) = \alpha$. The output of this oracle is defined as:

$$\forall \gamma, \quad h_i^{(\gamma)} = \begin{cases} g^{k_i}, & if\ \tau_i^{(\gamma)} = 0 \\ \left(g^b\right)^{k_i}, & if\ \tau_i^{(\gamma)} = 1 \end{cases}$$

where $\overrightarrow{k_i} \in_R \mathbb{Z}_p{}^*$. The challenger makes an entry in the hash list $L_{h1} = \left\langle \overrightarrow{h_i}, \overrightarrow{\mathbb{S}_i}, \overrightarrow{\tau_i}, \overrightarrow{k_i} \right\rangle$ for future use and returns $\overrightarrow{h_i}$.

H_2 *Oracle* : The adversary queries the challenger with inputs $(\overrightarrow{u_{i1}})$ or $\left(\overrightarrow{\mathbb{A}_i}, \overrightarrow{u_{i1}}\right)$. If the H_2 *Oracle* was already queried with $\left(\overrightarrow{\mathbb{A}_i}, \overrightarrow{u_{i1}}\right)$ as input, the challenger extracts the value $\overrightarrow{\overrightarrow{c_i}}$ from the hash list L_{h2} described below and returns the value. If the H_2 *Oracle* was already queried with $(\overrightarrow{u_{i1}})$ as input, the challenger extracts the value $\overrightarrow{c_i}$ from the hash list L_{h2} described below and returns the value. Otherwise, the challenger chooses a random vector $\overrightarrow{\overrightarrow{c_i}} \in_R \mathbb{Z}_p{}^{*m_i}$ or $\overrightarrow{c_i} \in_R \mathbb{Z}_p{}^{*m_i}$ respectively. It makes an entry in the hash list $L_{h2} = \left\langle \overrightarrow{c_i}, \overrightarrow{u_{i1}}, \overrightarrow{\mathbb{A}_i} \right\rangle$ or $L_{h2} = \langle \overrightarrow{c_i}, \overrightarrow{u_{i1}} \rangle$ and returns $\overrightarrow{\overrightarrow{c_i}}$ or $\overrightarrow{c_i}$.

H_3 *Oracle* : The adversary queries the challenger with inputs $(\overrightarrow{u_{i1}}, \overrightarrow{v_{i1}}, \overrightarrow{u_{i2}}, \overrightarrow{v_{i2}})$ or $(\overrightarrow{\mathbb{A}_i}, \overrightarrow{u_{i1}}, \overrightarrow{v_{i1}}, \overrightarrow{u_{i2}}, \overrightarrow{v_{i2}})$. If the H_3 *Oracle* was already queried with $\left(\overrightarrow{\mathbb{A}_i}, \overrightarrow{u_{i1}}, \overrightarrow{v_{i1}}, \overrightarrow{u_{i2}}, \overrightarrow{v_{i2}}\right)$ as input, the challenger extracts the vector $\overrightarrow{\overrightarrow{b_i}}$ from the hash list L_{h3} described below and returns the value. On the other hand if the H_3 *Oracle* was already queried with $(\overrightarrow{u_{i1}}, \overrightarrow{v_{i1}}, \overrightarrow{u_{i2}}, \overrightarrow{v_{i2}})$ as input, the challenger extracts the vector $\overrightarrow{b_i}$ from the hash list L_{h3}. Otherwise, the challenger chooses a random vector $\overrightarrow{\overrightarrow{b_i}} \in_R (\mathbb{Z}_p)^{*m_i}$ or $\overrightarrow{b_i} \in_R (\mathbb{Z}_p)^{*m_i}$ respectively. It makes an entry in the hash list $L_{h3} = \left\langle \overrightarrow{\overrightarrow{b_i}}, \overrightarrow{\mathbb{A}_i}, \overrightarrow{u_{i1}}, \overrightarrow{v_{i1}}, \overrightarrow{u_{i2}}, \overrightarrow{v_{i2}} \right\rangle$ or $L_{h3} = \left\langle \overrightarrow{b_i}, \overrightarrow{u_{i1}}, \overrightarrow{v_{i1}}, \overrightarrow{u_{i2}}, \overrightarrow{v_{i2}} \right\rangle$ and returns $\overrightarrow{\overrightarrow{b_i}}$ or $\overrightarrow{b_i}$ respectively.

H_4 *Oracle* : The adversary queries the challenger with inputs $(\overrightarrow{u_{i1}}, \overrightarrow{v_{i1}}, \overrightarrow{u_{i2}}, \overrightarrow{v_{i2}})$ or $(\overrightarrow{\mathbb{A}_i}, \overrightarrow{u_{i1}}, \overrightarrow{v_{i1}}, \overrightarrow{u_{i2}}, \overrightarrow{v_{i2}})$. If the H_4 *Oracle* was already queried with $\left(\overrightarrow{\mathbb{A}_i}, \overrightarrow{u_{i1}}, \overrightarrow{v_{i1}}, \overrightarrow{u_{i2}}, \overrightarrow{v_{i2}}\right)$ as input, the challenger extracts the vector $\overrightarrow{\overrightarrow{e_i}}$ from the hash list L_{h4} described below and returns the value. On the other hand if the H_4 *Oracle* was already queried with $(\overrightarrow{u_{i1}}, \overrightarrow{v_{i1}}, \overrightarrow{u_{i2}}, \overrightarrow{v_{i2}})$ as input, the challenger extracts the vector $\overrightarrow{b_i}$ from the hash list L_{h4}.

Otherwise, the challenger chooses a random vector $\overrightarrow{e_i} \in_R (\mathbb{Z}_p)^{*^{m_i}}$ or $\overrightarrow{b_i} \in_R (\mathbb{Z}_p)^{*^{m_i}}$ respectively. It makes an entry in the hash list $L_{h4} = \left\langle \overrightarrow{e_i}, \overrightarrow{\mathbb{A}_i}, \overrightarrow{u_{i1}}, \overrightarrow{v_{i1}}, \overrightarrow{u_{i2}}, \overrightarrow{v_{i2}} \right\rangle$ or $L_{h4} = \langle \overrightarrow{e_i}, \overrightarrow{u_{i1}}, \overrightarrow{v_{i1}}, \overrightarrow{u_{i2}}, \overrightarrow{v_{i2}} \rangle$ and returns $\overrightarrow{e_i}$ or $\overrightarrow{e_i}$ respectively.

H_5 *Oracle* : The adversary queries the challenger with inputs $\left(g^{\overrightarrow{t_i}}, g^{\overrightarrow{w_i}}, M_i, \rho_i\right)$. If the H_5 *Oracle* was already queried with $\left(g^{\overrightarrow{t_i}}, g^{\overrightarrow{w_i}}, M_i, \rho_i\right)$ as input, the challenger extracts the vector $\overrightarrow{f_i}$ from the hash list L_{h5} described below and returns the value. Otherwise, the challenger chooses a random vector $\overrightarrow{f_i} \in_R \mathbb{Z}_p{}^{*^{m_i}}$. It makes an entry in the hash list $L_{h5} = \left\langle \overrightarrow{f_i}, g^{\overrightarrow{t_i}}, g^{\overrightarrow{w_i}}, M_i, \rho_i \right\rangle$ and returns $\overrightarrow{f_i}$.

H_6 *Oracle* : The adversary queries the challenger with inputs $\left(\overrightarrow{Z_1}, \overrightarrow{Z_2}, \overrightarrow{Z_3}\right)$. If the H_6 *Oracle* was already queried with $\left(\overrightarrow{Z_1}, \overrightarrow{Z_2}, \overrightarrow{Z_3}\right)$ as input, the challenger extracts the vector $\overrightarrow{l_i}$ from the hash list L_{h6} described below and returns the value. Otherwise, the challenger chooses a random vector $\overrightarrow{l_i} \in_R (\mathbb{Z}_p)^{*^{m_i}}$. It makes an entry in the hash list $L_{h6} = \left\langle \overrightarrow{l_i}, \overrightarrow{Z_1}, \overrightarrow{Z_2}, \overrightarrow{Z_3} \right\rangle$ and returns $\overrightarrow{l_i}$.

Party corruption: The adversary presents the challenger with an attribute vector $\overrightarrow{\mathbb{S}_i}$ and the challenger should return the private key of that user i. The challenger proceeds in the following way:

The challenger checks if the H_1 *Oracle* was already queried for $\overrightarrow{\mathbb{S}_i}$. If yes and $\bigvee_\gamma \tau_i^{(\gamma)} = 1$, it *aborts*. Otherwise it extracts $\overrightarrow{k_i}$, $\overrightarrow{h_i}$ from the list L_{h1} and proceeds to the next step. If $\overrightarrow{\mathbb{S}_i}$ was not queried before, the challenger runs the H_1 *Oracle* with $\overrightarrow{\mathbb{S}_i}$ as input. If $\bigvee_\gamma \tau_i^{(\gamma)} = 1$, it *aborts*. Else the challenger chooses $\overrightarrow{k_i} \in_R \mathbb{Z}_p^{*^{m_i}}$, computes $\overrightarrow{h_i} = g^{\overrightarrow{k_i}}$, adds the tuple $\left\langle \overrightarrow{h_i}, \overrightarrow{\mathbb{S}_i}, \overrightarrow{\tau_i}, \overrightarrow{k_i} \right\rangle$ to the L_{h1} list.

The challenger does not know the master secret key s_1 as master public key $y_1 = g^a$ setting $s_1 = a$. Therefore in order to generate the private key of users, the challenger makes use of the random oracles and generates the private key as described below:

- The challenger chooses $\overrightarrow{c_i}, \overrightarrow{b_i}, \overrightarrow{e_i}, \overrightarrow{x_i}', \overrightarrow{r_i}' \in_R Z_p{}^{*^{m_i}}$.
- It sets $\overrightarrow{u_{i1}} = g^{\overrightarrow{x'_i}} . y_1{}^{-\overrightarrow{c_i}}$.
- It sets $H_2\left(\overrightarrow{u_{i1}}\right) = \overrightarrow{c_i}$ and adds the tuple $\langle \overrightarrow{c_i}, \overrightarrow{u_{i1}} \rangle$ the L_{h2} list.
- It sets $\overrightarrow{d_{i1}} = \overrightarrow{x'_i}$, $\overrightarrow{d_{i2}} = \overrightarrow{x'_i} + \overrightarrow{r'_i}\overrightarrow{b_i} + s_2\overrightarrow{e_i}$ and $\overrightarrow{u_{i2}} = g^{\overrightarrow{r'_i}} \cdot y_1{}^{\overrightarrow{c_i} \cdot \overrightarrow{b_i}^{-1}}$.
- It computes $\overrightarrow{v_{i1}} = g^{\overrightarrow{k_i} . x'_i} . y_1{}^{-\overrightarrow{k_i} . \overrightarrow{c_i}}$ and $\overrightarrow{v_{i2}} = g^{\overrightarrow{k_i} . r'_i} . y_1{}^{\overrightarrow{k_i} . \overrightarrow{c_i} . \overrightarrow{b_i}^{-1}}$.
- It also sets the hash function values $H_3\left(\overrightarrow{u_{i1}}, \overrightarrow{v_{i1}}, \overrightarrow{u_{i2}}, \overrightarrow{v_{i2}}\right) = \overrightarrow{b_i}$, $H_4\left(\overrightarrow{u_{i1}}, \overrightarrow{v_{i1}}, \overrightarrow{u_{i2}}, \overrightarrow{v_{i2}}\right) = \overrightarrow{e_i}$ and adds the tuples $\left\langle \overrightarrow{b_i}, \overrightarrow{u_{i1}}, \overrightarrow{v_{i1}}, \overrightarrow{u_{i2}}, \overrightarrow{v_{i2}} \right\rangle$, $\langle \overrightarrow{e_i}, \overrightarrow{u_{i1}}, \overrightarrow{v_{i1}}, \overrightarrow{u_{i2}}, \overrightarrow{v_{i2}} \rangle$ to the lists L_{h3} and L_{h4} respectively.
- It computes $\overrightarrow{h_i}^{s_2}$.
- It returns the tuple $\left\langle \overrightarrow{u_{i1}}, \overrightarrow{v_{i1}}, \overrightarrow{u_{i2}}, \overrightarrow{v_{i2}}, \overrightarrow{d_{i1}}, \overrightarrow{d_{i2}}, \overrightarrow{h_i}^{s_2} \right\rangle$ as the private key of the user with attribute vector $\overrightarrow{\mathbb{S}_i}$ and makes an entry in the list $L_E = \left\langle \overrightarrow{u_{i1}}, \overrightarrow{v_{i1}}, \overrightarrow{u_{i2}}, \overrightarrow{v_{i2}}, \overrightarrow{d_{i1}}, \overrightarrow{d_{i2}}, \overrightarrow{\mathbb{S}_i} \right\rangle$.

Lemma 2: The private key returned by the challenger during the PartyCorruption query are consistent with the system.

Proof: We now prove that the components returned by the challenger are consistent with that of the system. The components returned by the challenger should satisfy the 3 checks given in Secret Key Sanity Check.

- ***Test 1 :*** Check if $\dfrac{g^{\overrightarrow{d_{i1}}}}{y_1^{H_2\left(\overrightarrow{u_{i1}}\right)}} \stackrel{?}{=} \overrightarrow{u_{i1}}$.

  This can be verified as $\dfrac{g^{\overrightarrow{x'_i}}}{g^{a.H_2\left(\overrightarrow{u_{i1}}\right)}}$ where $\overrightarrow{c_i} = H_2\left(\overrightarrow{u_{i1}}\right)$. This is equal to $g^{\overrightarrow{x'_i} - a.\overrightarrow{c_i}} = g^{\overrightarrow{x'_i}} . y_1{}^{-\overrightarrow{c_i}} = \overrightarrow{u_{i1}}$.

– ***Test 2 :*** Check if $\dfrac{g^{\overrightarrow{d_{i2}}}}{\overrightarrow{u_{i2}}^{H_3\left(\overrightarrow{u_{i1}},\overrightarrow{v_{i1}},\overrightarrow{u_{i2}},\overrightarrow{v_{i2}}\right)}.y_2{}^{H_4\left(\overrightarrow{u_{i1}},\overrightarrow{v_{i1}},\overrightarrow{u_{i2}},\overrightarrow{v_{i2}}\right)}} \stackrel{?}{=} \overrightarrow{u_{i1}}$.

  This follows as $\dfrac{g^{\overrightarrow{x'_i}+r'_i\overrightarrow{b_i}+s_2\overrightarrow{e_i}}}{\left(g^{\overrightarrow{r'_i}}.y_1{}^{\overrightarrow{c_i}.\overrightarrow{b_i}^{-1}}\right)^{\overrightarrow{b_i}}.g^{s_2.\overrightarrow{e_i}}} = g^{\overrightarrow{x'_i}-a.\overrightarrow{c_i}} = g^{\overrightarrow{x'_i}}.y_1{}^{-\overrightarrow{c_i}} = \overrightarrow{u_{i1}}$, as $\overrightarrow{b_i} = H_3\left(\overrightarrow{u_{i1}},\overrightarrow{v_{i1}},\overrightarrow{u_{i2}},\overrightarrow{v_{i2}}\right)$ and $\overrightarrow{e_i} = H_4\left(\overrightarrow{u_{i2}},\overrightarrow{v_{i2}}\right)$.

– ***Test 3 :*** Check if $\dfrac{\overrightarrow{h_i}^{\overrightarrow{d_{i2}}}}{\overrightarrow{v_{i2}}^{H_3\left(\overrightarrow{u_{i1}},\overrightarrow{v_{i1}},\overrightarrow{u_{i2}},\overrightarrow{v_{i2}}\right)}.\left(\overrightarrow{h_i}^{s_2}\right)^{H_4\left(\overrightarrow{u_{i1}},\overrightarrow{v_{i1}},\overrightarrow{u_{i2}},\overrightarrow{v_{i2}}\right)}} \stackrel{?}{=} \overrightarrow{v_{i1}}$.

  This follows as $\dfrac{\overrightarrow{h_i}^{\overrightarrow{x'_i}+\overrightarrow{r'_i}.\overrightarrow{b_i}+s_2.\overrightarrow{e_i}}}{\left(g^{\overrightarrow{k_i}.\overrightarrow{r'_i}}.y_1{}^{\overrightarrow{k_i}.\overrightarrow{c_i}.\overrightarrow{b_i}^{-1}}\right)^{\overrightarrow{b_i}}.\left(\overrightarrow{h_i}^{s_2}\right)^{\overrightarrow{e_i}}} = \overrightarrow{h_i}^{\overrightarrow{x'_i}}.y_1{}^{-\overrightarrow{k_i}.\overrightarrow{c_i}} = \overrightarrow{v_{i1}}$ where $\overrightarrow{b_i} = H_3\left(\overrightarrow{u_{i1}},\overrightarrow{v_{i1}},\overrightarrow{u_{i2}},\overrightarrow{v_{i2}}\right)$ and $\overrightarrow{e_i} = H_4\left(\overrightarrow{u_{i1}},\overrightarrow{v_{i1}},\overrightarrow{u_{i2}},\overrightarrow{v_{i2}}\right)$.

Thus the components generated by the challenger are consistent with the system as the tests 1, 2 and 3 are satisfied. □

Session Simulation: The adversary requires the challenger to simulate shared secret keys. The challenger simulates sessions other than the test session. Here we mention the party which initiates the session as the *owner* of the session and the other party who responds to the request of the owner as the *peer*. We have to consider the following cases during the session simulation phase.

Case 1: In this case, the adversary has executed the PartyCorruption query with respect to i. Hence the adversary knows the secret key of i. The adversary treats i as owner and generates the tuple of values given by $\left\langle \overrightarrow{u_{i1}},\overrightarrow{v_{i1}},\overrightarrow{d_{i2}},\overrightarrow{b_i},\overrightarrow{e_i},\overrightarrow{h_i}^{s_2},g^{\overrightarrow{t_i}},\overrightarrow{w_i}+\overrightarrow{d_{i1}}\cdot H_5\left(g^{\overrightarrow{t_i}},g^{\overrightarrow{w_i}},M_i,\rho_i\right),g^{\overrightarrow{w_i}},X_i^{(1)},\overrightarrow{\overrightarrow{U_i}},M_i,\rho_i \right\rangle$ and passes it to the challenger and asks the challenger to complete the session with j as the peer.

Case 1a: If $\bigvee_\gamma \tau_j^{(\gamma)} = 0$, the challenger knows the secret key corresponding to all γ and hence executes the actual protocol and delivers the session key to the adversary.

Case 1b: If $\bigvee_\gamma \tau_j^{(\gamma)} = 1$, the challenger does not know the secret key corresponding to some γ and hence simulates the session key as follows:

1. The challenger first performs the checks presented in the Step 2 of the Key Agreement protocol, on $\left\langle \overrightarrow{u_{i1}},\overrightarrow{v_{i1}},\overrightarrow{d_{i2}},\overrightarrow{b_i},\overrightarrow{e_i},\overrightarrow{h_i}^{s_2},g^{\overrightarrow{t_i}},\overrightarrow{w_i}+\overrightarrow{d_{i1}}\cdot H_5\left(g^{\overrightarrow{t_i}},g^{\overrightarrow{w_i}},M_i,\rho_i\right),g^{\overrightarrow{w_i}},X_i^{(1)},\overrightarrow{\overrightarrow{U_i}},M_i,\rho_i \right\rangle$.
2. The challenger generates the parameters for the party j in the form of a similar tuple of values given by $\left\langle \overrightarrow{u_{j1}}=g^{\overrightarrow{x_j}},\overrightarrow{v_{j1}}=\overrightarrow{h_j}^{\overrightarrow{x_j}},\overrightarrow{d_{j2}}=\overrightarrow{x_j}+\overrightarrow{r_j}.\overrightarrow{b_j}+s_2\cdot\overrightarrow{e_j},\overrightarrow{b_j},\overrightarrow{e_j},\overrightarrow{h_j}^{s_2},g^{\overrightarrow{t_j}},\overrightarrow{w'_j}+\overrightarrow{x_j}\cdot\overrightarrow{f_j},g^{\overrightarrow{w'_j}}\cdot y_1{}^{-\overrightarrow{c_j}\cdot\overrightarrow{f_j}},X_j^{(1)},\overrightarrow{\overrightarrow{U_j}},M_j,\rho_j \right\rangle$, where it computes $\overrightarrow{r_j},\overrightarrow{x_j} \in_R \mathbb{Z}_p{}^{*m_j}$, $\overrightarrow{t_j},\overrightarrow{w'_j},\overrightarrow{f_j},\overrightarrow{\sigma_j} \in_R Z_p{}^{*l_j}$, $\overrightarrow{h_j} = H_1\left(\overrightarrow{\mathbb{S}_j}\right)$, $\overrightarrow{b_j} = H_3\left(\overrightarrow{u_{j1}},\overrightarrow{v_{j1}},g^{\overrightarrow{r_j}},\overrightarrow{h_j}^{\overrightarrow{r_j}}\right)$ and $\overrightarrow{e_j} = H_4\left(\overrightarrow{u_{j1}},\overrightarrow{v_{j1}},g^{\overrightarrow{r_j}},\overrightarrow{h_j}^{\overrightarrow{r_j}}\right)$.
3. If H_5 was already queried with inputs $\left(g^{\overrightarrow{t_j}},g^{\overrightarrow{w'_j}}\cdot y_1{}^{-\overrightarrow{c_j}\cdot\overrightarrow{f_j}},M_j,\rho_j\right)$, generate a fresh $\overrightarrow{w'_j}$ and recompute the last but two components. With very high probability, the new $\left(g^{\overrightarrow{t_j}},g^{\overrightarrow{w'_j}}\cdot y_1{}^{-\overrightarrow{c_j}\cdot\overrightarrow{f_j}},M_j,\rho_j\right)$ will not result in a previously queried input set to H_5. Set $H_5\left(g^{\overrightarrow{t_j}},g^{\overrightarrow{w'_j}}\cdot y_1{}^{-\overrightarrow{c_j}\cdot\overrightarrow{f_j}},M_j,\rho_j\right)$ as $\overrightarrow{f_j}$.
4. The parameters generated by the challenger will satisfy ***Check 1*** in Step 2 of Key Agreement. This is because the parameters $\left\langle \overrightarrow{u_{j1}},\overrightarrow{v_{j1}},\overrightarrow{d_{j2}},\overrightarrow{b_j},\overrightarrow{e_j},\overrightarrow{h_j}^{s_2}\right\rangle$ are generated in the same way as the original scheme.

5. The parameters generated by the challenger will satisfy ***Check 2*** in the Step 2 of Key Agreement of Section 5, on account of the following.

$$\frac{g^{\overrightarrow{w_j'}+\overrightarrow{x_j}\cdot\overrightarrow{f_j}}}{\left(g^{\overrightarrow{x_j}}\right)^{H_5\left(g^{\overrightarrow{t_j}},g^{\overrightarrow{w_j'}}\cdot y_1{}^{-\overrightarrow{c_j}\cdot\overrightarrow{f_j}},M_j,\rho_j\right)}.(y_1)^{\overrightarrow{c_j}\cdot H_5\left(g^{\overrightarrow{t_j}},g^{\overrightarrow{w_j'}}\cdot y_1{}^{-\overrightarrow{c_j}\cdot\overrightarrow{f_j}},M_j,\rho_j\right)}} = g^{\overrightarrow{w_j'}}\cdot y_1{}^{-\overrightarrow{c_j}\cdot\overrightarrow{f_j}} = g^{\overrightarrow{w_j}}$$

6. The parameters generated by the challenger will satisfy ***Check 3*** in Step 2 of Key Agreement. This is because the parameters $\left\langle X_j^{(1)}, \overrightarrow{\overrightarrow{U_j}}\right\rangle$ are generated in the same way as the original scheme.
7. Thus the parameters generated by the challenger are consistent with that of the system.
8. The challenger sends the parameters to the adversary.
9. The challenger computes $\overrightarrow{\overrightarrow{Z_1}} = \left(g^{\overrightarrow{x_i}}\cdot y_1{}^{\overrightarrow{c_i}}\cdot g^{\overrightarrow{t_i}}\right)^{\overrightarrow{x_j}+\overrightarrow{t_j}}$ where $\overrightarrow{c_i} = H_2\left(\overrightarrow{u_{i1}}\right)$. It also computes $\overrightarrow{P_1} = \left(\overrightarrow{u_{i1}}\cdot y_1{}^{\overrightarrow{c_i}}\cdot g^{\overrightarrow{t_i}}\right)^{\overrightarrow{c_j}}$ and $P_2 = y_1$ where $\overrightarrow{c_j} = H_2\left(\overrightarrow{u_{j1}}\right)$.
10. The challenger computes $\overrightarrow{Z_2} = \overrightarrow{v_{i1}}\cdot\overrightarrow{v_{j1}}$ and $\overrightarrow{Z_3} = \left(g^{\overrightarrow{t_i}}\right)^{\overrightarrow{t_j}}$.
11. The challenger is given access to the $DH\left(y_1,\cdot,\cdot\right)$ oracle, since we assume the hardness of Strong-Diffie Hellman problem. The challenger makes use of the $DH\left(y_1,\cdot,\cdot\right)$ Oracle to answer the query as follows:
    - The challenger finds a $\overrightarrow{Z}$ such that $DH\left(P_2, \overrightarrow{P_1}, \overrightarrow{Z_1}/\overrightarrow{\overrightarrow{Z_1}}\right)$ (valid since $P_2 = y_1$) and $H_6\left(\overrightarrow{Z_1},\overrightarrow{Z_2},\overrightarrow{Z_3}\right) = \overrightarrow{Z}$, where $\overrightarrow{Z_2} = \overrightarrow{v_{i1}}\cdot\overrightarrow{v_{j1}}$ and $\overrightarrow{Z_3} = \left(g^{\overrightarrow{t_i}}\right)^{\overrightarrow{t_j}}$.
    - If a $\overrightarrow{Z}$ exists, the challenger returns $\overrightarrow{Z}$ as the shared secret key.
    - Otherwise the challenger chooses $\overrightarrow{Z} \in_R \mathbb{Z}_p{}^{*m_j}$ and for any further query of the form $\left(\overrightarrow{Z_1},\overrightarrow{Z_2},\overrightarrow{Z_3}\right)$ to the H_6 *Oracle*, if $DH\left(P_2, \overrightarrow{P_1}, \overrightarrow{Z_1}/\overrightarrow{\overrightarrow{Z_1}}\right)$, $\overrightarrow{Z_2} = \overrightarrow{v_{i1}}\cdot\overrightarrow{v_{j1}}$ and $\overrightarrow{Z_3} = \left(g^{\overrightarrow{t_i}}\right)^{\overrightarrow{t_j}}$, the challenger returns $\overrightarrow{Z}$ as the result to the query.

Finally the challenger returns $\overrightarrow{Z}$ as the shared secret key.

Case 2: The adversary does not know the secret key of i, the owner of the session. Here the adversary simply asks the challenger to generate a session with i as owner and j as peer.

Case 2a: The case where $\bigvee_\gamma \tau_i^{(\gamma)} = 0$ and $\bigvee_\gamma \tau_j^{(\gamma)} = 0$. In this case, the challenger can simulate the computations of both the parties since the challenger knows the private key of the owner i and the peer j.

Case 2b: The case where either $\bigvee_\gamma \tau_i^{(\gamma)} = 0$ or $\bigvee_\gamma \tau_j^{(\gamma)} = 0$. Without loss of generality let us consider that $\bigvee_\gamma \tau_i^{(\gamma)} = 0$ and $\bigvee_\gamma \tau_j^{(\gamma)} = 1$. Here the challenger knows the secret key of i but does not know the secret key of j. Hence for i the challenger will generate the session secret key as per the algorithm. For j the challenger has to simulate as follows:

1. The challenger generates the parameters for the party j in the form of a similar tuple of values given by $\left\langle \overrightarrow{u_{j1}} = g^{\overrightarrow{x_j}}, \overrightarrow{v_{j1}} = \overrightarrow{h_j}^{\overrightarrow{x_j}}, \overrightarrow{d_{j2}} = \overrightarrow{x_j} + \overrightarrow{r_j}.\overrightarrow{b_j} + s_2\cdot\overrightarrow{e_j}, \overrightarrow{b_j}, \overrightarrow{e_j}, \overrightarrow{h_j}^{s_2}, g^{\overrightarrow{t_j}}, \overrightarrow{w_j'} + \overrightarrow{x_j}\cdot\overrightarrow{f_j}, g^{\overrightarrow{w_j'}}\cdot y_1{}^{-\overrightarrow{c_j}\cdot\overrightarrow{f_j}}, X_j^{(1)}, \overrightarrow{\overrightarrow{U_j}}, M_j, \rho_j \right\rangle$, where it computes $\overrightarrow{r_j}, \overrightarrow{x_j} \in_R Z_p{}^{*m_j}$, $\overrightarrow{t_j}, \overrightarrow{w_j'}, \overrightarrow{f_j}, \overrightarrow{\sigma_j} \in_R Z_p{}^{*l_j}$, $\overrightarrow{h_j} = H_1\left(\overrightarrow{\mathbb{S}_j}\right)$, $\overrightarrow{b_j} = H_3\left(\overrightarrow{u_{j1}}, \overrightarrow{v_{j1}}, g^{\overrightarrow{r_j}}, \overrightarrow{h_j}^{\overrightarrow{r_j}}\right)$ and $\overrightarrow{e_j} = H_4\left(\overrightarrow{u_{j1}}, \overrightarrow{v_{j1}}, g^{\overrightarrow{r_j}}, \overrightarrow{h_j}^{\overrightarrow{r_j}}\right)$.

2. The challenger also generates the parameters for the party i in the form of a similar tuple of values given by
$\left\langle \overrightarrow{u_{i1}} = g^{\overrightarrow{x_i}}, \overrightarrow{v_{i1}} = \overrightarrow{h_i}^{\overrightarrow{x_i}}, \overrightarrow{d_{i2}} = \overrightarrow{x_i} + \overrightarrow{r_i}.\overrightarrow{b_i} + s_2 \cdot \overrightarrow{e_i}, \overrightarrow{b_i}, \overrightarrow{e_i}, \overrightarrow{h_i}^{s_2}, g^{\overrightarrow{t_i}}, \overrightarrow{w_i'} + \overrightarrow{x_i} \cdot \overrightarrow{f_i}, g^{\overrightarrow{w_i'}} \cdot y_1{}^{-\overrightarrow{c_i} \cdot \overrightarrow{f_i}}, X_i^{(1)}, \overrightarrow{\overrightarrow{U_i}}, M_i, \rho_i \right\rangle$
with i's private key for user i.

3. If H_5 was already queried with inputs $\left(g^{\overrightarrow{t_j}}, g^{\overrightarrow{w_j'}} \cdot y_1{}^{-\overrightarrow{c_j} \cdot \overrightarrow{f_j}}, M_j, \rho_j\right)$, generate a fresh $\overrightarrow{w_j'}$ and recompute the last but two components. With very high probability, the new $\left(g^{\overrightarrow{t_j}}, g^{\overrightarrow{w_j'}} \cdot y_1{}^{-\overrightarrow{c_j} \cdot \overrightarrow{f_j}}, M_j, \rho_j\right)$ will not result in a previously queried input set to H_5. Set $H_5\left(g^{\overrightarrow{t_j}}, g^{\overrightarrow{w_j'}} \cdot y_1{}^{-\overrightarrow{c_j} \cdot \overrightarrow{f_j}}, M_j, \rho_j\right)$ as $\overrightarrow{f_j}$.

4. Similarly if H_5 was already queried with inputs $\left(g^{\overrightarrow{t_i}}, g^{\overrightarrow{w_i'}} \cdot y_1{}^{-\overrightarrow{c_i} \cdot \overrightarrow{f_i}}, M_i, \rho_i\right)$, generate a fresh $\overrightarrow{w_i'}$ and recompute the last but two components. With very high probability, the new $\left(g^{\overrightarrow{t_i}}, g^{\overrightarrow{w_i'}} \cdot y_1{}^{-\overrightarrow{c_i} \cdot \overrightarrow{f_i}}, M_i, \rho_i\right)$ will not result in a previously queried input set to H_5. Set $H_5\left(g^{\overrightarrow{t_i}}, g^{\overrightarrow{w_i'}} \cdot y_1{}^{-\overrightarrow{c_i} \cdot \overrightarrow{f_i}}, M_i, \rho_i\right)$ as $\overrightarrow{f_i}$.

5. The challenger computes $\overrightarrow{\overrightarrow{Z_1}} = \left(g^{\overrightarrow{x_i}} \cdot y_1{}^{\overrightarrow{c_i}} \cdot g^{\overrightarrow{t_i}}\right)^{\overrightarrow{x_j} + \overrightarrow{t_j}}$ where $\overrightarrow{c_i} = H_2\left(\overrightarrow{u_{i1}}\right)$. It also computes $\overrightarrow{P_1} = \left(\overrightarrow{u_{i1}} \cdot y_1{}^{\overrightarrow{c_i}} \cdot g^{\overrightarrow{t_i}}\right)^{\overrightarrow{c_j}}$ and $P_2 = y_1$ where $\overrightarrow{c_j} = H_2\left(\overrightarrow{u_{j1}}\right)$.

6. The challenger computes $\overrightarrow{Z_2} = \overrightarrow{v_{i1}} \cdot \overrightarrow{v_{j1}}$ and $\overrightarrow{Z_3} = \left(g^{\overrightarrow{t_i}}\right)^{\overrightarrow{t_j}}$.

7. The challenger is given access to the $DH\left(y_1, \cdot, \cdot\right)$ oracle, since we assume the hardness of Strong-Diffie Hellman problem. The challenger makes use of the $DH\left(y_1, \cdot, \cdot\right)$ Oracle to answer the query as follows:
   - The challenger finds a $\overrightarrow{Z}$ such that $DH\left(P_2, \overrightarrow{P_1}, \overrightarrow{Z_1}/\overrightarrow{\overrightarrow{Z_1}}\right)$ (valid since $P_2 = y_1$) and $H_6\left(\overrightarrow{Z_1}, \overrightarrow{Z_2}, \overrightarrow{Z_3}\right) = \overrightarrow{Z}$, where $\overrightarrow{Z_2} = \overrightarrow{v_{i1}} \cdot \overrightarrow{v_{j1}}$ and $\overrightarrow{Z_3} = \left(g^{\overrightarrow{t_i}}\right)^{\overrightarrow{t_j}}$.
   - If a $\overrightarrow{Z}$ exists, the challenger returns $\overrightarrow{Z}$ as the shared secret key.
   - Otherwise the challenger chooses $\overrightarrow{Z} \in_R {Z_p}^{*^{m_j}}$ and for any further query of the form $\left(\overrightarrow{Z_1}, \overrightarrow{Z_2}, \overrightarrow{Z_3}\right)$ to the H_6 *Oracle*, if $DH\left(P_2, \overrightarrow{P_1}, \overrightarrow{Z_1}/\overrightarrow{\overrightarrow{Z_1}}\right)$, $\overrightarrow{Z_2} = \overrightarrow{v_{i1}} \cdot \overrightarrow{v_{j1}}$ and $\overrightarrow{Z_3} = \left(g^{\overrightarrow{t_i}}\right)^{\overrightarrow{t_j}}$, the challenger returns $\overrightarrow{Z}$ as the result to the query.

Finally the challenger returns $\overrightarrow{Z}$ as the shared secret key.

Case 2c: The case where $\bigvee_\gamma \tau_i^{(\gamma)} = 1$ and $\bigvee_\gamma \tau_j^{(\gamma)} = 1$. In this case the challenger does not know the secret key of both i and j. Hence the challenger has to simulate the session values for both i and j, which is done as follows:

1. The challenger generates the parameters for the party j in the form of a similar tuple of values given by
$\left\langle \overrightarrow{u_{j1}} = g^{\overrightarrow{x_j}}, \overrightarrow{v_{j1}} = \overrightarrow{h_j}^{\overrightarrow{x_j}}, \overrightarrow{d_{j2}} = \overrightarrow{x_j} + \overrightarrow{r_j}.\overrightarrow{b_j} + s_2 \cdot \overrightarrow{e_j}, \overrightarrow{b_j}, \overrightarrow{e_j}, \overrightarrow{h_j}^{s_2}, g^{\overrightarrow{t_j}}, \overrightarrow{w_j} + \overrightarrow{x_j} \cdot \overrightarrow{f_j}, g^{\overrightarrow{w_j}} \cdot y_1{}^{-\overrightarrow{c_j} \cdot \overrightarrow{f_j}}, X_j^{(1)}, \overrightarrow{\overrightarrow{U_j}}, M_j, \rho_j \right\rangle$,
where it computes $\overrightarrow{r_j}, \overrightarrow{x_j} \in_R {\mathbb{Z}_p}^{*^{m_j}}$, $\overrightarrow{t_j}, \overrightarrow{w_j}, \overrightarrow{f_j}, \overrightarrow{\sigma_j} \in_R {Z_p}^{*l_j}$, $\overrightarrow{h_j} = H_1\left(\overrightarrow{\mathbb{S}_j}\right)$, $\overrightarrow{b_j} = H_3\left(\overrightarrow{u_{j1}}, \overrightarrow{v_{j1}}, g^{\overrightarrow{r_j}}, \overrightarrow{h_j}^{\overrightarrow{r_j}}\right)$ and $\overrightarrow{e_j} = H_4\left(\overrightarrow{u_{j1}}, \overrightarrow{v_{j1}}, g^{\overrightarrow{r_j}}, \overrightarrow{h_j}^{\overrightarrow{r_j}}\right)$.

2. The challenger also generates the parameters for the party i in the form of a similar tuple of values given by $\left\langle \overrightarrow{u_{i1}} = g^{\overrightarrow{x_i}}, \overrightarrow{v_{i1}} = \overrightarrow{h_i}^{\overrightarrow{x_i}}, \overrightarrow{d_{i2}} = \overrightarrow{x_i} + \overrightarrow{r_i}.\overrightarrow{b_i} + s_2 \cdot \overrightarrow{e_i}, \overrightarrow{b_i}, \overrightarrow{e_i}, \overrightarrow{h_i}^{s_2}, g^{\overrightarrow{t_i}}, \overrightarrow{w_i'} + \overrightarrow{x_i} \cdot \overrightarrow{f_i}, g^{\overrightarrow{w_i'}} \cdot y_1{}^{-\overrightarrow{c_i} \cdot \overrightarrow{f_i}}, X_i^{(1)}, \overrightarrow{\overrightarrow{U_i}}, M_i, \rho_i \right\rangle$,

where it computes $\vec{r_i}, \vec{x_i} \in_R {\mathbb{Z}_p^*}^{m_i}$, $\vec{t_i}, \vec{w_i'}, \vec{f_i}, \vec{\sigma_i} \in_R {Z_p^*}^{l_i}$, $\vec{h_i} = H_1\left(\vec{\mathbb{S}_i}\right)$, $\vec{b_i} = H_3\left(\overrightarrow{u_{i1}}, \overrightarrow{v_{i1}}, g^{\vec{r_i}}, \vec{h_i}^{\vec{r_i}}\right)$ and $\vec{e_i} = H_4\left(\overrightarrow{u_{i1}}, \overrightarrow{v_{i1}}, g^{\vec{r_i}}, \vec{h_i}^{\vec{r_i}}\right)$.

3. If H_5 was already queried with inputs $\left(g^{\vec{t_j}}, g^{\vec{w_j'}} \cdot {y_1}^{-\vec{c_j} \cdot \vec{f_j}}, M_j, \rho_j\right)$, generate a fresh $\vec{w_j'}$ and recompute the last but two components. With very high probability, the new $\left(g^{\vec{t_j}}, g^{\vec{w_j'}} \cdot {y_1}^{-\vec{c_j} \cdot \vec{f_j}}, M_j, \rho_j\right)$ will not result in a previously queried input set to H_5. Set $H_5\left(g^{\vec{t_j}}, g^{\vec{w_j'}} \cdot {y_1}^{-\vec{c_j} \cdot \vec{f_j}}, M_j, \rho_j\right)$ as $\vec{f_j}$.
4. Similarly if H_5 was already queried with inputs $\left(g^{\vec{t_i}}, g^{\vec{w_i'}} \cdot {y_1}^{-\vec{c_i} \cdot \vec{f_i}}, M_i, \rho_i\right)$, generate a fresh $\vec{w_i'}$ and recompute the last but two components. With very high probability, the new $\left(g^{\vec{t_i}}, g^{\vec{w_i'}} \cdot {y_1}^{-\vec{c_i} \cdot \vec{f_i}}, M_i, \rho_i\right)$ will not result in a previously queried input set to H_5. Set $H_5\left(g^{\vec{t_i}}, g^{\vec{w_i'}} \cdot {y_1}^{-\vec{c_i} \cdot \vec{f_i}}, M_i, \rho_i\right)$ as $\vec{f_i}$.
5. The challenger computes $\overrightarrow{\overrightarrow{Z_1}} = \left(g^{\vec{x_i}} \cdot {y_1}^{\vec{c_i}} \cdot g^{\vec{t_i}}\right)^{\vec{x_j} + \vec{t_j}}$ where $\vec{c_i} = H_2\left(\overrightarrow{u_{i1}}\right)$. It also computes $\vec{P_1} = \left(\overrightarrow{u_{i1}} \cdot {y_1}^{\vec{c_i}} \cdot g^{\vec{t_i}}\right)^{\vec{c_j}}$ and $P_2 = y_1$ where $\vec{c_j} = H_2\left(\overrightarrow{u_{j1}}\right)$.
6. The challenger computes $\vec{Z_2} = \overrightarrow{v_{i1}} \cdot \overrightarrow{v_{j1}}$ and $\vec{Z_3} = \left(g^{\vec{t_i}}\right)^{\vec{t_j}}$.
7. The challenger is given access to the $DH(y_1, \cdot, \cdot)$ oracle, since we assume the hardness of Strong-Diffie Hellman problem. The challenger makes use of the $DH(y_1, \cdot, \cdot)$ Oracle to answer the query as follows:
   - The challenger finds a $\vec{Z}$ such that $DH\left(P_2, \vec{P_1}, \vec{Z_1}/\overrightarrow{\overrightarrow{Z_1}}\right)$ (valid since $P_2 = y_1$) and $H_6\left(\vec{Z_1}, \vec{Z_2}, \vec{Z_3}\right) = \vec{Z}$, where $\vec{Z_2} = \overrightarrow{v_{i1}} \cdot \overrightarrow{v_{j1}}$ and $\vec{Z_3} = \left(g^{\vec{t_i}}\right)^{\vec{t_j}}$.
   - If a $\vec{Z}$ exists, the challenger returns $\vec{Z}$ as the shared secret key.
   - Otherwise the challenger chooses $\vec{Z} \in_R {\mathbb{Z}_p^*}^{m_j}$ and for any further query of the form $\left(\vec{Z_1}, \vec{Z_2}, \vec{Z_3}\right)$ to the H_6 *Oracle*, if $DH\left(P_2, \vec{P_1}, \vec{Z_1}/\overrightarrow{\overrightarrow{Z_1}}\right)$, $\vec{Z_2} = \overrightarrow{v_{i1}} \cdot \overrightarrow{v_{j1}}$ and $\vec{Z_3} = \left(g^{\vec{t_i}}\right)^{\vec{t_j}}$, the challenger returns $\vec{Z}$ as the result to the query.

Finally the challenger returns $\vec{Z}$ as the shared secret key.

Test Session: The adversary impersonates as user i and sends the parameters as the following tuple of values $\left\langle \overrightarrow{u_{i1}}, \overrightarrow{v_{i1}}, \overrightarrow{d_{i2}}, \vec{b_i}, \vec{e_i}, \vec{h_i}^{s_2}, g^{\vec{t_i}}, \vec{w_i} + \overrightarrow{d_{i1}} \cdot H_5\left(g^{\vec{t_i}}, g^{\vec{w_i}}, M_i, \rho_i\right), g^{\vec{w_i}}, X_i^{(1)}, \overrightarrow{\overrightarrow{U_i}}, M_i, \rho_i \right\rangle$ to the challenger for session simulation. The challenger runs the H_1 *Oracle* with input $\vec{\mathbb{S}_i}$. The test session is assumed to run between two users i and j, where adversary impersonates as i and challenger has to generate parameters for user j. If $\bigvee_\gamma \tau_i^{(\gamma)} = 0$, it aborts. Else it does the following:

- The challenger now passes on to the adversary, the parameters as being the following tuple of values $\left\langle \overrightarrow{u_{j1}} = g^{\vec{x_j}}, \overrightarrow{v_{j1}} = \vec{h_j}^{\vec{x_j}}, \overrightarrow{d_{j2}} = \vec{x_j} + \vec{r_j} \cdot \vec{b_j} + s_2 \cdot \vec{e_j}, \vec{b_j}, \vec{e_j}, \vec{h_j}^{s_2}, D \cdot g^{-\overrightarrow{d_{j1}}}, \vec{w_j} + \overrightarrow{d_{j1}} \cdot H_5\left(D \cdot g^{-\overrightarrow{d_{j1}}}, g^{\vec{w_j}}, M_j, \rho_j\right), X_j^{(1)}, \overrightarrow{\overrightarrow{U_j}}, \right.$ where $\overrightarrow{d_{j1}}$ is the private key component associated with User j which is known to the challenger, $\vec{r_j}, \vec{x_j} \in_R {\mathbb{Z}_p^*}^{m_j}$, $\vec{w_j}, \vec{\sigma_j} \in_R {Z_p^*}^{l_j}$, $\vec{h_j} = H_1\left(\vec{\mathbb{S}_j}\right)$, $\vec{b_j} = H_3\left(\overrightarrow{u_{j1}}, \overrightarrow{v_{j1}}, g^{\vec{r_j}}, \vec{h_j}^{\vec{r_j}}\right)$ and $\vec{e_j} = H_4\left(\overrightarrow{u_{j1}}, \overrightarrow{v_{j1}}, g^{\vec{r_j}}, \vec{h_j}^{\vec{r_j}}\right)$. The parameters passed satisfy the checks as they are generated in the way similar to the scheme and $g^{\vec{t_j}} = D \cdot g^{-\overrightarrow{d_{j1}}} = g^{b \cdot \vec{1} - \overrightarrow{d_{j1}}}$

- The challenger performs the checks specified in *Step* 2 of the **Key Agreement** algorithm described in *Section* 5 on $\left\langle \overrightarrow{u_{i1}}, \overrightarrow{v_{i1}}, \overrightarrow{d_{i2}}, \overrightarrow{b_i}, \overrightarrow{e_i}, \overrightarrow{h_i}^{s_2}, g^{\overrightarrow{t_i}}, \overrightarrow{w_i} + \overrightarrow{d_{i1}} \cdot H_5\left(g^{\overrightarrow{t_i}}, g^{\overrightarrow{w_i}}, M_i, \rho_i\right), g^{\overrightarrow{w_i}}, X_i^{(1)}, \overrightarrow{\overrightarrow{U}}_i, M_i, \rho_i \right\rangle$. If the checks pass, the challenger proceeds to next step. Else, it aborts.
- The challenger returns a $\overrightarrow{Z} \in_R {\mathbb{Z}_p^*}^{m_i}$ as the shared secret key. This won't be a valid shared secret key. But in order to find that this is invalid the adversary should have queried the H_6 *Oracle* with a valid tuple $\left(\overrightarrow{Z_1}, \overrightarrow{Z_2}, \overrightarrow{Z_3}\right)$. Thus the challenger computes $\overrightarrow{\overrightarrow{Z}}_2 = \left(\overrightarrow{Z_2}/\overrightarrow{v_{j1}}\right)^{\overrightarrow{k_i}^{-1}}$ and $\overrightarrow{\overrightarrow{Z}}_3 = \overrightarrow{Z_3} \cdot \left(g^{\overrightarrow{t_i}}\right)^{\overrightarrow{d_{j1}}}$. The challenger also computes $\overrightarrow{S} = \left(\overrightarrow{Z_1}/\overrightarrow{\overrightarrow{Z}}_2 \cdot \overrightarrow{\overrightarrow{Z}}_3\right)^{\overrightarrow{c_i}^{-1}}$ where $\overrightarrow{c_i} = H_2\left(\overrightarrow{u_{i1}}\right)$.
- Finally the challenger can return the solution for the CDH hard problem as shown in the lemma below. In particular, since $\bigvee_\gamma \tau_i^{(\gamma)} = 1$, there exists a γ such that $\tau_i^{(\gamma)} = 1$. The challenger returns the value $S^{(\gamma)}$.

Lemma 3: The challenger returns the solution to the CDH instance of the SDH hard problem set in the beginning.

Proof: The challenger computes $\overrightarrow{S} = \left(\overrightarrow{Z_1}/\overrightarrow{\overrightarrow{Z}}_2 \cdot \overrightarrow{\overrightarrow{Z}}_3\right)^{\overrightarrow{c_i}^{-1}}$ where $\overrightarrow{c_i} = H_2\left(\overrightarrow{u_{i1}}\right)$.

- $\overrightarrow{S} = \left(g^{\left(\overrightarrow{d_{i1}}+\overrightarrow{t_i}\right)\left(\overrightarrow{d_{j1}}+b\cdot\overrightarrow{1}-\overrightarrow{d_{j1}}\right)} \overrightarrow{\overrightarrow{Z}}_2 \cdot \overrightarrow{\overrightarrow{Z}}_3\right)^{\overrightarrow{c_i}^{-1}}$.
- Since $\bigvee_\gamma \tau_i^{(\gamma)} = 1$, there exists a γ such that $\tau_i^{(\gamma)} = 1$. Now, $\overline{Z}_2^{(\gamma)} = \left(Z_2^{(\gamma)}/v_{j1}^{(\gamma)}\right)^{\left(k_i^{(\gamma)}\right)^{-1}} = \left(v_{i1}^{(\gamma)} \cdot v_{j1}^{(\gamma)}/v_{j1}^{\gamma}\right)^{\left(k_i^{(\gamma)}\right)^{-1}} = \left({h_i^{(\gamma)}}^{x_i^{(\gamma)}}\right)^{\left(k_i^{(\gamma)}\right)^{-1}} = \left(g^{b\cdot k_i^{(\gamma)}}\right)^{x_i^{(\gamma)}\cdot\left(k_i^{(\gamma)}\right)^{-1}} = g^{b\cdot x_i^{(\gamma)}}$. (**Note:** The component $h_i^{(\gamma)} = \left(g^b\right)^{k_i^{(\gamma)}}$ as $\tau_i^{(\gamma)} = 1$.).
- $\overrightarrow{\overrightarrow{Z}}_3 = \overrightarrow{Z_3} \cdot \left(g^{\overrightarrow{t_i}}\right)^{\overrightarrow{d_{j1}}} = \left(g^{\overrightarrow{t_i}}\right)^{\left(b\cdot\overrightarrow{1}-\overrightarrow{d_{j1}}\right)} \cdot \left(g^{\overrightarrow{t_i}}\right)^{\overrightarrow{d_{j1}}} = g^{b\cdot\overrightarrow{t_i}}$.
- Therefore $S^{(\gamma)} = \left(g^{\left(x_i^{(\gamma)}+a\cdot c_i^{(\gamma)}+t_i^{(\gamma)}\right)\left(d_{j1}^{(\gamma)}+b-d_{j1}^{(\gamma)}\right)}/g^{b\cdot x_i^{(\gamma)}} \cdot g^{b\cdot t_i^{(\gamma)}}\right)^{{c_i^{(\gamma)}}^{-1}} = g^{ab}$.

Thus we have proved that the challenger returns the solution to the CDH Problem. □

8 Probability Analysis

In this section we present the probability analysis of our scheme presented in Section 4.

Theorem 1 : If ϵ is the probability of the adversary in distinguishing between a random shared secret key and a valid shared secret key, the probability of solving the underlying SDH problem, ϵ' is given by

$$\epsilon' = \epsilon.\left(1 - \frac{1}{q_E + 2}\right)^{q_E+1}.\left(\frac{1}{q_E + 2}\right)$$

where q_E = Number of key extract or Party Corruption queries.

Proof: A solution to the hard problem can be generated only if the following events hold good.

- S_1 : The challenger is able to answer all the Party Corruption queries. In other words, the challenger should not abort in the Party Corruption phase.

- S_2 : In the test session, the private key of user that the adversary impersonates should not be computable.
- S_3 : In the test session, the challenger should be able to compute the private key of the user it is simulating.
- S_4 : The challenger should choose the valid tuple $\left(\overrightarrow{Z_1}, \overrightarrow{Z_2}, \overrightarrow{Z_3}\right)$ from the list L_{h6} which has the hard problem injected in it.

Therefore, a solution to SDH problem can be obtained if

$$(Adversary\ succeeds\ in\ the\ game\ in\ Section\ 3) \bigwedge S_1 \bigwedge S_2 \bigwedge S_3 \bigwedge S_4.$$
$$Pr\left(breaking\ SDH\right) = Pr\left(Adversary's\ success\right).Pr\left(S_1\right) \cdot Pr\left(S_2\right) \cdot Pr\left(S_3\right) \cdot Pr\left(S_4\right).$$

Consider the H_1 *Oracle*. Assume $P\left(\tau_i^{(\gamma)} = 0\right) = \alpha$. Let q_E be the total number of key extract or Party Corruption queries. Now q_E can be divided into two mutually disjoint subsets $\bar{A}$ and $\bar{B}$. Let $\bar{A}$ be a set of queries for which $H_1\left(\overrightarrow{\mathbb{S}_i}\right)$ resulted in $\tau_i^{(\gamma)} = 0$ and hence the private keys can be computed as described in Party Corruption phase and it will not abort in the Party corruption phase. Let $\overline{B}$ be the set for which $H_1\left(\overrightarrow{\mathbb{S}_i}\right)$ resulted in $\tau_i^{(\gamma)} = 1$ and hence an abort in the Party Corruption phase. Therefore private keys cannot be computed for identities in $\overline{B}$. There are $\alpha^{m_i}.q_E$ identities in $\overline{A}$ and remaining $\left(1-\alpha^{m_i}\right).q_E$ identities in $\overline{B}$.

- $Pr\left(A1\right) = Pr\left(\overrightarrow{\mathbb{S}_i} \in \overline{A}\right)$ for all the q_E queries. This is equal to $\left(\dfrac{\alpha^{m_i}.q_E}{q_E}\right)^{q_E} = \alpha^{m_i \cdot q_E}$.
- $Pr\left(A2\right) = Pr\left(\overrightarrow{\mathbb{S}_i} \in \overline{B}\right)$, where $\overrightarrow{\mathbb{S}_i}$ is the attribute vector of the user i that the adversary impersonates in the Test Session. Therefore $\tau_i^{(\gamma)} = 1$ in this case and hence $h_i = \left(g^b\right)^{k_i^\gamma}$. This is needed to solve the SDH problem. The probability is equal to $\dfrac{\left(1-\alpha^{m_i}\right).q_E}{q_E} = 1 - \alpha^{m_i}$.
- $Pr\left(A3\right) = Pr\left(\overrightarrow{\mathbb{S}_j} \in \overline{A}\right)$, $\overrightarrow{\mathbb{S}_j}$ is the attribute vector of the user j the challenger emulates in the Test Session. This ensures that the private key of j is computable by the challenger. This is equal to α^{m_i}.
- $Pr\left(A4\right) = Pr\left(a\ valid\ \langle Z_1, Z_2, Z_3\rangle \in\ L_{h_6}\ is\ chosen\ by\ the\ challenger\right) = \dfrac{1}{h_6}$, where h_6 is the number of queries made to the H_6 *Oracle*.

Therefore the probability of solving the SDH problem, $\epsilon' = \epsilon.\alpha^{q_E}.\left(1-\alpha\right).\alpha$.

$$\epsilon' = \epsilon.\frac{1}{h_6}.\alpha^{m_i \cdot q_E + 1}.\left(1-\alpha^{m_i}\right).$$

By maximizing this probability with respect to α, we get $\alpha = \left(\dfrac{q_E+1}{q_E+2}\right)^{\dfrac{1}{m_i}}$.

Therefore $\epsilon' = \epsilon.\frac{1}{h_6}\left(1 - \dfrac{1}{q_E+2}\right)^{q_E+1} . \left(\dfrac{1}{q_E+2}\right)$.