A Constant Time, Single Round Attribute-Based Authenticated Key Exchange in Random Oracle Model

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

1 – Theoretical Computer Science Lab,
Department of Computer Science and Engineering,
Indian Institute of Technology Madras,
Chennai, India
suvradip@cse.iitm.ac.in, prangan55@gmail.com,
y.sreenivasarao@yahoo.co.in, srini131293@gmail.com

Abstract. In this paper, we present a single round two-party *attribute-based authenticated key exchange* (ABAKE) protocol in the framework of ciphertext-policy attribute-based systems. 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. The proposed protocol is pairing free, working in prime order group and having tight reduction to Strong Diffie Hellman (SDH) problem under the attribute-based Canetti Krawzyck (CK) model which is a natural extension of the CK model (which is for the PKI-based authenticated key exchange) for the attribute-based setting. The security proof is given in the random oracle model. Our ABAKE protocol does not depend on any underlying attribute-based encryption or signature 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. Another notable feature of our construction is that it involves only constant number of exponentiations per party unlike the state-of-the-art ABAKE protocols where the number of exponentiations performed by each party depends on the size of the linear secret sharing matrix. We achieve this by doing appropriate precomputation of the secret share generation. Ours is the *first* construction that achieves this property. Our scheme has several other advantages. The major one being the capability to handle active adversaries. Most of the previous ABAKE 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 (ABAKE), CK model, ABCK model, Forward secrecy, Key Compromise Impersonation (KCI) attacks.

1 Introduction

Attribute-based Encryption (ABE), introduced by Sahai and Waters [22], 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 [12], [5], [11], [15], [19]. Attribute-based systems fall under two categories: (i) key-policy attribute-based systems, e.g. [12] in which users' secret keys are associated with access policies over a universe of attributes and the ciphertexts are associated with sets of attributes and (ii) ciphertext-policy attribute-based systems, e.g. [5] in which users' private keys are associated with the attributes and the ciphertexts are associated with access policies. In this work, we consider ciphertext-policy attribute-based systems. In reality, a user's access privileges are often granted based on the functional role he/she assumes in an organization, where a role reduces to no more than a set of attributes. In this regard, Ciphertext-Policy ABE (CP-ABE) enables cryptographic access control with respect to functional roles.

Attribute-Based AKE (ABAKE) is a new variant of the Authenticated key Exchange (AKE) that allows users to authenticate each other using their *attributes* unlike in the PKI settings where the users authenticate each other using their public keys. 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. Another interesting application is sharing of medical history of patients with doctors who are appropriately qualified but both doctors and patients would like to remain anonymous without revealing their specific identities. This may be of particular use for health chat rooms or online medical consultancy services where the patients would like to keep their identity hidden and also the doctors who are providing consultancy services to the patients would like to keep their identity anonymous to avoid legal hassles later on. Hence an authenticated key exchange protocol that critically uses attributes can be employed in these settings.

All the previously proposed ABAKE schemes build upon some well-known ABE schemes and the security guarantees of the underlying ABE scheme directly translated to the security of the ABAKE scheme. While a naive approach for designing a key exchange protocol may use encryption and signature algorithms as building blocks, such a solution will be computationally very expensive. Specifically for attribute-based systems the encryption and signing algorithms are very complex involving a number of variables corresponding to attributes, access structures and pairing operations. Hence a fundamental question in the design of key exchange protocol is :

Is it possible to design a protocol for AKE in attribute-based settings, handcrafted using only basic group operations rather than using encryption or signature schemes as building blocks?
Our paper answers affirmatively to this question.

1.1 Related Works

In the recent literature some ABAKE are proposed. Ateniese *et al.* [2] 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 conditions by allowing only a single threshold gate as opposed to several threshold gates that may be present in a general ABAKE settings. Gorantla *et al.* [10] proposed the first ABAKE scheme based on on the CP-ABE scheme of Bethencourt et al. [4] which provides parties with the fine-grained access control based on the attributes of parties. However it does not provide the flexibility of each user to select their access structures which they want their peers to satisfy. In fact, it is an attribute-based group key exchange scheme where the access policy is defined globally and only those members whose attributes satisfy the access structure will be able to establish a common secret session key among themselves. Besides the security of their scheme is analyzed based on the Bellare-Rogaway(BR) model [4].
Birkett and Stebila [6] 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. The scheme is proven secure in the random oracle model based on the BR model. The BR model does not allow the adversary to reveal the session specific information and ephemeral keys. In an independent work Steinwandt and Corona [25] proposed a two-round attribute-based group key exchange based on the group key exchange scheme of Bohli et al. [7]. This scheme also satisfies forward secure. Yoneyama [28] 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 [14] model. But it does not achieve full security (i.e. adaptive security) as it relies on Waters CP-ABE [27] which is selectively secure.
All the previous works on attribute-based key agreement except [6] do not consider an active adversary. The scheme in [6] uses a predicate-based signature to achieve security against active adversaries. However the security of their protocol is proved in a much weaker security model namely the BR model. Also since this scheme uses predicate signature as its underlying building block it has to be unforgeable against revealing the randomness of the underlying signature scheme. Unfortunately as observed in [6], there is no such predicate signature scheme satisfying this property. 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 [28], 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 sends 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 sends it to A. Thus the final shared secret key of A and B will not be in agreement. So although the adversary may not know the actual session key between the two parties he can launch this type of denial-of-service (DOS) attack. Our protocol avoids this kind of an attack by incorporating appropriate verification mechanisms that would abort the process in case of any change in values to be agreed upon.

The schemes discussed above are all in random oracle (RO) model. The trade off between complexity and security of schemes proven secure under the RO model and *standard* model (StdM) are well known. Schemes proven secure under standard model are often very complex and inefficient due to stringent security demands posed by standard model. For instance, the ABAKE scheme of Yoneyama [30] uses a CCA-secure ciphertext policy attribute-based key encapsulation mechanism (CP-AB-KEM) for key exchange. This is instantiated in their paper by Waters CP-ABE scheme [27] and hence is quite complex. In Waters CP-ABE system [27], the size of the parameters output by the Setup algorithm also depends on the attribute universe, i.e., the number of attributes in the system. The predicate-based signature scheme of [6] can be instantiated using some predicate-based (attribute-based) signature schemes [23], [16], [17], [20]. Instantiations using [23], [16] cannot achieve expressive ABAKE (i.e., these signatures only allow threshold access policies). In [17], three signature schemes are proposed. One instantiation in [17] is expressive, but the security proof is given in Generic Group Model (GGM). Other instantiations in [17] need large communication complexity. The instantiation in [20] provides fully secure and expressive access policies in StdM. However, communication complexity of it is larger than that of the efficient instantiation in [17], depending on the size to represent access policies. On the other hand, the schemes proven secure under RO models are often simpler and efficient. The schemes discussed in previous section and most notably our scheme, is very simple and this is mainly due to less severe demands of RO model.

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 [1] in the RO model. We extend the techniques used in [26] to the attribute-based framework. Doing this is not trivial since in an attribute-based system the keys and the ciphertexts have richer structure than identity-based encryption schemes. Besides, we need to keep the attributes of parties unknown. We are able to achieve a tight reduction to the SDH problem based on the RO model.
- A key exchange is supposed to be fundamentally simpler than an encryption scheme. All the previous known attribute-based key agreement protocols use well known existing ABE 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 underlying ABE scheme. Moreover, our construction is also much more efficient compared to the state-of-the-art ABAKE protocols as it *does not* involve any pairing computations. Another significant aspect of the complexity of our scheme is that it involves only $O(1)$ exponentiations (to be specific only 8) and this is independent of the number of attributes or number of parties in the system. A comparison of our protocol with the existing state-of-the-art attribute-based two-party and group key exchange are shown in Table 1.

Scheme	**Type of ABAKE**	**No of Rounds**	**Exp (each party)**	**No of Pairings (each party)**	**Basic Building Blocks**	**Security Model & Assumptions**
Gorantla et al. [10]	ABGKE	1	$size(M_I) \cdot \lvert P\rvert + 2(size(M)) + 1$	$2size(M_I) + 3$	IND-CCA secure EP-AB-KEM (from Bethencourt et al.'s ABE), Pseudo-random function	BR ,GGM, RO
Yonehama [28]	2-party ABAKE	1	$2(size(M) \times n) + (size(M) \times (n_{max} - n))$	$(size(M_I))^2 \times n_{max}$	Waters ABE	eCK, DBDH, RO
Steinwandt et al. [25]	ABGKE	2	Depends on the underlying Signcryption Scheme	Depends on the underlying Signcryption Scheme	Attribute-based Signcryption Scheme	CK, CDH, RO
Ours	2-party ABAKE	1	8*	—	Basic Group Operations	CK, SDH, RO

Table 1 : Comparison with the existing schemes

* see Section 4.1 for details.

- Our scheme is also resistant to an 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 such as forward secrecy, key compromise impersonation attacks.
- Finally, we prove the security of our ABAKE system in the Attribute-Based CK (ABCK) model [29] which is a natural extension of the CK model [8] for attribute-based 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.

In Table 1, **Type of ABAKE** refers to the settings in which the protocols are applicable, e.g. ABGKE means it is an attribute-based group key exchange protocol as [10] and [25]. Also **Exp** refers to the number of exponentiations each party needs to compute in the considered protocols. Gorantla et al.'s ABGKE was based on the ABE scheme of Bethencourt et al. [5]. From the design principle of [5], they constructed an IND-CCA2-secure Key Encapsulation Mechanism (KEM) for attribute-based settings, which they called, Encapsulation Policy attribute-based KEM (EP-AB-KEM). The number of exponentiations each party needs to compute during encryption is two for each leaf node in the ciphertext's access structure and $|L|$ denotes the number of leaf nodes in the access structure.

The number of leaves in the access structure corresponds to the number of rows in the share generating matrix M, i.e., size of M denoted as $size(M)$. In the decryption algorithm, each user needs to perform two pairing operations for each leaf in the access structure that is matched by a private key attribute, i.e., the size of the submatrix M_I of M as shown in Definition 2 in Section 2.2 and at most one exponentiation for each node along a path from each such node to the root. We denote by $|P|$ the maximum length of this path from the node to the root. The scheme [28] is a two-party ABAKE like ours and it uses the Waters ABE [27] as its underlying building block. However Waters ABE had a flaw, they rectified the flaw and used it for their ABAKE scheme. Here n_{max} denotes the maximum number of columns in the share generating matrix and n denotes the number of the columns of the share generating matrix corresponding to the encryptor which in most cases will be less than n_{max}, i.e., $n \leq n_{max}$. Here $size(M)$ denotes the number of rows of the share generating matrix and $I \subset \{1, 2, \cdots, l\}$ corresponds to those rows of the decryptor whose attributes satisfy the access structure of the encryptor (after applying the injective function selected by the encryptor), i.e. $size(M_I)$. The protocol in [25] is also a ABGKE which uses attribute-based signcryption as its main workhorse. So the number of pairing and exponentiation operations also depends on the underlying attribute-based signcryption schemes which is generally very large and computationally intensive considering the state-of-the-art attribute-based signcryption schemes like [21], [9] etc. Our protocol has considerable advantages over the above mentioned protocols. Our protocol involves no pairing operations and a *constant* number of exponentiations by appropriately doing the computations related to share generation in a preprocessing stage as will be discussed in Section 3.1. So the computation cost at each party for our protocol is independent of the size or depth of the access structure.

2 Preliminaries

Notation. Throughout this work, we denote the security parameter by κ. We denote by $x \in_R X$ the fact that the value x is chosen uniformly at random from the set of values X. The notation $\mathbb{G}^*$ denotes all the invertible elements of the group $\mathbb{G}$. We denote by $\vec{a}$ a vector, which is the tuple of values $(a_1, \ldots, a_n)$, where n is the length

of the vector $\vec{a}$. For a vector $\vec{V}$ chosen by a party P_i we use the notation $V^{(i)}$. The kth component of this vector is denoted by $V_k^{(i)}$. If the length of $V^{(i)}$ is m say, then the entire vector $V^{(i)}$ is given by $(V_1^{(i)}, V_2^{(i)}, \cdots, V_m^{(i)})$. When we write $\{V_k^{(i)}\}_{k=a}^{b}$, we mean the tuple of values $(V_a^{(i)}, \cdots, V_b^{(i)})$. If not mentioned otherwise, we will always assume the counting of the components of a vector starts from the index 1. When we write $\mathcal{A}^{f()}$ we mean that $\mathcal{A}$ is given oracle access to the functionality f.

2.1 Access Structure

Definition 1. *[3] Let $\mathcal{P} = \{P_1, \cdots P_n\}$ be a set of parties. A collection $\mathbb{A} \subseteq 2^{\mathcal{P}}$ is monotone if $\forall B, C$: if $B \in \mathbb{A}$ and $B \subseteq C$, then $C \in \mathbb{A}$. An access structure (respectively, monotone access structure) is a collection (respectively, monotone collection) $\mathbb{A}$ of non-empty subsets of $\mathcal{P}$, that is, $\mathbb{A} \subseteq 2^{\mathcal{P}} \setminus \{\emptyset\}$. The sets in $\mathbb{A}$ are called authorized sets, and the sets not in $\mathbb{A}$ are called unauthorized sets.*

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

2.2 Linear Secret-Sharing Scheme

Our construction will employ Linear Secret-Sharing Schemes (LSSSs) [3].

Definition 2. *A secret-sharing scheme Π over a set of parties $\mathcal{P}$ is called linear (over $\mathbb{Z}_p$) if:*

- *The shares for each party form a vector over $\mathbb{Z}_p$.*
- *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 ith row of A is labeled by a party $\rho(i)$ (ρ is a function from $\{1, \ldots, l\}$ to $\mathcal{P}$). When we consider the column vector $\vec{v} = (s, r_2, \ldots, r_n)$, where $s \in \mathbb{Z}_p$ is the secret to be shared and $r_2, \ldots, r_n \in \mathbb{Z}_p$ are randomly chosen, then $A\vec{v}$ is the vector of l shares of the secret s according to Π. The share $(A\vec{v})_i$ belongs to party $\rho(i)$.*

It is shown in [3] that every LSSS according to the above definition also enjoys the *linear reconstruction* property: suppose Π is an LSSS for access structure $\mathbb{A}$, let S denote an authorized set, and define $I \subseteq \{1, \cdots, l, \}$ as $I = \{i : \rho(i) \in S\}$. Then there exist constants $\{\omega_i \in \mathbb{Z}_p\}_{i \in I}$ such that for any valid shares $\{\lambda_i\}$ of a secret s according to Π, $\sum_{i \in I} \omega_i \lambda_i = s$. These constants $\{\omega_i\}$ can be found in time polynomial in the size of the share-generating matrix A [3]. We note that for the security property of LSSS, no such constants $\{\omega_i\}$ exist for unauthorized sets.

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. So, naturally boolean formulas are used to describe the access policy, and equivalent LSSS are used to encrypt the message and decrypt the ciphertext in a CP-ABE system.

2.3 Monotone Span Program

The labelled matrix (A, ρ) in Definition 2 is also called a Monotone Span Program (MSP) [13]. Karchmer and Widgerson [13] introduced the model of MSP, and proved that if there is a MSP for some boolean function then there exists a LSSS for the corresponding access structure. We give the formal definitions and conclusions as in [13], [18].

Definition 3. *A MSP $\mathcal{M}$ is a quadruple $(\mathbb{F}, M, \vec{\varepsilon}, \rho)$ where $\mathbb{F}$ is a field, M is a matrix (with m rows and $d \leq m$ columns) over $\mathbb{F}$, $\rho : \{1, 2, \ldots, m\} \rightarrow \{1, 2, \ldots, n\}$ is a surjective function and the row vector $\vec{\varepsilon} = (1, 0, 0 \cdots 0) \in \mathbb{F}^d$ is called the target vector. The size of $\mathcal{M}$ is the number m of rows is denoted by size$(\mathcal{M})$.*

As the function ρ labels each row i of the matrix M to a party $P_{\rho(i)}$, each party can be regraded as the owner of one or more rows. For any set of parties $G \subseteq \mathcal{P}$, let us denote the sub-matrix consisting of rows owned by the parties in G by M_G. The span of the matrix M, denoted by span(M), is the subspace generated by the rows of M. A MSP $\mathcal{M}$ is said to compute an access structure $\mathbb{A}$ if $G \in \mathbb{A} \Leftrightarrow \vec{\varepsilon} \in span(\mathcal{M}_G)$.

LSSS induced from MSP [13, 18]. Assume that there is a MSP $\mathcal{M} = (\mathbb{F}, M, \vec{\varepsilon}, \rho)$, of size m, computing the access structure $\mathbb{A}$. Then there is a LSSS Π over $\mathbb{F}$ realizing the access structure in which the total size of the shares is the number of rows in the span program, i.e., the size of the span program (i.e., m) [13].

We mention the linear reconstruction property of the LSSS here. For any secret $s \in \mathbb{Z}_p^*$, let $\vec{v} = (s, r_2, \ldots, r_n) \in (\mathbb{Z}_p^*)^d$ denote a random vector. For any authorized set $S \in \mathbb{A}$, let $I = \{i : P_{\rho(i)} \in S\}$ and let $\vec{M}_i$ denote the ith row of M and the shares $\{\lambda_i = (M\vec{v})_i = \vec{M}_i \cdot \vec{v} : i \in I\}$ are held by S. Since S is an authorized set, it holds that $\sum_{i \in I} \omega_i \vec{M}_i = \vec{\varepsilon}$, where the $\{\omega_i\}_{i \in I}$ are the reconstruction constants. Then S can compute:

$$\sum_{i \in I} \omega_i \lambda_i = \sum_{i \in I} \omega_i \left(\vec{M}_i \cdot \vec{v} \right) = \left(\sum_{i \in I} \omega_i \vec{M}_i \right) \cdot \vec{v} = \vec{\varepsilon} \cdot \vec{v} = s. \tag{1}$$

2.4 Complexity Assumptions

The complexity assumptions required for our construction are Computational Diffie-Hellman Problem (CDH), Decisional Diffie-Hellman Problem (DDH) and Strong Diffie-Hellman Problem (SDH) assumptions which are well known and well studied assumptions. Due to space constraints we present these in the appendix A.

2.5 ABCK Security Model

In this section we describe the ABCK model which is a natural extension of the CK model for attribute-based settings. All the definitions given here are for *single-round* two-party ABAKE scheme. For multiple rounds, we need to extend these definitions appropriately. 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 $\mathbb{S}_P$ given by a party P, and outputs a static secret key $SK_{\mathbb{S}_P}$ corresponding to $\mathbb{S}_P$.

▶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 an ABAKE protocol executes the KeyExchange algorithm which initially takes as input the master public key MPK, an access structure $\mathbb{A}$ and a private key for a set of attributes $\mathbb{S}$. The party A (resp., B) starts the protocol by taking as input the master public key MPK, the set of attributes $\mathbb{S}_A$ (resp., $\mathbb{S}_B$), the access policy or access structure $\mathbb{A}_A$ (resp., $\mathbb{A}_B$), and outputs a message say out (resp., out'). The out sent by A is considered to as in' for B and out' sent by B is considered to as in for A. After the exchange is over, the party A attempts to construct the session key Z_A using the key construction function that takes MPK, the set of attributes $\mathbb{S}_A$, the static secret key $SK_{\mathbb{S}_A}$, the access policy or access structure $\mathbb{A}_B$, out and in as parameters. The party B attempts to construct the session key Z_B, using the key construction function that takes MPK, the set of attributes $\mathbb{S}_B$, the static secret key $SK_{\mathbb{S}_B}$, the access policy or access structure $\mathbb{A}_A$, out' and in' as parameters. The session key Z_A will be equal to Z_B if and only if $\mathbb{S}_A \in \mathbb{A}_B$ and $\mathbb{S}_B \in \mathbb{A}_A$ (i.e., the attributes of one party satisfies the access structure of its peer and vice versa). The session key $Z = Z_A = Z_B$ is defined as the session key established between party A and B.

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 form $(\mathcal{I}, \mathbb{A}_A)$ or $(\mathcal{R}, \mathbb{A}_A, \mathbb{A}_B, out)$, where $\mathcal{I}$ and $\mathcal{R}$ denote role identifiers, and A and B are user identifiers. If A was activated with $(\mathcal{I}, \mathbb{A}_A)$, then A is called the session *initiator*. If B was activated with $(\mathcal{R}, \mathbb{A}_A, \mathbb{A}_B, out)$, then B is called the session *responder*. After getting activated with an incoming message of the form $(\mathcal{I}, \mathbb{A}_B, out')$ from the responder B, the initiator A computes the session key. Similarly after getting activated with an incoming message of the form $(\mathcal{R}, \mathbb{A}_A, out)$ from the initiator A, the responder B computes the session key. 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 identifier sid as $(\mathcal{I}, \mathbb{A}_A, \mathbb{A}_B, out, in)$ where out and in are respectively the components sent to B and received from B. If B is the responder of a session initiated by A, it sets the sid as $(\mathcal{R}, \mathbb{A}_A, \mathbb{A}_B, out', in')$ where out' and in' are respectively the components sent to A and received by B. We note that the sid of the responder is defined immediately when it receives a message from the initiator of the form $(\mathcal{I}, \mathbb{A}_A, out)$, whereas sid of the initiator is defined only when it receives the response from its peer.

Adversary. The adversary $\mathcal{A}$ is also modeled 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. Apart from this the adversary can also get secret keys corresponding to a polynomial number of users of its choice adaptively. The adversary can also register attributes of its choice on behalf of any party. The adversary can also access the session states of a polynomial number of sessions of parties which allows him to obtain all the ephemeral secrets or session states corresponding to those sessions.

To model these, the adversary is given access to the following oracle queries:

1. Send(*message*): The ability of the adversary to control the communication network is modeled by 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 $(\mathcal{I}, \mathbb{S}_A, \mathbb{S}_B, m, \star)$

2. SessionStateReveal(sid): The adversary $\mathcal{A}$ obtains the ephemeral secret keys and the session state associated with the session *sid*, if the session is not yet completed (the session key is not established yet). Session state includes all chosen randomness and intermediate computation results, but not the static secret key.
   We assume that once a session gets successfully completed, the session key is output and all the associated session states are erased. So we allow the adversary to make SessionStateReveal queries on an incomplete session. The former case where the adversary makes SessionStateReveal query on a completed session is captured by the SessionKeyReveal(sid) oracle query.
3. SessionKeyReveal(sid): $\mathcal{A}$ is given the session key of a completed session *sid*, provided that the session holds a session key.
4. PartyCorruption($\mathbb{S}_P$): The adversary learns the static secret key corresponding to the set of attributes $\mathbb{S}_P$.
5. 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 Establish($P, \mathbb{S}_P$) query issued by the adversary, then we call the party P *dishonest*.
   If a party is not *corrupt* or *dishonest,* 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 4 (Matching Sessions). *Let Π be a protocol and $sid = (\zeta, \mathbb{A}_A, \mathbb{A}_B, out, in)$ and $sid' = (\zeta', \mathbb{A}_B, \mathbb{A}_A, in', out')$ 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., $\mathbb{S}_{\mathbb{B}}$ satisfies $\mathbb{A}_A$*
- *The attributes of user A satisfy the access structure of user B, i.e., $\mathbb{S}_{\mathbb{A}}$ satisfies $\mathbb{A}_B$*
- *$out = in'$ and $in = out'$*
- *$\zeta \neq \zeta'$*

Definition 5 (Freshness). *A session with identifier sid is called fresh if none of the following queries by an adversary $\mathcal{A}$ are allowed on that session sid or it's matching session sid' (if it exists)*

- *$\mathcal{A}$ issues a SessionKeyReveal query on sid or sid'*
- *$\mathcal{A}$ issues a SessionStateReveal query on sid or sid'*
- *$\mathcal{A}$ issues a Party Corruption($\mathbb{S}_P$) query on the party P owning the session sid or a Party Corruption($\mathbb{S}_{P'}$) query on P' which is the peer of the party P in the Test session (defined below).*
- *$\mathcal{A}$ issues an Establish($P, \mathbb{S}_P$) query on party P or an Establish($P', \mathbb{S}_{P'}$) query on party P'.*

The adversary begins the second phase of the game by choosing a fresh session sid^* and issuing a Test(sid^*) query, where the Test query are defined as follows:
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. The 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 6 (ABCK security). *We say that an ABAKE scheme Π is secure in the ABCK model, if the following conditions hold:*

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

The following notions of security may also be considered depending upon the types of oracle queries the adversary is allowed to ask:

1. Key Independence: An adversary $\mathcal{A}$ can ask Send(*message*), SessionKeyReveal(*sid*), Establish($P, \mathbb{S}_P$), but not PartyCorruption query.
2. Forward Secrecy: An adversary $\mathcal{A}$ can ask all the queries as before for key independence, and in addition PartyCorruption query. Note that forward secrecy implies key independence.
3. Key Compromise Impersonation (KCI): Here the adversary $\mathcal{A}$ can ask all the queries for both key independence and forward secrecy as before. In particular we allow the adversary to corrupt the owner of the test session which captures KCI attacks.

Our model also captures these security properties by giving the adversary access to the appropriate oracle access.

3 Our Construction

Design Rationale: Suppose two users P_i and P_j wish to establish a common session key among each other. Each user obtains his private key from Private Key Generator (PKG) after proving he is a legitimate user. In order to validate these private key components, the user performs the key sanity check mechanism. If it passes, then as a pre-processing phase, each user formulates several access structures he wants to be satisfied by the other user's attribute vector. Now, the two users participate in key agreement session. User P_i checks whether the session state information obtained by P_j is valid. If yes, P_i can compute the common shared secret key if and only if his attribute vector satisfies the access structure used to calculate the session state information sent by P_j, and vice versa. In this construction, access structures are boolean formulas which are represented by LSSS. The main idea of our construction is that the secret of our LSSS is chosen as one of the ephemeral keys t_k ($k \in \{i, j\}$) and as part of session state we are using the share values as Diffie-Hellman exponents. If the other party has legitimate attributes, he can exponentiate the constants generated in the secret reconstruction phase corresponding to the rows of the secret reconstruction submatrix

of LSSS and get back the Diffie-Hellman value of the secret, i.e., g^{t_k}. Note that this value g^{t_k} is bound to the values $\widetilde{c}_k$, $\widetilde{b}_k$ and $\widetilde{e}_k$ and also used in the construction of the shared secret key. So if the other party does not possess the appropriate attributes he cannot construct the g^{t_k} value and hence the session keys will not be in agreement.

We now give the detailed description of our one-round attribute-based key agreement protocol.

►Setup: The Key Generation Centre (KGC) or Private Key Generator (PKG) chooses a group $\mathbb{G}$ of prime order p. Let g be the generator of group $\mathbb{G}$. The PKG picks $s_1, s_2 \in_R \mathbb{Z}_p^*$, and 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: $H_1 : \{0,1\}^* \to \mathbb{G}$, $H_2 : \{0,1\}^* \to \mathbb{Z}_p^*$. It then makes $params$ public and keeps msk to itself, where $params$ and msk are defined as follows:

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

►KeyGen: On input an attribute vector $\vec{\mathbb{S}}^{(i)} = (\mathbb{S}_1^{(i)}, \mathbb{S}_2^{(i)}, \cdots, \mathbb{S}_{m_i}^{(i)})$ corresponding to a party P_i, the PKG does the following to generate its private key :

- Choose $x_i \in_R \mathbb{Z}_p^*$.
- Compute $u_{1,i} = g^{x_i}$ and set $h_i = H_1\big(\vec{\mathbb{S}}^{(i)}\big)$.
- Compute $v_{1,i} = {h_i}^{x_i}$.
- Pick $r_i \in_R \mathbb{Z}_p^*$, compute $u_{2,i} = g^{r_i}$ and $v_{2,i} = {h_i}^{r_i}$.
- Set $c_i = H_2\left(u_{1,i}\right)$, $b_i = H_2(u_{1,i}, v_{1,i}, u_{2,i}, v_{2,i}, 0)$ and $e_i = H_2(u_{1,i}, v_{1,i}, u_{2,i}, v_{2,i}, 1)$.
- Compute $d_{1,i} = x_i + s_1 \cdot c_i$ where s_1 is the master secret key.
- Compute $d_{2,i} = x_i + r_i \cdot b_i + s_2 \cdot e_i$ and $\hat{h}_i = {h_i}^{s_2}$

Finally, PKG sends $\langle u_{1,i}, v_{1,i}, u_{2,i}, v_{2,i}, d_{1,i}, d_{2,i}, \hat{h}_i \rangle$ to the party P_i.

Similarly party P_j with attribute vector $\vec{\mathbb{S}}^{(j)} = (\mathbb{S}_1^{(j)}, \mathbb{S}_2^{(j)}, \cdots, \mathbb{S}_{m_j}^{(j)})$ gets it's private key $\langle u_{1,j}, v_{1,j}, u_{2,j}, v_{2,j}, d_{1,j}, d_{2,j}, \hat{h}_j \rangle$ from the PKG corresponding to his attributes where the respective components of the private key of P_j are computed in a similar fashion as of P_i.

The users after receiving the private key components from the PKG performs Key Sanity Check as shown in Appendix B to ensure the correctness of the received components.

►KeyAgreement: The two parties P_i and P_j with attribute vectors $\vec{\mathbb{S}}^{(i)}$ and $\vec{\mathbb{S}}^{(j)}$ respectively get their respective private keys from the PKG. They now proceed with the key agreement phase as follows:

- First, P_i decides an access structure $\mathbb{A}^{(i)}$ and he hopes that the set of attributes $\vec{\mathbb{S}}^{(j)}$ of party P_j satisfies $\mathbb{A}^{(i)}$. Note that the access structure $\mathbb{A}^{(i)}$ will be represented by $(M^{(i)}, \rho^{(i)})$ where $M^{(i)}$ is the $l_i \times n_i$ share generating matrix and $\rho^{(i)}$ is the injective labeling function corresponding to this matrix $M^{(i)}$ that maps the rows of $M^{(i)}$ to attributes in our case. Similarly, party P_j also decides an access structure $\mathbb{A}^{(j)}$ and he hopes that the set of attributes $\vec{\mathbb{S}}^{(i)}$ of party P_i satisfies $\mathbb{A}^{(j)}$. This access structure $\mathbb{A}^{(j)}$ will also be specified by a $l_j \times n_j$ share generating matrix $M^{(j)}$ and the injective labeling function $\rho^{(j)}$ that maps the rows of $M^{(j)}$ to attributes.
- Party P_i then chooses an ephemeral secret component $w_i \in_R \mathbb{Z}_p^*$ and computes $W_i = g^{w_i}$. Similarly party P_j chooses an ephemeral secret component $w_j \in_R \mathbb{Z}_p^*$ and computes $W_j = g^{w_j}$.
- Party P_i (resp., P_j) also chooses a random vector $\vec{\sigma}^{(i)} \in_R (\mathbb{Z}_p^*)^{n_i}$ where $\vec{\sigma}^{(i)}$ is of the form $\vec{\sigma}^{(i)} = (t_i, {\sigma_2}^{(i)}, \cdots \sigma_{n_i}^{(i)})$. Similarly party P_j chooses $\vec{\sigma}^{(j)} \in_R (\mathbb{Z}_p^*)^{n_j}$ where

$\vec{\sigma}^{(j)}$ is of the form $\vec{\sigma}^{(j)} = (t_j, {\sigma_2}^{(j)}, \cdots \sigma_{n_j}^{(j)})$. Here t_i in the place of ${\sigma_1}^{(i)}$ (resp., t_j in the place of ${\sigma_1}^{(j)}$) represents the secret value corresponding to the underlying LSSS scheme.

- Party P_i now computes the following values:
  1. Compute $X_i = g^{t_i}$.
  2. Compute $\widetilde{c}_i = H_2(X_i, u_{1,i})$, $\widetilde{b}_i = H_2(X_i, u_{1,i}, v_{1,i}, u_{2,i}, v_{2,i}, 0)$ and $\widetilde{e}_i = H_2(X_i, u_{1,i}, v_{1,i}, u_{2,i}, v_{2,i}, 1)$.
  3. For each row $\tau \in \{1, 2, \ldots, l_i\}$, compute $T_\tau^{(i)} = g^{\vec{M}_\tau^{(i)} \cdot \vec{\sigma}^{(i)}}$, where $\vec{M}_\tau^{(i)}$ is τth row of the matrix $M^{(i)}$.
  4. Compute $\eta_i = w_i + d_{1,i} \cdot H_2(\{T_\tau^{(i)}\}_{\tau=1}^{l_i}, W_i, M^{(i)}, \rho^{(i)})$.

  Party P_i then sends the values $\big[\vec{F}^{(i)} = (u_{1,i}, v_{1,i}, d_{2,i}, b_i, e_i, \widetilde{c}_i, \widetilde{b}_i, \widetilde{e}_i, \hat{h}_i, M^{(i)}, \rho^{(i)})$, $\vec{V}^{(i)} = (\eta_i, \{T_\tau^{(i)}\}_{\tau=1}^{l_i}, W_i)\big]$ to party P_j as shown in Table 2.
- Similarly party P_j also computes the values:
  1. Compute $X_j = g^{t_j}$.
  2. Compute $\widetilde{c}_j = H_2(X_j, u_j^{(1)})$, $\widetilde{b}_j = H_2(X_j, u_{1,j}, v_{1,j}, u_{2,j}, v_{2,j}, 0)$ and $\widetilde{e}_j = H_2(X_j, u_{1,j}, v_{1,j}, u_{2,j}, v_{2,j}, 1)$.
  3. For each row $\tau \in \{1, 2, \ldots, l_j\}$, compute $T_\tau^{(j)} = g^{\vec{M}_\tau^{(j)} \cdot \vec{\sigma}^{(j)}}$, where $\vec{M}_\tau^{(j)}$ is τth row of the matrix $M^{(j)}$.
  4. Compute $\eta_j = w_j + d_{1,j} \cdot H_2(\{T_\tau^{j}\}_{\tau=1}^{l_j}, W_j, M^{(j)}, \rho^{(j)})$.

  Party P_j then sends the values $\big[\vec{F}^{(j)} = (u_{1,j}, v_{1,j}, d_{2,j}, b_j, e_j, \widetilde{c}_j, \widetilde{b}_j, \widetilde{e}_j, \hat{h}_j, M^{(j)}, \rho^{(j)})$, $\vec{V}^{(j)} = (\eta_j, \{T_\tau^{(j)}\}_{\tau=1}^{l_j}, W_j)\big]$ to party P_i.
- Party P_j on receiving the tuple $[\vec{F}^{(i)}, \vec{V}^{(i)}]$ from party P_i checks the consistency of the individual components as shown in Table 2. We note that to check the consistency of the hash values $\widetilde{c}_j, \widetilde{b}_j, \widetilde{e}_j$, the party P_j needs to get back the secret component X_i which is only possible if the party P_j possesses an authorized set, i.e., the party P_j has the required attributes to compute t_i. In other words as shown in subsection 2.2 and 2.3, if the party P_j has permissible attributes that comprises an authorized set, he can compute the required constants corresponding to the submatrix that will be generated and the secret value can be reconstructed back from these constants and the corresponding shares of the party.

Remark 1. We now show the correctness of the steps 2 (b) of our key agreement protocol.

$$\left(\frac{g^{d_{2,i}}}{u_{1,i}.{y_2}^{e_i}}\right)^{{b_i}^{-1}} = \left(\frac{g^{x_i + r_i \cdot b_i + s_2 \cdot e_i}}{g^{x_i} \cdot g^{s_2 \cdot e_i}}\right)^{{b_i}^{-1}} = \left(g^{r_i \cdot b_i}\right)^{{b_i}^{-1}} = g^{r_i} = u_{2,i}.$$

$$\left(\frac{{h_i}^{d_{2,i}}}{v_{1,i} \cdot ({h_i}^{s_2})^{e_i}}\right)^{{b_i}^{-1}} = \left(\frac{{h_i}^{x_i + r_i \cdot b_i + s_2 \cdot e_i}}{{h_i}^{x_i} \cdot ({h_i}^{s_2})^{e_i}}\right)^{{b_i}^{-1}} = \left({h_i}^{r_i \cdot b_i}\right)^{{b_i}^{-1}} = {h_i}^{r_i} = v_{2,i}.$$

If the attribute vector $\vec{\mathbb{S}}_j$ satisfies the access structure $\mathbb{A}^{(i)}$ of user P_i, then $\sum_{\tau \in I} \omega_\tau \vec{M}_\tau^{(i)} \cdot \vec{\sigma}^{(i)} = (\sum_{\tau \in I} \omega_\tau \vec{M}_\tau^{(i)}) \cdot \vec{\sigma}^{(i)} = (1, 0, \ldots, 0) \cdot (t_i, {\sigma_2}^{(i)}, \cdots \sigma_{n_i}^{(i)}) = t_i$.
Hence, $X_i' = \prod_{\tau \in I} (T_\tau^{(i)})^{\omega_\tau} = \prod_{\tau \in I} (g^{\vec{M}_\tau^{(i)} \cdot \vec{\sigma}^{(i)}})^{\omega_\tau} = g^{\sum_{\tau \in I} \omega_\tau \vec{M}_\tau^{(i)} \cdot \vec{\sigma}^{(i)}} = g^{t_i}$.

Therefore, the components that are recomputed are valid and if the attributes of party P_j satisfy the access structure of party P_i, the computation of $\widetilde{b}_i = H_2({\sigma_1}^{(i)}, u_{1,i}, v_{1,i}, u_{2,i}, v_{2,i}, 0)$ and $\widetilde{e}_i = H_2({\sigma_1}^{(i)}, u_{1,i}, v_{1,i}, u_{2,i}, v_{2,i}, 1)$ will match with the one obtained from P_i. So, any tampering done with these values during transit will always be caught.

Party P_i	Party P_j
1. Local Computation:	**1. Local Computation:**
(a) Choose $w_i \in_R \mathbb{Z}_p^*$,	**(a)** Choose $w_j \in_R \mathbb{Z}_p^*$,
(b) Compute $W_i = g^{w_i}$	**(b)** Compute $W_j = g^{w_j}$
(c) Choose $\vec{\sigma}^{(i)} = (t_i, {\sigma_2}^{(i)}, \ldots, \sigma_{n_i}^{(i)}) \in_R (\mathbb{Z}_p^*)^{n_i}$, where t_i is a secret value.	**(c)** Chooses $\vec{\sigma}^{(j)} = (t_j, {\sigma_2}^{(j)}, \ldots, \sigma_{n_j}^{(j)}) \in_R (\mathbb{Z}_p^*)^{n_j}$, where t_j is a secret value.
(d) $X_i = g^{t_i}$	**(d)** $X_j = g^{t_j}$
(e) Compute: **(i)** $\widetilde{c}_i = H_2(X_i, u_{1,i})$	**(e)** Compute: **(i)** $\widetilde{c}_j = H_2(X_j, u_{1,j})$
(ii) $\widetilde{b}_i = H_2\left(X_i, u_{1,i}, v_{1,i}, u_{2,i}, v_{2,i}, 0\right)$	**(ii)** $\widetilde{b}_j = H_2\left(X_j, u_{1,j}, v_{1,j}, u_{2,j}, v_{2,j}, 0\right)$
(iii) $\widetilde{e}_i = H_2\left(X_i, u_{1,i}, v_{1,i}, u_{2,i}, v_{2,i}, 1\right)$	**(iii)** $\widetilde{e}_j = H_2\left(X_j, u_{1,j}, v_{1,j}, u_{2,j}, v_{2,j}, 1\right)$
(f) For each row $\tau \in \{1, 2, \ldots, l_i\}$, compute $T_\tau^{(i)} = g^{\vec{M}_\tau^{(i)} \cdot \vec{\sigma}^{(i)}}$, where $\vec{M}_\tau^{(i)}$ is τth row of the matrix $M^{(i)}$.	**(f)**For each row $\tau \in \{1, 2, \ldots, l_j\}$, compute $T_\tau^{(j)} = g^{\vec{M}_\tau^{(j)} \cdot \vec{\sigma}^{(j)}}$, where $\vec{M}_\tau^{(j)}$ is τth row of the matrix $M^{(j)}$.
(g) $\eta_i = w_i + d_{1,i} \cdot H_2\big(\{T_\tau^{(i)}\}_{\tau=1}^{l_i}, W_i, M^{(i)}, \rho^{(i)}\big)$	**(g)** $\eta_j = w_j + d_{1,j} \cdot H_2\big(\{T_\tau^{(j)}\}_{\tau=1}^{l_j}, W_j, M^{(j)}, \rho^{(j)}\big)$
$\xrightarrow{\vec{F}^{(i)} = (u_{1,i}, v_{1,i}, d_{2,i}, b_i, e_i, \widetilde{c}_i, \widetilde{b}_i, \widetilde{e}_i, \hat{h}_i, M^{(i)}, \rho^{(i)}),\ \ \vec{V}^{(i)} = (\eta_i, \{T_\tau^{(i)}\}_{\tau=1}^{l_i}, W_i)}$	
$\xleftarrow{\vec{F}^{(j)} = (u_{1,j}, v_{1,j}, d_{2,j}, b_j, e_j, \widetilde{c}_j, \widetilde{b}_j, \widetilde{e}_j, \hat{h}_j, M^{(j)}, \rho^{(j)}),\ \ \vec{V}^{(j)} = (\eta_j, \{T_\tau^{(j)}\}_{\tau=1}^{l_j}, W_j)}$	
2. Verification:	**2. Verification:**
(a) ***Check 1: (Membership Testing)***	**(a)** ***Check 1: (Membership Testing)***
Check if **(i)** ${F_1}^{(j)}, {F_2}^{(j)}, {F_9}^{(j)} \in \mathbb{G}^*$;	Check if **(i)** ${F_1}^{(i)}, {F_2}^{(i)}, {F_9}^{(i)} \in \mathbb{G}^*$;
(ii) $\{{F_k}^{(j)}\}_{k=3}^{8} \subset \mathbb{Z}_p^*$	**(ii)** $\{{F_k}^{(i)}\}_{k=3}^{8} \subset \mathbb{Z}_p^*$
If $\neg$(**a**) or $\neg$(**b**) or both, **Abort**	If $\neg$(**a**) or $\neg$(**b**) or both, **Abort**
(b) ***Check for correctness of*** $\vec{F}^{(j)}$***:***	**(b)** ***Check for correctness of*** $\vec{F}^{(i)}$***:***
Compute **(i)** $u'_{2,j} = \left(\dfrac{g^{d_{2,j}}}{u_{1,j} \cdot {y_2}^{e_j}}\right)^{{b_j}^{-1}}$	Compute **(i)** $u'_{2,i} = \left(\dfrac{g^{d_{2,i}}}{u_{1,i} \cdot {y_2}^{e_i}}\right)^{{b_i}^{-1}}$
(ii) $v'_{2,j} = \left(\dfrac{{h_j}^{d_{2,j}}}{v_{1,j} \cdot (\hat{h}_j)^{e_j}}\right)^{{b_j}^{-1}}$	**(ii)** $v'_{2,i} = \left(\dfrac{{h_i}^{d_{2,i}}}{v_{1,i} \cdot (\hat{h}_i)^{e_i}}\right)^{{b_i}^{-1}}$
(iii) If $\vec{\mathbb{S}}^{(i)}$ satisfies $\mathbb{A}^{(j)}$, then compute $X'_j = \prod_{\tau \in I} (T_\tau^{(j)})^{\omega_\tau}$ where $I = \{\tau : \rho(\tau) \in \vec{\mathbb{S}}^{(i)}\}$ and $\{\omega_\tau\}_{\tau \in I} \subset \mathbb{Z}_p^*$	**(iii)** If $\vec{\mathbb{S}}^{(j)}$ satisfies $\mathbb{A}^{(i)}$, then compute $X'_i = \prod_{\tau \in I} (T_\tau^{(i)})^{\omega_\tau}$ where $I = \{\tau : \rho(\tau) \in \vec{\mathbb{S}}^{(j)}\}$ and $\{\omega_\tau\}_{\tau \in I} \subset \mathbb{Z}_p^*$
(iv) ***Check 2 :*** Check if	**(iv)** ***Check 2 :*** Check if
$\widetilde{c}_j \stackrel{?}{=} H_2(X'_j, u_{1,j})$	$\widetilde{c}_i \stackrel{?}{=} H_2(X'_i, u_{1,i})$
$\widetilde{b}_j \stackrel{?}{=} H_2(X'_j, u_{1,j}, v_{1,j}, u'_{2,j}, v'_{2,j}, 0)$	$\widetilde{b}_i \stackrel{?}{=} H_2(X'_i, u_{1,i}, v_{1,i}, u'_{2,i}, v'_{2,i}, 0)$
$\widetilde{e}_j \stackrel{?}{=} H_2(X'_j, u_{1,j}, v_{1,j}, u'_{2,j}, v'_{2,j}, 1)$	$\widetilde{e}_i \stackrel{?}{=} H_2(X'_i, u_{1,i}, v_{1,i}, u'_{2,i}, v'_{2,i}, 1)$
If any of them not equal, **Abort**, else proceed.	If any of them not equal, **Abort**, else proceed.

Party P_i	Party P_j
(c) ***Check for correctness of*** $\vec{V}^{(j)}$***:***	**(c)** ***Check for correctness of*** $\vec{V}^{(i)}$***:***
Check 3 : Check if	***Check 3 :*** Check if
$\left[\frac{g^{\eta_j}}{(g^{x_j})^{\xi_j}\cdot(y_1)^{c_j\cdot\xi_j}}\right] \stackrel{?}{=} g^{w_j}$	$\left[\frac{g^{\eta_i}}{(g^{x_i})^{\xi_i}\cdot(y_1)^{c_i\cdot\xi_i}}\right] \stackrel{?}{=} g^{w_i}$
here $c_j = H_2(u_{1,j}), \xi_j = H_2(\{T_\tau^{(j)}\}, W_j, M^{(j)}, \rho^{(j)})$.	where $c_i = H_2(u_{1,i}), \xi_i = H_2(\{T_\tau^{(i)}\}, W_i, M^{(i)}, \rho^{(i)})$.
If not equal **Abort**, else proceed to step 3.	If not equal **Abort**, else proceed to step 3.
3. ***Shared secret key generation:***	**3.** ***Shared secret key generation:***
Compute $Z_1 = \left(u_{1,j}\cdot {y_1}^{c_j}\cdot X'_j\right)^{d_{1,i}+t_i}$	Compute $Z_1 = \left(u_{1,i}\cdot {y_1}^{c_i}\cdot X'_i\right)^{d_{1,j}+t_j}$
$Z_2 = v_{1,i}\cdot v_{1,j}$	$Z_2 = v_{1,j}\cdot v_{1,i}$
$Z_3 = \left(X'_j\right)^{t_i}$.	$Z_3 = \left(X'_i\right)^{t_j}$.
$Z = H_2(Z_1, Z_2, Z_3)$.	$Z = H_2(Z_1, Z_2, Z_3)$;
Return Z.	**Return** Z.

Table 2. Description of the Key Agreement protocol.

Remark 2. Check 3 is done to verify the value of $\eta_i = w_i + d_{i1}\cdot H_2(\{T_\tau^{(i)}\}_{\tau=1}^{l_i}, W_i, M^{(i)}, \rho^{(i)})$ which ensures that an active adversary cannot tamper with the components exchanged and affect the shared secret key generation.

$$\begin{aligned}
&\frac{g^{\left(w_i + d_{1,i}\cdot H_2(\{T_\tau^{(i)}\}_{\tau=1}^{l_i}, W_i, M^{(i)}, \rho^{(i)})\right)}}{(g^{x_i})^{H_2(\{T_\tau^{(i)}\}_{\tau=1}^{l_i}, W_i, M^{(i)}, \rho^{(i)})}\cdot(y_1)^{c_i\cdot H_2(\{T_\tau^{(i)}\}_{\tau=1}^{l_i}, W_i, M^{(i)}, \rho^{(i)})}}\\
&= \frac{g^{\left(w_i + (x_i + s_1\cdot c_i)\cdot H_2(\{T_\tau^{(i)}\}_{\tau=1}^{l_i}, W_i, M^{(i)}, \rho^{(i)})\right)}}{g^{x_i\cdot H_2(\{T_\tau^{(i)}\}_{\tau=1}^{l_i}, W_i, M^{(i)}, \rho^{(i)})}\cdot g^{s_1\cdot c_i\cdot H_2(\{T_\tau^{(i)}\}_{\tau=1}^{l_i}, W_i, M^{(i)}, \rho^{(i)})}}\\
&= g^{w_i}.
\end{aligned}$$

We now show the correctness of our protocol, i.e., the keys computed by both the parties are the *same*.

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

Proof. Party P_i computes: since $u_{1,j} = g^{x_j}$ and $x_j + s_1\cdot c_j = d_{1,j}$, we have
$Z_1 = \left(u_{1,j}{y_1}^{c_j}X'_j\right)^{d_{1,i}+t_i} = \left(u_{1,j}{y_1}^{c_j}g^{t_j}\right)^{d_{1,i}+t_i} = \left(g^{(x_j+s_1\cdot c_j+t_j)}\right)^{d_{1,i}+t_i} = g^{(d_{1,j}+t_j)(d_{1,i}+t_i)}$.
Party P_j computes: since $u_{1,i} = g^{x_i}$ and $x_i + s_1\cdot c_i = d_{1,i}$, we have
$Z_1 = \left(u_{1,i}{y_1}^{c_i}X'_i\right)^{d_{1,j}+t_j} = \left(u_{1,i}{y_1}^{c_i}g^{t_i}\right)^{d_{1,j}+t_j} = \left(g^{(x_i+s_1c_i+t_i)}\right)^{d_{1,j}+t_j} = g^{(d_{1,i}+t_i)(d_{1,j}+t_j)}$.

Thus, Z_1 computed by both the parties are identical. Z_2 and Z_3 are also consistent. Hence, the final shared secret key Z computed by both the parties is consistent. □

3.1 Complexity Analysis of Our Protocol

In this section we give the complexity analysis of our protocol. Firstly, we note the computational cost of each party is dominated by the number of exponentiations it needs to do in the actual execution of the protocol. In the naive implementation of our protocol, the number of exponentiations performed by each party will depend on the size of the share generating matrix for share generation and secret reconstruction phase. However, here we show by doing appropriate preprocessing, the number of exponentiations at each party can be made $O(1)$ (precisely 8). The detailed analysis is shown as follows:

1. In the Local Computation phase of our protocol, each party needs to perform 2 exponentiations corresponding to steps 1 (b), 1 (d) respectively.
2. In the naive implementation of our protocol, in the share generation phase (step 1 (f)), each party needs to perform $size(M)$ many exponentiations where $size(M)$ denotes the number of rows in the share generating matrix M. However, it is to be noted that the computation of $T_\tau^{(i)} = g^{\vec{M}_\tau^{(i)} \cdot \vec{\sigma}^{(i)}}$ values can be precomputed in the preprocessing steps as the share generation phase does not depend on the other party's access structure. So each party can locally choose random access structure that it needs the other party to satisfy and it precomputes the $T_\tau^{(i)}$ values and stores it in a table T say. So the actual computational cost for each party in the actual protocol execution is independent of the cost of share generation. For each execution of the protocol a party can simply pick up an unused tuple of values from the table T and use it for the current session.
3. In step 2 (b) (i) and (ii) of the verification phase, each party can get away with performing no exponentiation in the actual protocol execution. This is beacuse the checks does not require the knowledge of ephemeral secret keys w_i or w_j or the value η_i or η_j. The components $\langle u_{1,i}, v_{1,i}, d_{2,i}, b_i, e_i, \hat{h}_i \rangle$, are sent only once by each party because they are part of static private key of that party and are invariant across all sessions in which this party is involved. So a party can perform these checks in the preprocessing step for the first time itself; next time onwards it need not do these checks.
4. In step 2 (b) (iii), each party needs to reconstruct back the secret value if he/she is a legitimate party, i.e., its attributes satisfy the access structure of the other party. The secret reconstruction cost of each party comes for free since we are working with access structures specified as boolean formula. This is due to the fact that the secret can be constructed by using Gaussian Elimination method in $O(n^3)$ time for access policies expressed as boolean formulas (for more details, see [24]).
5. For performing the checks 3 and 4 in step 2 (c), each party needs to do 4 exponentiations in all. Finally in step 3 in the Shared secret key generation phase, each party neds to perform 2 exponentiations (the value $(y_1)^{c_j}$ can again be precomputed).

So in total each party needs to perform 8 exponentiations in the actual execution of our protocol by performing these preprocessing steps as mentioned.

4 Security Proof

Due to space constraints we present the formal security proof for the proposed protocol (described in the previous section) in Appendix C. The detailed probability calculation is also included in Appendix C.1. The proof is based on the ABCK security model

described in section 2.5. The scheme is proved secure under the Strong Diffie-Hellman (SDH) assumption in the random oracle model. The security proof is modeled as a game between the challenger and the adversary. The proof of the following Theorem 1 is presented in Appendix C.

Theorem 1. *Under the SDH assumption in $\mathbb{G}$ and the random oracle model, our protocol is secure in the ABCK model. If ϵ is the probability of the adversary in distinguishing between a random shared secret key and a valid shared secret key in the test session, the probability of solving the underlying SDH problem, ϵ' is given by:*

$$\epsilon' = \epsilon \cdot \frac{1}{h_5}\big(1 - \frac{1}{q_E + 2}\big)^{q_E+1} \cdot \big(\frac{1}{q_E + 2}\big)$$

where q_E = Number of key extract or Party Corruption queries and h_5 is the number of queries on the hash oracle H_2 of the form $\langle Z_1, Z_2, Z_3 \rangle$.

5 Additional Security Properties

The proposed protocol also 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 previously established shared secret keys. 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 freshness property of our ABCK model allows the adversary to corrupt both the parties in the test session. In the security proof also the challenger can perfectly simulate the PartyCorruption queries. The proposed scheme offers forward secrecy.

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

Resistance to Ephemeral Key Compromise Impersonation: Generally users pick the ephemeral keys (w_i, g^{w_i}) 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 party P_i without knowing the private key of P_i, it cannot generate the components $d_{2,i}$ and the signature on g^{w_i} (we assume that w_i is erased immediately after the signature on g^{w_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.

6 Conclusion

We propose a single-round bipartite attribute based AKE. The main advantages of our protocol is that it is efficient, 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 number of exponentiations per party and also involves no pairing operations. 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 standard model without using attribute-based encryption schemes or signatures as basic building blocks, i.e, designing an ABAKE scheme in standard model handcrafted from scratch.

References

1. Abe, M., Kiltz, E., Okamoto, T.: Compact cca-secure encryption for messages of arbitrary length. In: Public Key Cryptography–PKC 2009, pp. 377–392. Springer (2009)
2. Ateniese, G., Kirsch, J., Blanton, M.: Secret handshakes with dynamic and fuzzy matching. In: NDSS. vol. 7, pp. 1–19 (2007)
3. Beimel, A.: Secure schemes for secret sharing and key distribution. Ph.D. thesis, Technion-Israel Institute of technology, Faculty of computer science (1996)
4. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Advances in CryptologyCRYPTO93. pp. 232–249. Springer (1994)
5. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-policy attribute-based encryption. In: Security and Privacy, 2007. SP'07. IEEE Symposium on. pp. 321–334. IEEE (2007)
6. Birkett, J., Stebila, D.: Predicate-based key exchange. In: Information Security and Privacy. pp. 282–299. Springer (2010)
7. Bohli, J.M., Vasco, M.I.G., Steinwandt, R.: Secure group key establishment revisited. International Journal of Information Security 6(4), 243–254 (2007)
8. Canetti, R., Krawczyk, H.: Analysis of key-exchange protocols and their use for building secure channels. In: Advances in CryptologyEUROCRYPT 2001, pp. 453–474. Springer (2001)
9. Emura, K., Miyaji, A., Rahman, M.S.: Dynamic attribute-based signcryption without random oracles. International Journal of Applied Cryptography 2(3), 199–211 (2012)
10. Gorantla, M.C., Boyd, C., Nieto, J.M.G.: Attribute-based authenticated key exchange. In: Information Security and Privacy. pp. 300–317. Springer (2010)
11. Goyal, V., Jain, A., Pandey, O., Sahai, A.: Bounded ciphertext policy attribute based encryption. In: Automata, languages and programming, pp. 579–591. Springer (2008)
12. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: Proceedings of the 13th ACM conference on Computer and communications security. pp. 89–98. Acm (2006)
13. Karchmer, M., Wigderson, A.: On span programs. In: Structure in Complexity Theory Conference. pp. 102–111 (1993)
14. LaMacchia, B., Lauter, K., Mityagin, A.: Stronger security of authenticated key exchange. In: Provable Security, pp. 1–16. Springer (2007)

15. Lewko, A., Okamoto, T., Sahai, A., Takashima, K., Waters, B.: Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In: Advances in Cryptology–EUROCRYPT 2010, pp. 62–91. Springer (2010)
16. Li, J., Au, M.H., Susilo, W., Xie, D., Ren, K.: Attribute-based signature and its applications. In: Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security. pp. 60–69. ACM (2010)
17. Maji, H.K., Prabhakaran, M., Rosulek, M.: Attribute-based signatures. In: Topics in Cryptology–CT-RSA 2011, pp. 376–392. Springer (2011)
18. Nikov, V., Nikova, S.: New monotone span programs from old. IACR Cryptology ePrint Archive 2004, 282 (2004)
19. Okamoto, T., Takashima, K.: Fully secure functional encryption with general relations from the decisional linear assumption. In: Advances in Cryptology–CRYPTO 2010, pp. 191–208. Springer (2010)
20. Okamoto, T., Takashima, K.: Efficient attribute-based signatures for non-monotone predicates in the standard model. Cloud Computing, IEEE Transactions on 2(4), 409–421 (2014)
21. Pandit, T., Pandey, S.K., Barua, R.: Attribute-based signcryption: Signer privacy, strong unforgeability and ind-cca2 security in adaptive-predicates attack. In: Provable Security, pp. 274–290. Springer (2014)
22. Sahai, A., Waters, B.: Fuzzy identity-based encryption. In: Advances in Cryptology–EUROCRYPT 2005, pp. 457–473. Springer (2005)
23. Shahandashti, S.F., Safavi-Naini, R.: Threshold attribute-based signatures and their application to anonymous credential systems. In: Progress in Cryptology–AFRICACRYPT 2009, pp. 198–216. Springer (2009)
24. Sreenivasa Rao, Y., Dutta, R.: Attribute-based key-insulated signature for boolean formula. International Journal of Computer Mathematics (ahead-of-print), 1–25 (2015)
25. Steinwandt, R., Corona, A.S.: Attribute-based group key establishment. IACR Cryptology ePrint Archive 2010, 235 (2010)
26. Vivek, S.S., Selvi, S.S.D., Venkatesan, L.R., Rangan, C.P.: Efficient, pairing-free, authenticated identity based key agreement in a single round. In: Provable Security, pp. 38–58. Springer (2013)
27. Waters, B.: Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In: Public Key Cryptography–PKC 2011, pp. 53–70. Springer (2011)
28. Yoneyama, K.: Strongly secure two-pass attribute-based authenticated key exchange. In: Pairing-Based Cryptography-Pairing 2010, pp. 147–166. Springer (2010)
29. Yoneyama, K.: Two-party round-optimal session-policy attribute-based authenticated key exchange without random oracles. In: Information Security and Cryptology-ICISC 2011, pp. 467–489. Springer (2011)
30. Yoneyama, K.: Generic construction of two-party round-optimal attribute-based authenticated key exchange without random oracles. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences 96(6), 1112–1123 (2013)

A Complexity Assumptions

Definition 7 (Computation Diffie-Hellman (CDH) Problem). *Given* $(g, g^a, g^b) \in \mathbb{G}^3$ *for unknown* $a, b \in \mathbb{Z}_p^*$*, where* $\mathbb{G}$ *is a cyclic prime order multiplicative group with* g

as a generator and p is 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

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

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

Definition 8 (Decisional Diffie-Hellman (DDH) Problem). *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 p as 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

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

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

Definition 9 (Strong Diffie Hellman (SDH) Problem [1]). *Let κ be the security parameter and $\mathbb{G}$ be a multiplicative group of order p. 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 CDH problem using a DDH oracle)*
The advantage of an adversary $\mathcal{A}$ in solving the SDH problem is defined as the probability with which $\mathcal{A}$ solves the above SDH problem.

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

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

B Secret Key Sanity Check

After receiving the private key from the PKG in the key extraction 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. $c_i = H_2(u_{1,i})$

b. $b_i = H_2(u_{1,i}, v_{1,i}, u_{2,i}, v_{2,i}, 0)$

c. $e_i = H_2(u_{1,i}, v_{1,i}, u_{2,i}, v_{2,i}, 1)$

Test 1: Check if $\frac{g^{d_{1,i}}}{y_1^{H_2(u_{1,i})}} \stackrel{?}{=} u_{1,i}$.

This can be verified as $\frac{g^{x_i + s_1 \cdot c_i}}{g^{s_1 \cdot H_2(u_{1,i})}}$, where $c_i = H_2(u_{1,i})$.
This is equal to $g^{x_i} = u_{1,i}$. This check ensures the correctness of $d_{1,i}$ and $u_{1,i}$.

Test 2: Check if $\frac{g^{d_{2,i}}}{(u_{2,i})^{H_2(u_{1,i},v_{1,i},u_{2,i},v_{2,i},0)} \cdot y_2{}^{H_2(u_{1,i},v_{1,i},u_{2,i},v_{2,i},1)}} \stackrel{?}{=} u_{1,i}$.

This can be verified as $\frac{g^{(x_i+r_i \cdot b_i+s_2 \cdot e_i)}}{g^{r_i \cdot H_2(u_{1,i},v_{1,i},u_{2,i},v_{2,i},0)} \cdot g^{s_2 \cdot H_2(u_{1,i},v_{1,i},u_{2,i},v_{2,i},1)}} = g^{x_i} = u_{1,i}$, as $b_i = H_2(u_{1,i}, v_{1,i}, u_{2,i}, v_{2,i}, 0)$ and $e_i = H_2(u_{1,i}, v_{1,i}, u_{2,i}, v_{2,i}, 1)$.

This check ensures the correctness of $d_{2,i}, u_{2,i}, v_{1,i}, v_{2,i}$.

Test 3 : Check if $\frac{(h_i)^{d_{2,i}}}{v_{2,i}{}^{H_2(u_{1,i},v_{1,i},u_{2,i},v_{2,i},0)} \cdot (\hat{h}_i)^{H_2(u_{1,i},v_{1,i},u_{2,i},v_{2,i},1)}} \stackrel{?}{=} v_{1,i}$.

This can be verified as $\frac{h_i{}^{x_i+r_i \cdot b_i+s_2 \cdot e_i}}{(h_i{}^{r_i})^{H_2(u_{1,i},v_{1,i},u_{2,i},v_{2,i},0)} \cdot (\hat{h}_i)^{H_2(u_{1,i},v_{1,i},u_{2,i},v_{2,i},1)}} = h_i{}^{x_i} = v_{1,i}$ where $\hat{h}_i = h_i^{s_2}, b_i = H_2(u_{1,i}, v_{1,i}, u_{2,i}, v_{2,i}, 0)$ and $e_i = H_2(u_{1,i}, v_{1,i}, u_{2,i}, v_{2,i}, 1)$.

Test 3 ensures the correctness of $\hat{h}_i$. Test 2 and Test 3 ensures that g and h_i are raised to the same exponent x_i in $u_{1,i}$ and $v_{1,i}$ respectively.

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

C Proof of Theorem 1

Here we present the detailed security proof of Theorem 1.

Proof. We now give the formal security proof for our protocol from section 3.

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 $s_2 \in_R \mathbb{Z}_p^*$ 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 adversary queries the challenger for the hash value of the attribute vector $\vec{\mathbb{S}}_i$ corresponding to party P_i. If the H_1 *Oracle* was already queried with $\vec{\mathbb{S}}_i$ as input, the challenger returns the value computed before which is stored in the hash list L_{H_1} described below. Otherwise the challenger tosses a *coin* τ_i where the $Pr(\tau_i = 0) = \alpha$. The output of this oracle is defined as:

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

where $k_i \in_R \mathbb{Z}_p^*$. The challenger makes an entry in the hash list $L_{H_1} = \langle h_i, \vec{\mathbb{S}}_i, \tau_i, k_i \rangle$ for future use and returns h_i.

H_2 *Oracle* : When the adversary queries the hash function H_2 on any input say x, if the H_2 oracle was already queried before with this input, the challenger simply extracts the value from the hash list L_{H_2} described below and returns the value. Otherwise, the challenger chooses a random element say $y \in_R \mathbb{Z}_p^*$, makes an entry of the form $\langle x, y \rangle$ and returns y. For example, the adversary may query the challenger with inputs

$(u_{1,i})$ or $(u_{1,i}, v_{1,i}, u_{2,i}, v_{2,i}, 0)$ or $(u_{1,i}, v_{1,i}, u_{2,i}, v_{2,i}, 1)$ or $\left(\{T_\tau^{(i)}\}_{\tau=1}^{l_i}, W_i, M^{(i)}, \rho^{(i)}\right)$ or (Z_1, Z_2, Z_3).

Let h_1, h_2, h_3, h_4 and h_5 are the number of queries corresponding to each type of queries in the order mentioned above.

If the H_2 *Oracle* was already queried with $u_{1,i}$ as input, the challenger extracts the value c_i from the hash list L_{H_2} described below and returns the value. Otherwise, the challenger chooses a random value $c_i \in_R \mathbb{Z}_p^*$ respectively. It makes an entry in the hash list $L_{H_2} = \langle c_i, u_{1,i} \rangle$ and returns c_i. Similarly when the adversary queries the challenger with inputs $(u_{1,i}, v_{1,i}, u_{2,i}, v_{2,i}, 0)$ or $(u_{1,i}, v_{1,i}, u_{2,i}, v_{2,i}, 1)$, if the H_2 *Oracle* was already queried with $(u_{1,i}, v_{1,i}, u_{2,i}, v_{2,i}, 0)$ or $(u_{1,i}, v_{1,i}, u_{2,i}, v_{2,i}, 1)$ as input, the challenger extracts the value b_i or e_i from the hash list L_{H_2} described below and returns the value. Otherwise, the challenger chooses a random value $b_i \in_R \mathbb{Z}_p^*$ or $e_i \in_R \mathbb{Z}_p^*$. It makes an entry in the hash list $L_{H_2} = \langle b_i, (u_{1,i}, v_{1,i}, u_{2,i}, v_{2,i}, 0) \rangle$ or $L_{H_2} = \langle e_i, (u_{1,i}, v_{1,i}, u_{2,i}, v_{2,i}, 1) \rangle$ and returns b_i or e_i respectively. Similarly the rest of the queries will also be answered in a similar fashion.

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

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

The challenger does not know the master secret key s_1 as the master public key is set as $y_1 = g^a$ (implicitly 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 $c_i, b_i, e_i, {x_i}', {r_i}' \in_R \mathbb{Z}_p^*$.
- It sets $u_{1,i} = g^{x_i'} \cdot {y_1}^{-c_i}$.
- It sets $H_2(u_{1,i}) = c_i$ and stores the tuple $\langle c_i, u_{1,i} \rangle$ in the L_{H_2} list.
- It sets $d_{1,i} = x_i'$, $d_{2,i} = x_i' + r_i' \cdot b_i + s_2 \cdot e_i$ and $u_{2,i} = g^{r_i'} \cdot {y_1}^{{c_i \cdot b_i}^{-1}}$.
- It computes $v_{1,i} = g^{k_i \cdot x_i'} \cdot {y_1}^{-k_i \cdot c_i}$ and $v_{2,i} = g^{k_i \cdot r_i'} \cdot {y_1}^{{k_i \cdot c_i \cdot b_i}^{-1}}$.
- It also sets the hash function values $H_2(u_{1,i}, v_{1,i}, u_{2,i}, v_{2,i}, 0) = b_i$, $H_2(u_{1,i}, v_{1,i}, u_{2,i}, v_{2,i}, 1) = e_i$ and adds the tuples $\langle b_i, u_{1,i}, v_{1,i}, u_{2,i}, v_{2,i}, 0 \rangle$, $\langle e_i, u_{1,i}, v_{1,i}, u_{2,i}, v_{2,i}, 1 \rangle$ to the list L_{H_2}.
- It computes ${h_i}^{s_2}$.
- It returns the tuple $\langle u_{1,i}, v_{1,i}, u_{2,i}, v_{2,i}, d_{1,i}, d_{2,i}, {h_i}^{s_2} \rangle$ as the private key of the user with attribute vector $\vec{\mathbb{S}}_i$ and makes an entry in the list $L_E = \langle u_{1,i}, v_{1,i}, u_{2,i}, v_{2,i}, d_{1,i}, d_{2,i}, {h_i}^{s_2}, \vec{\mathbb{S}}_i \rangle$.

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 three checks given in Secret Key Sanity Check.

– ***Test 1 :*** Check if $\frac{g^{d_{1,i}}}{y_1^{H_2(u_{1,i})}} \stackrel{?}{=} u_{1,i}$.

This can be verified as $\frac{g^{x'_i}}{g^{a \cdot H_2(u_{1,i})}}$ where $c_i = H_2(u_{1,i})$.

This is equal to $g^{x'_i - a \cdot c_i} = g^{x'_i} \cdot {y_1}^{-c_i} = u_{1,i}$.

– ***Test 2 :*** Check if $\frac{g^{d_{2,i}}}{{u_{2,i}}^{H_2(u_{1,i},v_{1,i},u_{2,i},v_{2,i},0)} \cdot {y_2}^{H_2(u_{1,i},v_{1,i},u_{2,i},v_{2,i},1)}} \stackrel{?}{=} u_{1,i}$.

This follows as $\frac{g^{x'_i + r'_i \cdot b_i + s_2 \cdot e_i}}{\left(g^{r'_i} \cdot {y_1}^{c_i \cdot {b_i}^{-1}}\right)^{b_i} \cdot g^{s_2 \cdot e_i}} = g^{x'_i - a \cdot c_i} = g^{x'_i} \cdot {y_1}^{-c_i} = u_{1,i}$,

as $b_i = H_2(u_{1,i}, v_{1,i}, u_{2,i}, v_{2,i}, 0)$ and $e_i = H_2(u_{1,i}, v_{1,i}, u_{2,i}, v_{2,i}, 1)$.

– ***Test 3 :*** Check if $\frac{h_i^{d_{2,i}}}{{v_{2,i}}^{H_2(u_{1,i},v_{i,1},u_{i,2},v_{i,2},0)} \cdot ({h_i}^{s_2})^{H_2(u_{1,i},v_{1,i},u_{2,i},v_{2,i},1)}} \stackrel{?}{=} v_{1,i}$.

This follows as $\frac{h_i^{x'_i + r'_i \cdot b_i + s_2 \cdot e_i}}{\left(g^{k_i \cdot r'_i} \cdot {y_1}^{k_i \cdot c_i \cdot {b_i}^{-1}}\right)^{b_i} \cdot ({h_i}^{s_2})^{e_i}} = {h_i}^{x'_i} \cdot {y_1}^{-k_i \cdot c_i} = v_{1,i}$

where $b_i = H_2(u_{1,i}, v_{1,i}, u_{2,i}, v_{2,i}, 0)$ and $e_i = H_2(u_{1,i}, v_{1,i}, u_{2,i}, v_{2,i}, 1)$.

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 P_i. Hence the adversary knows the static secret key of P_i. The adversary treats P_i as owner and generates the tuple of values given by $\langle u_{1,i}, v_{1,i}, d_{2,i}, b_i, e_i, {h_i}^{s_2}, \{T_\tau^{(i)}\}_{\tau=1}^{l_i}, \eta_i = w_i + d_{1,i} \cdot H_2\left(\{T_\tau^{(i)}\}_{\tau=1}^{l_i}, W_i, M^{(i)}, \rho^{(i)}\right), W_i, X_i, M^{(i)}, \rho^{(i)}\rangle$ and passes it to the challenger and asks the challenger to complete the session with P_j as the peer.

Case 1a: If $\tau_j = 0$, the challenger knows the secret key and hence executes the actual protocol and delivers the session key to the adversary.

Case 1b: If $\tau_j = 1$, the challenger does not know the secret key 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 $\langle u_{1,i}, v_{1,i}, d_{2,i}, b_i, e_i, {h_i}^{s_2}, \{T_\tau^{(i)}\}_{\tau=1}^{l_i}, \eta_i, W_i, M^{(i)}, \rho^{(i)}\rangle$.
2. The challenger generates the parameters for the party P_j in the form of a similar tuple of values given by $\langle u_{1,j} = g^{x_j}, v_{1,j} = {h_j}^{x_j}, d_{2,j} = x_j + r_j \cdot b_j + s_2 \cdot e_j, b_j, e_j, \widetilde{c}_j, \widetilde{b}_j, \widetilde{e}_j, {h_j}^{s_2}, \{T_\tau^{(j)}\}_{\tau=1}^{l_j}, w_{j'} + x_j \cdot f_j, g^{w_{j'}} \cdot {y_1}^{-c_j \cdot f_j}, M^{(j)}, \rho^{(j)}\rangle$, where $r_j, x_j, w'_j, f_j \in_R \mathbb{Z}_p^*$, $\vec{\sigma}^{(j)} = (t_j, {\sigma_2}^{(j)}, \ldots, \sigma_{n_j}^{(j)}) \in_R \left(\mathbb{Z}_p^*\right)^{n_i}$. It computes $h_j = H_1(\vec{\mathbb{S}}_j)$, $b_j = H_2\left(u_{1,j}, v_{1,j}, g^{r_j}, h_j^{r_j}, 0\right)$, $e_j = H_2(u_{1,j}, v_{1,j}, g^{r_j}, {h_j}^{r_j}, 1)$, $\widetilde{b}_j = H_2\left(g^{t_j}, u_{1,j}, v_{1,j}, g^{r_j}, h_j^{r_j}, 0\right), \widetilde{e}_j = H_2\left(g^{t_j}, u_{1,j}, v_{1,j}, g^{r_j}, {h_j}^{r_j}, 1\right)$ and $\{T_\tau^{(i)}\}_{\tau=1}^{l_i}$ is computed as per the protocol specification.

3. If H_2 was already queried with inputs $\left(\{T_\tau^{(j)}\}_{\tau=1}^{l_j}, g^{w_j'} \cdot {y_1}^{-c_j \cdot f_j}, M^{(j)}, \rho^{(j)}\right)$, generate a fresh w_j' and recompute the last but two components. With very high probability, the new $\left(\{T_\tau^{(j)}\}_{\tau=1}^{l_j}, g^{w_j'} \cdot {y_1}^{-c_j \cdot f_j}, M^{(j)}, \rho^{(j)}\right)$ will not result in a previously queried input set to H_2. Set $H_2\left(\{T_\tau^{(j)}\}_{\tau=1}^{l_j}, g^{w_j'} \cdot {y_1}^{-c_j \cdot f_j}, M^{(j)}, \rho^{(j)}\right)$ as f_j.
4. The parameters generated by the challenger will satisfy ***Check 2*** in Step 2 of Key Agreement. This is because the parameters $\langle u_{1,j}, v_{1,j}, d_{2,j}, b_j, e_j, {h_j}^{s_2}\rangle$ are generated in the same way as the original scheme.
5. The parameters generated by the challenger will satisfy ***Check 3*** in the Step 2 of Key Agreement of Section 3. In fact the expression

$$\frac{g^{w_j'+x_j \cdot f_j}}{(g^{x_j})^{H_2(\{T_\tau^{(j)}\}_{\tau=1}^{l_j}, g^{w_j'} \cdot {y_1}^{-c_j \cdot f_j}, M^{(j)}, \rho^{(j)})} \cdot (y_1)^{c_j \cdot H_2(\{T_\tau^{(j)}\}_{\tau=1}^{l_j}, g^{w_j'} \cdot {y_1}^{-c_j \cdot f_j}, M^{(j)}, \rho^{(j)})}}$$

   is indeed equal to $g^{w_j'} \cdot {y_1}^{-c_j \cdot f_j} = g^{w_j}$
6. Thus the parameters generated by the challenger are consistent with that of the system.
7. The challenger sends the parameters to the adversary.
8. The challenger computes $\overline{Z}_1 = \left(g^{x_i} \cdot {y_1}^{c_i} \cdot g^{t_i}\right)^{x_j + t_j}$ where $c_i = H_2(u_{1,i})$. It also computes $P_1 = \left(u_{1,i} \cdot {y_1}^{c_i} \cdot g^{t_i}\right)^{c_j}$ and $P_2 = y_1$ where $c_j = H_2(u_{1,j})$. Note that the challenger can compute the value g^{t_i} only if it's attributes satisfy the access structure of party P_j.
9. The challenger computes $Z_2 = v_{1,i} \cdot v_{1,j}$ and $Z_3 = \left(g^{t_i}\right)^{t_j}$.
10. 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 Z such that $DH\left(P_2, P_1, Z_1/\overline{Z}_1\right)$ (valid since $P_2 = y_1$) and $H_2(Z_1, Z_2, Z_3) = Z$, where $Z_2 = v_{1,i} \cdot v_{1,j}$ and $Z_3 = \left(g^{t_i}\right)^{t_j}$.
    - If a Z exists, the challenger returns Z as the shared secret key.
    - Otherwise the challenger chooses $Z \in_R \mathbb{Z}_p^*$ and for any further query of the form (Z_1, Z_2, Z_3) to the H_2 *Oracle*, if $DH\left(P_2, P_1, Z_1/\overline{Z}_1\right)$, $Z_2 = v_{1,i} \cdot v_{1,j}$ and $Z_3 = \left(g^{t_i}\right)^{t_j}$, the challenger returns Z as the result to the query.

Finally the challenger returns Z as the shared secret key.

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

Case 2a: The case where $\tau_i = 0$ and $\tau_j = 0$. In this case, the challenger can simulate the computations of both the parties since the challenger knows the private key of the owner P_i and the peer P_j.

Case 2b: The case where either $\tau_i = 0$ or $\tau_j = 0$. Without loss of generality let us consider that $\tau_i = 0$ and $\tau_j = 1$. Here the challenger knows the secret key of i but does not know the secret key of P_j. Hence for P_i the challenger will generate the session secret key as per the algorithm. For P_j the challenger simulates similar to ***Case 1b***

Case 2c: The case where $\tau_i = 1$ and $\tau_j = 1$. In this case the challenger does not know the secret key of both P_i and P_j. Hence the challenger has to simulate the session values for both P_i and P_j, which is done identically to ***Case 1b***.

Test Session: The adversary impersonates as user P_i and sends the parameters as the following tuple of values $\langle u_{1,i}, v_{1,i}, d_{2,i}, b_i, e_i, \widetilde{c}_i, \widetilde{b}_i, \widetilde{e}_i, {h_i}^{s_2}, \{T_\tau^{(i)}\}_{\tau=1}^{l_i}, \eta_i = w_i + d_{1,i} \cdot H_2(\{T_\tau^{(i)}\}_{\tau=1}^{l_i}, W_i, M^{(i)}, \rho^{(i)}), W_i, M^{(i)}, \rho^{(i)}\rangle$ 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 P_i and P_j, where adversary impersonates as P_i and challenger has to generate parameters for user P_j. If $\tau_i = 0$, it aborts. Else it does the following:

- The challenger now passes on to the adversary the parameters as being the following tuple of values:
  $\langle u_{1,j} = {g^{x_j}}, v_{1,j} = {h_j}^{x_j}, d_{2,j} = x_j + r_j \cdot b_j + s_2 \cdot e_j, b_j, e_j, {h_j}^{s_2}, \{T_\tau^{(j)}\}_{\tau=1}^{l_j}, w_j + d_{1,j} \cdot H_2(\{T_\tau^{(j)}\}_{\tau=1}^{l_j}, g^{w_j}, M^{(j)}, \rho^{(j)}), M^{(j)}, \rho^{(j)}\rangle$,
  where $T_\tau^{(j)} = (D \cdot g^{-d_{1,j}})^{M_{\tau 1}^{(j)}} \cdot \prod_{\varsigma=2}^{n_j} g^{\sigma_\varsigma^{(j)} M_{\tau\varsigma}^{(j)}}$, here $\vec{M}_\tau^{(j)} = (M_{\tau 1}^{(j)}, M_{\tau 2}^{(j)}, \ldots, M_{\tau n_j}^{(j)})$ is τth row of the matrix $M^{(j)}$. Note that $t_j = b - d_{1,j}$ is implicitly defined, and $d_{1,j}$ is the private key component associated with user P_j which is known to the challenger, and $\{T_\tau^{(i)}\}_{\tau=2}^{l_i} \in_R \mathbb{G}$, $r_j, x_j \in_R \mathbb{Z}_p^*$, $w_j, \vec{\sigma}^{(j)} \in_R (\mathbb{Z}_p^*)^{n_j}$, $h_j = H_1(\vec{\mathbb{S}}_j)$, $b_j = H_2(u_{1,j}, v_{1,j}, g^{r_j}, {h_j}^{r_j}, 0)$, $e_j = H_2(u_{1,j}, v_{1,j}, g^{r_j}, {h_j}^{r_j}, 1)$. The parameters passed satisfy the checks as they are generated in the way similar to the scheme and $g^{t_j} = g^{b-d_{1,j}} = D \cdot g^{-d_{1,j}}$.
- The challenger performs the checks specified in *Step* 2 of the **Key Agreement** algorithm described in Section 3 on $\langle u_{1,i}, v_{1,i}, d_{2,i}, b_i, e_i, {h_i}^{s_2}, \{T_\tau^{(i)}\}_{\tau=1}^{l_i}, \eta_i, W_i, X_i, M^{(i)}, \rho^{(i)}\rangle$. If the checks pass, the challenger proceeds to next step. Else, it aborts.
- The challenger returns a $Z \in_R \mathbb{Z}_p^*$ 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_2 *Oracle* with a valid tuple (Z_1, Z_2, Z_3). Thus the challenger computes $\overline{Z}_2 = (Z_2/v_{1,j})^{{k_i}^{-1}}$ and $\overline{Z}_3 = Z_3 \cdot \left(g^{t_i}\right)^{d_{1,j}}$. The challenger also computes $S = \left(Z_1/\overline{Z}_2 \cdot \overline{Z}_3\right)^{{c_i}^{-1}}$ where $c_i = H_2(u_{1,i})$.
- Finally the challenger can return the solution for the CDH hard problem as shown in the lemma below.

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

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

- $S = \left(g^{(d_{1,i}+t_i)(d_{1,j}+b-d_{1,j})}/\overline{Z}_2 \cdot \overline{Z}_3\right)^{{c_i}^{-1}}$. Since, $\tau_i = 1$,

$$\begin{aligned}
\overline{Z}_2 &= (Z_2/v_{1,j})^{(k_i)^{-1}} \\
&= (v_{1,i} \cdot v_{1,j}/v_{1,j})^{(k_i)^{-1}} = ({h_i}^{x_i})^{(k_i)^{-1}} \\
&= \left(g^{b \cdot k_i}\right)^{x_i \cdot (k_i)^{-1}} \\
&= g^{b \cdot x_i}.
\end{aligned}$$

  (**Note:** The component $h_i = \left(g^b\right)^{k_i}$ as $\tau_i = 1$.).

- $\overline{Z}_3 = Z_3 \cdot \left(g^{t_i}\right)^{d_{1,j}} = \left(g^{t_i}\right)^{\left(b-d_{1,j}\right)} \cdot \left(g^{t_i}\right)^{d_{1,j}} = g^{b \cdot t_i}$.
- Therefore $S = \left(g^{(x_i + a \cdot c_i + t_i)\left(d_{1,j}+b-d_{1,j}\right)} / g^{b \cdot x_i} \cdot g^{b \cdot t_i}\right)^{c_i{}^{-1}} = g^{ab}$.

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

C.1 Probability Analysis

In this section we present the probability analysis of our scheme presented in Section 3.

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 (Z_1, Z_2, Z_3) from the list L_{h2} 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 $P\left(\tau_i = 0\right) = \alpha$. 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 $H_1\left(\vec{\mathbb{S}}_i\right)$ resulted in $\tau_i = 0$ 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 $H_1\left(\vec{\mathbb{S}}_i\right)$ resulted in $\tau_i = 1$ and hence an abort in the Party Corruption phase. Therefore private keys cannot be computed for attributes in $\overline{B}$. There are $\alpha.q_E$ attributes in $\overline{A}$ and remaining $(1-\alpha).q_E$ attributes in $\overline{B}$.

- $Pr\left(S_1\right) = Pr\left(\vec{\mathbb{S}}_i \in \overline{A}\right)$ for all the q_E queries. This is equal to $\left(\dfrac{\alpha \cdot q_E}{q_E}\right)^{q_E} = \alpha^{q_E}$.
- $Pr\left(S_2\right) = Pr\left(\vec{\mathbb{S}}_i \in \overline{B}\right)$, where $\vec{\mathbb{S}}_i$ is the attribute vector of the party P_i that the adversary impersonates in the Test Session. Therefore $\tau_i = 1$ in this case and hence $h_i = \left(g^b\right)^{k_i}$. This is needed to solve the SDH problem. The probability is equal to $\dfrac{(1-\alpha).q_E}{q_E} = 1 - \alpha$.
- $Pr\left(S_3\right) = Pr\left(\vec{\mathbb{S}}_j \in \overline{A}\right)$, $\vec{\mathbb{S}}_j$ is the attribute vector of the party P_j the challenger emulates in the Test Session. This ensures that the private key of P_j is computable by the challenger. This is equal to α.

- $Pr\left(S_4\right) = Pr\left(a\ valid\ \langle Z_1, Z_2, Z_3\rangle \in\ L_{H_2}\ is\ chosen\ by\ the\ challenger\right) = \frac{1}{h_5}$, where h_5 is the number of queries made of the form $\langle Z_1, Z_2, Z_3\rangle$ to the H_2 *Oracle*.

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

  $$\epsilon' = \epsilon.\frac{1}{h_5}.\alpha^{q_E+1}.(1-\alpha).$$

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

  Therefore $\epsilon' = \epsilon \cdot \frac{1}{h_5}\left(1-\frac{1}{q_E+2}\right)^{q_E+1}.\left(\frac{1}{q_E+2}\right)$.