AdHocSign: an Ad Hoc Group Signature Scheme for Accountable and Anonymous Access to Outsourced Data

Wensheng Zhang and Chuang Wang
Iowa State University
{wzhang,chuangw}@iastate.edu

Abstract. This paper presents a new group signature scheme (named AdHocSign) for dynamically formed groups, to support accountable and anonymous access to outsourced data. Specifically, assume each user is assigned a certain set of attributes and the secrets associated with these attributes when he/she joins the system; the access to each piece of outsourced data is regulated by an access structure (i.e., a logical expression of attributes) which defines an adhoc group of users allowed to access the data. The proposed AdHocSign enables a user who is allowed to access a piece of data to authenticate himself/herself to the host of the data using its preloaded secrets and some auxiliary information provided by the host; no extra secrets are needed to be distributed to the users when new access structures are constructed or existing access structures are modified. Rigorous security analysis of the AdHocSign scheme has been conducted to prove its selfless-anonymity and traceability, based on the hardness assumption of the q-SDH and the Decisional Linear problems. The scheme has also been implemented and the evaluation results show that its computational cost is comparable to that of a state-of-the-art group signature scheme.

1 Introduction

Motivations With wider and wider application of Internet, especially the increasing adoption of the cloud computing paradigm, storing sensitive user data to un-trusted, remote hosts on Internet has been popular. The outsourced sensitive data are often stored in an encrypted form to protect the confidentiality from their hosts. To realize fine-grained control of accesses to the encrypted data, attribute-based encryption schemes (ABE) [2,13,5] have been proposed. Particularly, with the ciphertext-policy ABE scheme (CP-ABE) [5], each user owns a set of attributes and a set of secrets derived from the attributes. When a piece of encrypted data is outsourced to a host, the data is labeled with an access structure to specify who are allowed to access the data; the access structure is expressed in attributes and logical operations over the attributes. A user is allowed to access certain data if and only if the attributes owned by the user satisfy the access structure of the data. The following example described in [5] demonstrates an application of the ABE schemes: Each FBI staff is assigned a set of attributes based on his/her ranks, compartments, office locations, etc. Suppose the FBI public corruption offices in Knoxville and San Francisco are investigating an allegation of bribery involving a San Francisco lobbyist and a Tennessee congressman. The memos related to this case are associated with the following access structure: “Public corruption office” AND (“Knoxville” OR “San Francisco”) OR (“Rank-level” higher than 5). Thus, with ABE, only FBI personnel who are in the Knoxville or San Francisco branch of the Public corruption office, or who have rank-level higher than five, can access these memos.

In addition to fine-grained access control, the ABE schemes also provide user anonymity as well as system dynamism and scalability. Specifically, a user can access the data that he/she is authorized to access without exposing his/her identity to the data host; also, when a new access structure is defined for certain encrypted data, there is no need to distribute new attributes or secrets to users or change the attributes or secrets owned already by users.
The ABE schemes, however, do not provide accountability, which is necessary to trace out misbehaving users and stop them from abusing the privacy preservation features. For instance, within the context of the above example, if some confidential information about the case is disclosed, it is needed to find out who have accessed the information in order to identify the person who violates the policy.

To provide user accountability while enforcing access control and protecting user anonymity, the existing group signature schemes [3,8,12] may be applied in some simple scenarios. Particularly, if an access structure is defined as only one attribute, i.e., who are allowed to access a piece of data can always be specified as the set of users who own a certain attribute, the group signature schemes can be applied as follows: For each attribute, all users owning the attribute form a group. A group public key can be computed for the group by a trusted authority. When a user is given this attribute, the trusted authority computes a unique private key and gives it to the user. For a host hosting some data that can only be accessed by users owning this attribute, it is given the public key of the group. Based on the above key distribution, each user owning this attribute can authenticate himself/herself to the data host without exposing his/her identity. When needed, the authority is able to reveal the identify of authenticated user according to the messages (i.e., signature) sent by the user during the authentication process.

If the access structure of the data is complex, however, applying the group signature schemes may be inefficient or infeasible. Suppose the access structure for certain data is $a_1 \land a_2 \land (a_3 \lor a_4)$, meaning only users who own attributes $a_1$, $a_2$ and either $a_3$ or $a_4$, are allowed to access the data. To apply the group signature scheme, an ad hoc group may have to be formed to include all the users who are allowed to access the data according to the above access structure. The trusted authority needs to be contacted to compute the public key for this new group, and the authority needs to communicate with each of the users in the new group to distribute the private keys. These operations may incur high communication overhead in terms of bandwidth consumption and delay, especially when the group size is large, the group members are distributed largely, or the group members are intermittently connected to the network.

**Contributions** To address the above problem, we propose a new signature scheme, named AdHocSign, for ad hoc groups that are defined dynamically according to access structures. The key ideas are as follows.

- Instead of distributing group private keys to the members of an ad hoc group when the group is created (i.e., when a new access structure is defined), the AdHocSign scheme pre-loads some key materials to individual users when they join the system and are given attributes.

- When certain data is posted to a host, the host is given certain auxiliary information that is computed by a trusted authority according to the access structure of the data.

- When a user needs to access a piece of data, it contacts the host of the data to obtains the access structure of the data and the afore-mentioned auxiliary information pre-loaded to the host by the authority. If the user’s attributes satisfy the access structure (i.e., the user is a member of the adhoc group defined by the access structure), the user can compute his/her own private key for the ad hoc group based on the auxiliary information and pre-loaded key materials, and authenticates himself/herself to the host. The user does not expose his/her identity or ownership of attributes during the authentication.
The design leverages the *tradeoff* between communication and storage overheads. Particularly, the design aims to avoid the communication overhead that may be introduced by the dynamic grouping at the cost of increased storage overhead paid by the users for storing key materials since they join the system. Such a tradeoff should be economically beneficial as the storage capacity has been typically cheaper than communication bandwidth in current days.

We first propose a version of AdHocSign scheme that is applicable to the scenarios where all access structures can be expressed as conjunction-only logical expressions of attributes. Then, it is extended to an advanced version that is applicable to the scenarios where access structures are general (i.e., each access structure can be expressed as a conjunction of disjunctive logical expressions of attributes). The designs are based on the assumptions that the \(q\)-Strong Diffie-Hellman (\(q\)-SDH) problem and the Decision Linear problem are hard in a multiplicative cyclic group \(G_1\). Rigorous security analysis has been conducted to show that, if the \(q\)-SDH and the Decision Linear problem are hard, the scheme is both selflessly-anonymous and traceable. We have also implemented the AdHocSign scheme, and evaluated and analyzed its computational and storage overhead. The results show that, the computational cost of the scheme is comparable to the group signature scheme proposed by Boneh and Shacham [8]. A tradeoff between system scalability and user’s storage overhead exists in the AdHocSign scheme for general access structures; that is, to support more dynamically-constructed access structures, more key materials should be preloaded to users and thus higher storage overhead will be introduced.

**Organization** In the rest of this paper, background and formal overview of the AdHocSign scheme are presented in Section 2. Sections 3 and 4 elaborate the design and analysis of the AdHocSign scheme for conjunction-only access structures and for general access structures, respectively. System implementation and overhead evaluation are reported in Section 5. Section 6 summarizes related work, and finally Section 7 concludes the paper.

## 2 Preliminaries

### 2.1 System Model

We consider a distributed system composed of one or multiple data storage sites (called data *hosts*), multiple users, and an authority trusted by all the users and hosts. To facilitate access control to the data stored at the hosts, a set of attributes are defined. Each user \(e\) has a unique identity number denoted as \(x_e \in Z_p\), where \(Z_p\) is a field of integers modus a large prime number \(p\), and is assigned a set of attributes denoted as \(\{a_{e,1}, \cdots, a_{e,n_e}\}\).

When a piece of data is posted to a host, who are allowed to access the data is specified as a logical expression containing attributes and conjunction (\(\land\)) or disjunction (\(\lor\)) operators on the attributes. We call the logical expression an *access structure*. Generally, an access structure \(T\) can be defined as

\[
T = DT_1 \land \cdots \land DT_s, \tag{1}
\]

where

\[
DT_i = a_{T,i,1} \lor \cdots \lor a_{T,i,s_i} \tag{2}
\]

for each \(i = 1, \cdots, s\). For example, \(T = a_1 \land (a_2 \lor a_3) \lor (a_4 \lor a_5 \lor a_6)\) is an access structure defined based on attributes \(a_1, \cdots, a_6\).
As an access structure can be defined on the fly when a piece of data is posted, the group of users allowed to access the data may not have been predefined; hence, we call such a group an ad hoc group and our proposed scheme is to provide a mechanism for members of such an ad hoc group to sign messages in an anonymous and accountable manner. To protect the confidentiality of the data, we assume a certain encryption scheme, for example, the CP-ABE scheme [5], is used to encrypt the data to prevent the host and unauthorized users from decrypting the data.

To summarize, when a user in our system wants to access a piece of data, he/she first authenticates himself/herself to the host of the data using our proposed ad hoc group signature (AdHocSign) scheme. After the authentication succeeds, the data, in the encrypted form, is returned to the user, who can decrypt the data using the ABE schemes. To facilitate the authentication and data decryption, a user is preloaded with some information when she is assigned attributes when he/she joins the system.

2.2 Bilinear Pairing

Let $G_1$ be a multiplicative cyclic group of prime order $p$. Let $g$ be a generator of $G_1$ and $E$ be a bilinear map defined as $E : G_1 \times G_1 \rightarrow G_2$. The bilinear map $E$ has the following properties:

- **Bilinearity**: $\forall g_1, g_2 \in G_1$ and $a, b \in \mathbb{Z}_p$, it holds that $E(g_1^a, g_2^b) = E(g_1, g_2)^{ab}$.
- **Non-degeneracy**: $E(g, g) \neq 1$.

We also assume that $G_1$ is a bilinear group. That is, the group operation in $G_1$ and the bilinear map $E : G_1 \times G_1 \rightarrow G_2$ are both efficiently computable.

Note: in a more general-case scenario, a bilinear map can be defined as $E : G_1 \times G'_1 \rightarrow G_2$, where $G'$ is also a multiplicative cyclic group. To simplify the presentation, this paper assumes $G_1 = G'_1$ though our proposed scheme can be extended to the more general-case scenario.

2.3 Group Signature Scheme by Boneh and Shacham

Our proposed AdHocSign scheme is designed based on Boneh and Shacham’s verifier-local group signature scheme [8], which includes the following primitives:

- **BSG.KeyGen($n$)**. The primitive outputs a group public key $gpk$, an $n$-element vector of user private keys $gsk = (gsk[1], \ldots, gsk[n])$, and an $n$-element vector of user revocation tokens $grt = (grt[1], \ldots, grt[n])$.
- **BSG.Sign($gpk, gsk[i], M$)**. The primitive outputs a signature $\sigma$ of message $M$ for user $i$ who owns private key $gsk[i]$.
- **BSG.Verify($gpk, RL, \sigma, M$)**. The primitive verifies whether $\sigma$ is a valid signature of message $M$ signed by any user whose membership has not been revoked (i.e., the revocation key of the user is not in the revocation list $RL$).
- **BSG.Trace($M, \sigma, grt$)**. Given a message $M$, a signature $\sigma$ of the message and the vector of revocation tokens $grt$, the primitive outputs the index of the user who generated the signature.

Hash functions $H_0$ and $H$ are used in the scheme.

Also, due to the correctness of the scheme, the following property holds:

$$BSG\_Verify(gpk, RL, BSG\_Sign(gpk, gsk[i], M), M) \iff grt[i] \neq RL.$$ (3)
2.4 AdHocSign: Definition and Security Model

**Primitives** The AdHocSign scheme provides the following primitives:

- **Setup**. The primitive chooses system parameters.
- **AttributeInit**. The primitive chooses and outputs the secrets $S[a]$ for attribute $a$.
- **UserInit**(e,$A[e],S$). This primitive is called when a user joins the system. The primitive takes as inputs an user ID $e$, the set of attributes $A[e]$ owned by the user, and the set of secrets $S$ for all attributes. It outputs the private key $gsk_e$ of the user.
- **AccessStructureInit**(T,$S$). This primitive is called when a new access structure is defined. It takes as inputs an access structure $T$ and the set of secretes $S$ for all attributes. It outputs the public key $gpk_T$ regarding the access structure.
- **Sign**(gpk$_T$,gsk$_e$,M). This primitive takes as inputs the public key $gpk_T$ regarding a certain access structure $T$, a private key $gsk_e$ for a certain user $e$, and a message $M$. It outputs signature $\sigma_{M,T}$ on $M$ regarding $T$ for user $e$, or NULL if the attributes owned by the user do not satisfy the access structure.
- **Revvoke**(gpk$_T$,gsk$_e$). The primitive is called to get the revocation token of user $e$ regarding access structure $T$. It takes as inputs the public key $gpk_T$ regarding $T$ and the private key $gsk_e$ of $e$, and outputs the revocation token $grt_T[e]$.
- **Verf**(gpk$_T$,RL,$\sigma$,M). The primitive takes as inputs the public key $gpk_T$ regarding $T$, list $RL$ of revocation tokens, a signature $\sigma$ and message $M$. It outputs valid if and only if $\sigma$ is a valid signature of $M$ regarding $T$ that was generated by a user whose revocation token is not in $RL$.
- **Trace**(M,$\sigma$,grt$_T$). This primitive takes as inputs message $M$, signature $\sigma$ and the set of revocation tokens $grt_T$ regarding $T$. It outputs the ID of the user who generated the signature.

**Security Assumptions** The AdHocSign scheme is designed based on the assumptions about the hardness of the following problems.

- **q-Strong Diffie-Hellman (q-SDH) Problem in $\mathbb{G}_1$**: Given a $(q+1)$-tuple $g, g^x, g^{x^2}, \cdots, g^{x^q}$ of group $\mathbb{G}_1$ as input, output a pair $(c, g^{1/(c+1)})$ where $c \in \mathbb{Z}_p$. We say that the $(q,t,\epsilon)$-SDH assumption holds in $\mathbb{G}_1$ if no $t$-time algorithm has the probability of at least $\epsilon$ in solving the $q$-SDH problem in $\mathbb{G}_1$.

- **Decision Linear Problem in $\mathbb{G}_1$**: Given $u, v, h, u^a, v^b, h^c \in \mathbb{G}_1$ as input, output yes if $a+b = c$ and no otherwise. We say that the $(t,\epsilon)$-Decision Linear assumption holds in $\mathbb{G}_1$ if no $t$-time algorithm has probability of at least $\epsilon + 1/2$ in solving the Decision Linear problem in $\mathbb{G}_1$.

**Security model** The AdHocSign scheme should satisfy three requirements: correctness, traceability, and selfless-anonymity, which are formally defined as follows.

**Correctness** Given an access structure $T$, every signature generated by a user whose attributes satisfy the access structure must be verified as valid, except when the user is revoked. That is,

\[
[\sigma = \text{Sign}(gpk_T, gsk_e, M) \neq \text{NULL}] \implies \text{Verf}(gpk_T, RL, \sigma, M) = \text{valid} \iff grt_T[e] \not\in RL.
\]
**Traceability** The AdHocSign scheme is traceable if no adversary can win the traceability game defined below. In the traceability game, the adversary attempts to forge a signature that cannot trace to any of the users in his coalition using the Trace algorithm. Particularly, the traceability game between a challenger and an adversary \( \mathcal{A} \) is defined as follows.

- **Initialization.** The challenger runs algorithm \( \text{Setup} \) to get system parameters, and provides the public parameters to \( \mathcal{A} \). He also initializes the coalition of \( \mathcal{A} \), denoted as \( U \), to \( \emptyset \).

- \( \mathcal{A} \) can make following types of query to the challenger.
  - **Corruption.** \( \mathcal{A} \) requests the private key \( gsk_e \) of some user \( e \) who owns a certain set of attributes \( A[e] \). The challenger appends \( e \) to \( U \), and responds with \( gsk_e \).
  - **Access Structure Initialization.** \( \mathcal{A} \) requests the public key \( gpk_T \) for a certain access structure \( T \). The challenger responds with \( gpk_T \), which is computed by using the \( \text{AccessStructureInit} \) algorithm.
  - **Signing.** \( \mathcal{A} \) requests a signature on message \( M \) regarding access structure \( T \) for user \( e \). The challenger computes and replies \( \sigma \leftarrow \text{Sign}(gpk_T, gsk_e, M) \).

- **Response.** Finally, \( \mathcal{A} \) outputs \( M^* \), a set of RL* of revocation tokens, and a signature \( \sigma^* \). \( \mathcal{A} \) wins if: (i) \( \sigma^* \) is accepted by the verification algorithm as a valid signature on \( M^* \) regarding \( T \); (ii) \( \sigma^* \) traces to a user outside of the coalition \( U \setminus RL^* \), or the tracing algorithm fails; and (iii) \( \sigma^* \) is nontrivial, i.e., \( \mathcal{A} \) did not obtain \( \sigma^* \) by making a signing query at \( M^* \) regarding \( T' \), where every set of attributes satisfying \( T' \) also satisfies \( T \).

Algorithm \( \mathcal{A} (t, q_H, q_S, n, m, \epsilon) \)-breaks traceability in an \( n \)-user \( m \)-attribute AdHocSign scheme if: \( \mathcal{A} \) runs in time at most \( t \); \( \mathcal{A} \) makes at most \( q_H \) hash oracle queries on hash functions \( H_0 \) and \( H \), and at most \( q_S \) signing queries; and \( \mathcal{A} \) wins the traceability game with a probability of at least \( \epsilon \). If no algorithm can \( (t, q_H, q_S, n, m, \epsilon) \)-break traceability in an AdHocSign scheme, the scheme is \( (t, q_H, q_S, n, m, \epsilon) \)-traceable.

**Selfless-anonymity** The AdHocSign scheme is selflessly-anonymous if no adversary can win the selfless-anonymity game defined below. In the game, the adversary attempts to determine which of two users generated a signature when it is not given access to the secrets held by these two users. Particularly, the game is defined as follows.

- **Initialization.** The challenger runs algorithm \( \text{Setup} \) to get system parameters and provides the public parameters to \( \mathcal{A} \).

- **Queries.** As in the traceability game, \( \mathcal{A} \) can make corruption, access structure initialization and signing queries to the challenger, who responds as in the traceability game.

- **Challenge.** \( \mathcal{A} \) outputs a message \( M \), an access structure \( T \), and two user IDs \( i_0 \) and \( i_1 \). It must have not issued a corruption or revocation query at either user. The challenger chooses a bit \( b \) from \( \{0, 1\} \) uniformly at random, computes a signature on \( \sigma^* \leftarrow \text{Sign}(gpk_T, gsk_b, M) \), and provides \( \sigma^* \) to \( \mathcal{A} \).

- **Restricted Queries.** After obtaining the challenge, \( \mathcal{A} \) is allowed to make additional corruption, host preparation and signing queries with the restriction that the Corruption query cannot be made at users \( i_0 \) and \( i_1 \).

- **Response.** Finally, \( \mathcal{A} \) outputs a bit \( b' \), his guess of \( b \). The adversary wins if \( b' = b \).

Letting \( \mathcal{A} \)'s advantage in winning the game be \( \text{AdvSA}_A = |Pr[b = b'] - \frac{1}{2}| \), \( \mathcal{A} (t, q_H, q_S, n, m, \epsilon) \)-breaks selfless-anonymity in an \( n \)-user \( m \)-attribute AdHocSign scheme if: \( \mathcal{A} \) runs in time at most...
$t$; $\mathcal{A}$ makes at most $q_H$ queries to hash functions $H_0$ and $H$, and at most $q_S$ signing queries; and $AdvSA_A$ is at least $\epsilon$. If no algorithm can $(t, q_H, q_S, n, m, \epsilon)$-break selfless-anonymity in an AdHocSign scheme, the scheme is $(t, q_H, q_S, n, m, \epsilon)$-selflessly-anonymous.

3 Construction for Conjunction-only Access Structures

In this section, we present the design of AdHocSign when the access structure is a conjunction-only logical expression of attributes. Specifically, the general format of a conjunction-only access structure is as follows:

\[ T = a_{T,1} \land \cdots \land a_{T,s}, \tag{4}\]

where each $a_{T,i}$ ($i = 1, \cdots, s$) is an attribute.

3.1 Algorithms

The following algorithms implement the primitives defined in Section 2.4.

**CO_Setup.** The setup algorithm chooses the following system parameters:

- $\mathbb{Z}_p$: a finite field of integers where $p$ is a large prime number,
- $\mathbb{G}_1$: a multiplicative cyclic and bilinear group of prime order $p$,
- bilinear map $E: \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$,
- $g$: a generator of group $\mathbb{G}_1$, and
- a secret element $\gamma \in \mathbb{Z}_p$.

**CO_AttributeInit($a$).** The attribute initialization algorithm takes as input an attribute $a$, and outputs a secret number denoted as $\alpha_a$, where $\alpha_a$ is picked from $\mathbb{Z}_p$ uniformly at random. $\alpha_a$ is also stored at vector $\mathbb{S}$ at index $a$; i.e., $\mathbb{S}[a] \leftarrow \alpha_a$.

**CO_UserInit( $e$, $A[e]$, $\mathbb{S}$).** The user initialization algorithm takes the following inputs:

- $e$: the user ID;
- $A[e] = \{a_{e,i} | i = 1, \cdots, n_e\}$: the set of attributes assigned to user $e$; and
- $\mathbb{S}$: the set of secret numbers for attributes.

The output of the algorithm is a private key of the user, i.e.,

\[ gs\text{sk}_e = \langle x_e, \{(a_{e,i}, A_{e,a_{e,i}}) | i = 1, \cdots, n_e\} \rangle, \tag{5}\]

where $x_e$ is picked from $\mathbb{Z}_p$ uniformly at random and

\[ A_{e,a_{e,i}} = g^{\alpha_{a_{e,i}}} \cdot \tag{6}\]
\textbf{CO\_AccessStructureInit}(T, S). The algorithm takes as inputs the access structure \(T\) and the set of secrets \(S\) for all attributes. It picks \(r_{a_T,i} (i = 1, \ldots, s)\) from \(\mathbb{Z}_p\) uniformly at random, for each attribute \(a_{T,i}\) present in \(T\). Then, it computes and outputs the public key, denoted as \(gpk_T\), regarding \(T\). In particular,
\[
gpk_T = \langle T, \{r_{a_T,1}, \cdots, r_{a_T,s}\}, g_T, w_T \rangle, \tag{7}
\]
where
\[
g_T = g^{\sum_{k=1}^{s}(r_{a_T,k} \cdot a_{T,k})} \text{ and } w_T = g^{\gamma_T}. \tag{8}
\]

\textbf{CO\_Sign}(gpk_T, gsk_e, M). The inputs of the signature generation algorithm include
- \(gpk_T\): the public key regarding access structure \(T\) for which a signature is to be generated;
- \(gsk_e\): the private key held by user \(e\) for whom the signature is to be generated; and
- \(M\): the message to be signed.

The algorithm outputs \(\sigma_{M,T}\), signature of message \(M\) regarding \(T\). The signature is generated in the following steps:

1. If user \(e\) does not have all attributes present in \(T\), \text{NULL} is returned. Otherwise, the following steps are executed.
2. \(\hat{A}_{e,T} \leftarrow \prod_{i=1}^{s} A_{e,a_{T,i}}^{r_{a_T,i}}\);
3. \(\sigma_{M,T} \leftarrow \text{BSG\_Sign}(gpk = \{g_T, w_T\}, gsk = \{\hat{A}_{e,T}, x_e\}, M)\). Recall that \(\text{BSG\_Sign}\) is the signing primitive in the group signature scheme proposed by Boneh and Shacham. When calling the primitive, \(\{g_T, w_T\}\) is the public key and \(\{\hat{A}_{e,T}, x_e\}\) is the user private key.

\textbf{CO\_Revoke}(gpk_T, gsk_e). This algorithm takes as inputs the public key \(gpk_T\) for a certain access structure \(T\) and the private key \(gsk_e\) for a certain user \(e\). It computes the revocation token of \(e\) regarding \(T\) as
\[
grt_T[e] \leftarrow \prod_{i=1}^{s} A_{e,a_{T,i}}^{r_{a_T,i}}.
\]
Then, the algorithm returns \(grt_T[e]\).

\textbf{CO\_Verify}(gpk_T, RL_T, \sigma, M). To verify if \(\sigma\) is a signature of message \(M\) regarding access structure \(T\), the verification algorithm takes the public key \(gpk_T\) and the list \(RL_T\) of revocation keys as inputs. The algorithm can be implemented by calling
\[
\text{BSG\_Verify}(gpk = \{g_T, w_T\}, RL_T, \sigma, M).
\]
Recall that \(\text{BSG\_Verify}\) is the revocation primitive in the group signature scheme proposed by Boneh and Shacham.

\textbf{CO\_Trace}(M, \sigma, grt_T). This algorithm can be implemented by calling \(\text{BSG\_Trace}(M, \sigma, grt_T)\).
3.2 Security Analysis

Correctness

Theorem 1. The AdHocSign scheme for conjunction-only access structures is correct. Formally, if a user \( e \) has all attributes present in an access structure as defined in Equation (4), then

\[
\{ \text{CO.Verify}(gpk_T, RL_T, \text{CO.Sign}(gpk_T, gsk_e, M), M) = \text{valid} \} \iff \{ \text{grt}_T[e] \notin RL_T \}. \tag{9}
\]

Proof. Suppose user \( e \) has all attributes appearing in a conjunction-only access structure \( T \). According to algorithms \( \text{CO.Sign} \) and \( \text{CO.Verify} \), it holds that

\[
\text{CO.Sign}(gpk_T, gsk_e, M) = \text{BSG.Sign}(\{g_T, w_T\}, \{\hat{A}_{e,T}, x_e\}, M)
\]

and

\[
\text{CO.Verify}(gpk_T, RL_T, \sigma, M) = \text{BSG.Verify}(\{g_T, w_T\}, RL_T, \sigma, M),
\]

where

\[
w_T = g_T^\gamma, \quad \hat{A}_{e,T} = g_T^{x_e + x_e}, \quad \text{and} \quad \sigma = \text{CO.Sign}(gpk_T, gsk_e, M).
\]

Also, as the group signature scheme proposed by Boneh and Shacham is correct, property (3) holds and thus the following property also holds:

\[
\text{BSG.Verify}(\{g_T, w_T\}, RL_T, \sigma, M) \iff \hat{A}_{e,T} \notin RL_T.
\]

Further note that, \( \text{grt}_T[e] = \hat{A}_{e,T} \) according to algorithm \( \text{CO.Revoke} \). Therefore, property (9) holds.

Traceability Based on the definitions of traceability game and traceability in Section 2.4, the traceability of the AdHocSign scheme for conjunction-only access structure is stated in Theorem 2.

Theorem 2. If the \((q, t', \epsilon')\)-SDH assumption holds in \( \mathbb{G}_1 \), the AdHocSign scheme for conjunction-only access structures is \((t, q_H, q_S, n, m, \epsilon)\)-traceable, where \( n = q - 1 \), \( \epsilon = 8n\sqrt{\epsilon' q_H} + 2n/p \), and \( t' = \Theta(1) \cdot (t + m \cdot q) \).

Proof. In Appendix 1.

Selfless-anonymity The selfless-anonymity of our proposed scheme is stated in Theorem 3.

Theorem 3. The AdHocSign scheme for conjunction-only access structures is \((t, q_H, q_S, n, m, \epsilon)\)-selflessly-anonymous assuming the \((t', \epsilon')\) Decision Linear assumption holds in group \( \mathbb{G}_1 \) for \( \epsilon' = \frac{\epsilon}{2} \left( \frac{1}{n^2} - \frac{q_S q_H}{p} \right) \) and \( t' = \Theta(1) \cdot (t + mn) \).

Proof. In Appendix 2.
3.3 Overhead Analysis

With the AdHocSign scheme for conjunction-only access structures, a user needs to perform the following computations to sign a message regarding an access structure of form $T = a_1 \land \cdots \land a_s$:

(i) $s$ exponential operations in $G_1$, (ii) $s$ multiplication operations in $G_1$, and (iii) one invocation of $BSG\_Sign$ primitive. As shown in the results of system implementation and evaluation (Fig. 2 in Section 5), the computational cost of computing a private key (i.e., steps (i) and (ii)) is lower than that of $BSG\_Sign$ as long as $s$ is not too large. Hence, the overview cost is comparable to twice of that of $BSG\_Sign$.

Note that the computational cost of verifying a signature is the same as that of $BSG\_Verify$ primitive. Also a user in the AdHocSign scheme only needs to keep one piece of secret for each attribute owned by itself, which is the same as in Boneh and Shacham’s group signature scheme [8].

4 Construction for General Access Structures

To lay the foundation for our construction for general access structures, our construction for disjunction-only access structures is presented first, which is followed by the construction for general access structures and the security and overhead analysis of the construction.

4.1 Construction for Disjunction-only Access Structures: The Algorithms

We now describe the AdHocSign scheme for disjunction-only access structures as $T = a_{T,1} \lor \cdots \lor a_{T,s}$, where each $a_{T,i}$ ($i = 1, \cdots, s$) is an attribute.

**DO\_Setup.** The algorithm includes algorithm CO\_Setup. In addition, the algorithm also performs the following operations: (i) A secret $\xi$ is randomly chosen from $\mathbb{Z}_p$; (ii) hash functions $H_1 : \mathbb{Z}_p \cdot \mathbb{Z}_p \cdot \mathbb{Z}_p \cdot \mathbb{Z}_p \rightarrow G_1$ and $H_2 : \mathbb{Z}_p \cdot \mathbb{Z}_p \cdot \mathbb{Z}_p \rightarrow \mathbb{Z}_p$ are chosen.

**DO\_AttributeInit($a$, $N$).** The algorithm takes two inputs: $a$ which is ID of the attribute, and $N$ which is an integer. The algorithm outputs $N$ secret numbers to be associated with attribute $a$, denoted as $\alpha_{a,i}$ for $i = 1, \cdots, N$, where each $\alpha_{a,i}$ is picked from $\mathbb{Z}_p$ uniformly at random. All these secret numbers are recorded in vector $S[a]$.

**DO\_UserInit($e$, $A[e]$, $S$).** The inputs to the user initialization algorithm include the user ID $e$, the set of attributes

$$A[e] = \{a_{e,i} | i = 1, \cdots, n_e\}$$

which is owned by the user, and the set of secret keys

$$S = \{\alpha_{a_{e,i},j} | i = 1, \cdots, n_e; j = 1, \cdots, N\}$$

for the attributes.

The algorithm outputs the private key for user $e$, i.e.,

$$gske = \langle x_e, \{(a_{e,i}, A'_{e,a_{e,i},1}, \cdots, A'_{e,a_{e,i},N}) | i = 1, \cdots, n_e\} \rangle.$$
where $x_e$ is picked from $\mathbb{Z}_p$ uniformly at random,

$$A'_{e,a,e,i,j} = A_{e,a,e,i,j} \cdot H_1(e, a_{e,i,j}, H_2(\xi, a_{e,i,j}))^{-1} \quad \text{for} \quad j = 1, \ldots, N,$$

and

$$A_{e,a,e,i,j} = g^{a_{e,i,j}} \cdot x_e.$$

**DO AccessStructureInit**($T, S$). The inputs to the access structure initialization algorithm include the access structure $T$ and the set of secret keys $S$ for all attributes.

The algorithm picks $r_T$ from $\mathbb{Z}_p$ uniformly at random. For each $a_{T,i}$, the algorithm also picks $\delta_{T,i}$ from $\{1, \ldots, N\}$ such that $\alpha_{a_{T,i},\delta_{T,i}}$ is a secret associated with $a_{T,i}$ that has not been used in access structure initialization before. If such $\delta_{T,i}$ cannot be found successfully, the primitive fails. Otherwise, the algorithm computes and outputs the following public key regarding $T$:

$$gpk_T = \langle T, \tilde{R}_T, g_T = g^{r_T}, w_T = g_T^\gamma \rangle,$$

where

$$\tilde{R}_T = \{(r_{T,i} = \frac{r_T}{\alpha_{a_{T,i},\delta_{T,i}}}, \delta_{T,i}, h_{T,i} = H_2(\xi, a_{T,i}, \delta_{T,i}))\mid i = 1, \ldots, s\}. $$

**DO_Sign**($gpk_T, gsk_e, M$). The signature generation algorithm takes as inputs public key $gpk_T$ regarding $T$, private key $gsk_e$ of user $e$, and message $M$. It outputs signature $\sigma_{M,T}$ on message $M$ regarding $T$ for user $e$ as follows:

1. If user $e$ does not have any attribute appearing in $T$, NULL is returned.
2. Otherwise, assuming the user owns attribute $a_{T,i}$, the user computes

$$A_{e,a_{T,i},\delta_{T,i}} = A'_{e,a_{T,i},\delta_{T,i}} \cdot H_1(e, a_{T,i}, \delta_{T,i}, h_{T,i}) \quad \text{and} \quad \tilde{A}_e,T = A'_{e,a_{T,i},\delta_{T,i}};$$

recalling that $A'_{e,a_{T,i},\delta_{T,i}}$ and $h_{T,i}$ are included in $gpk_T$, and returns

$$\sigma_{M,T} = BSG_{\cdot}\text{Sign}(\{g_T, w_T\}, \{\tilde{A}_e,T, x_e\}, M).$$

**DO_Verify**($gpk_T, RL_T, \sigma, M$). The algorithm is the same as CO_Verify.

**Discussion:** According to the above AdhocSign scheme, for every dynamically-constructed conjunction-only access structure, one unique secret associated with every attribute occurring in the access structure is consumed (i.e., it cannot be used for other access structure any more) at every user owning the attribute.

The reason that the secret of an attribute cannot be reused for two or more access structures can be explained intuitively with the following example. Consider two access structures as follows:

$$T_1 = a \lor b,$$

and

$$T_2 = a \lor c,$$
and a user $e$ who owns attribute $b$ but not $a$ and $c$. The user is thus preloaded with $x_e$ and $A'_{e,b}$.

Suppose that, in the public key $gpk_{T_1}$ for $T_1$, parameters $r_{T_1,a}$ and $r_{T_1,b}$ are associated with attributes $a$ and $b$; in the public key $gpk_{T_2}$ for $T_2$, parameters $r_{T_2,a}$ and $r_{T_2,c}$ are associated with attributes $a$ and $c$. Assume user $e$ is only preloaded with one secret (denoted as $\alpha_0$) associated with $b$. Also, it is assumed that, for each user owning $a$, the same secret of attribute $a$ is used for deriving the private keys for $T_1$ and $T_2$.

Hence, the private key of user $e$ for $T_1$ is

$$(x_e, \hat{A}_{e,T_1} = A'_{e,b}^{r_{T_1,b}}),$$

where $A'_{e,b} = A'_{e,b} \cdot H_1(e, b, 1, h_{T_1,b})$. Furthermore, however, user $e$ is also able to derive the private key for $T_2$

$$(x_e, \hat{A}_{e,T_2} = A'_{e,b}^{r_{T_2,a}/r_{T_1,a} \cdot r_{T_2,a}})$$

though $e$ does not own attribute $a$ or $c$. This is because of the following:

- If user $e$ had owned attribute $a$ and thus been preloaded with $A'_{e,a}$, it would have held that

$$A_{e,a} = A'_{e,b}^{r_{T_1,a}/r_{T_1,a}},$$

where $A_{e,a} = A'_{e,a} \cdot H_1(e, a, 1, h_{T_2,a})$.

- Hence, it holds that

$$\hat{A}_{e,T_2} = A'_{e,a}^{r_{T_2,a}} = A'_{e,b}^{r_{T_1,a} \cdot r_{T_2,a}}.$$  

### 4.2 Construction for General Access Structures: The Algorithms

Integrating the AdHocSign scheme for disjunction-only access structures and that for conjunction-only access structures, the AdHocSign scheme for general access structures is designed as follows.

**G\_Setup, G\_AttributeInit($i$, $N$), G\_UserInit($e$, $A[e]$, $S$) and G\_Verify($gpk_T$, $RL_T$, $\sigma$, $M$).**

The algorithms are the same as DO\_Setup, DO\_AttributeInit, DO\_UserInit and CO\_Verify, respectively.

**G\_AccessStructureInit($T$, $S$).** The inputs to the access structure initialization algorithm include the access structure $T$ as defined in Equations (1) and (2), and the set of private keys $S$.

For each $DT_i = a_{T,i,1} \lor \cdots \lor a_{T,i,s}$, the algorithm picks $r_{T,i}$ from $\mathbb{Z}_p$ uniformly at random. Then, for each $a_{T,i,j}$ that is a part of $DT_i$, the algorithm finds $\delta_{T,i,j}$ from $\{1, \cdots, N\}$ such that $\alpha a_{T,i,j} \delta_{T,i,j}$ is a secret associated with attribute $a_{T,i,j}$ and has not been used in access structure initialization before, and computes $r_{T,i,j} = \alpha a_{T,i,j}^{\delta_{T,i,j}}$. If such $\delta_{T,i,j}$ for $j = 1, \cdots, s_i$ cannot be found in $\{1, \cdots, N\}$ successfully, the primitive fails. Finally, the algorithm computes and outputs the following public key:

$$gpk_T = \langle T, \{ (r_{T,i,j}, \delta_{T,i,j}, h_{T,i,j}) | i = 1, \cdots, s; \text{ for each } i, j = 1, \cdots, s_i \}, g_T = g_{\sum_{i=1}^{s} \tau_{T,i}}, w_T = g_T^{\gamma} \rangle,$$

where

$$h_{T,i,j} = H_2(\xi, a_{T,i,j}, \delta_{T,i,j}).$$

(10)
\textbf{G\_Sign}(gpk_T, gsk_e, M).} The signature generation algorithm takes as inputs public key \(gpk_T\) regarding \(T\), private key \(gsk_e\) of user \(e\), and message \(M\). It outputs signature \(\sigma_{M,T}\) on message \(M\) regarding \(T\) for user \(e\) as follows:

1. If the attributes owned by user \(e\) do not satisfy \(T\), \(\text{NULL}\) is returned.
2. Otherwise, assuming the user owns attribute \(a_{T,i,k_i}\) for \(i = 1, \ldots, s\), the user computes

\[
\hat{A}_{e,T} = \prod_{i=1}^{s} [A_{e,a_{T,i,k_i},\delta_{T,i,k_i}} \cdot H_1(x_e, a_{T,i,k_i}, \delta_{T,i,k_i}, h_{T,i,k_i})]^{r_{T,i,k_i}},
\]

and returns

\[
\sigma_{M,T} = BSG\_Sign(\{g_T, w_T\}, \{\hat{A}_{e,T}, x_e\}, M).
\]

\textbf{G\_Revoke}(gpk_T, gsk_e) This algorithm takes as inputs the public key \(gpk_T\) for a certain access structure \(T\) and the private key \(gsk_e\) for a certain user \(e\). It computes the revocation token of \(e\) regarding \(T\) as

\[
grt_T[e] ← \hat{A}_{e,T},
\]

where the computation of \(\hat{A}_{e,T}\) is the same as step 2 in primitive G\_Sign. Then, the algorithm returns \(grt_T[e]\).

\textbf{G\_Trace}(M, \sigma, grt_T).} This algorithm can be implemented by calling \(BSG\_Trace(M, \sigma, grt_T)\).

4.3 Security Analysis

Correctness

\textbf{Theorem 4.} The AdHocSign scheme for general access structures is correct. Formally, if the attributes owned by a user \(e\) satisfy an access structure \(T\) as defined in Equation (1), then

\[
\{G\_Verify(gpk_T, RL_T, G\_Sign(gpk_T, gsk_e, M), M) = \text{valid} \} ⇔ \{grt_T[e] \notin RL_T\}. \tag{11}
\]

\textbf{Proof.} Assume \(T = DT_1 \land \cdots \land DT_s\), and user \(e\) owns attribute \(a_{i,k_i}\) that appears in \(DT_i\) for \(i = 1, \ldots, s\). Hence, private key \(gsk_e\) includes the following information:

\[
x_e, \{(a_i,k_i, A_{e,a_{i,k_i},1}, \ldots, A_{e,a_{i,k_i},N}) \mid i = 1, \ldots, s\},
\]

where

\[
A'_{e,a_{i,k_i},j} = g^{\alpha_{a_{i,k_i},j}} \cdot H_1(x_e, a_{i,k_i}, j, H_2(\xi, a_{i,k_i}, j))^{-1} \text{ for } j = 1, \ldots, N.
\]

Also, from \(gpk_T\) user \(e\) can obtain the following information:

\[
\{(r_{T,i,k_i} = \frac{r_{T,i}^j}{\alpha_{a_{T,i,k_i},\delta_{T,i,k_i}}} \cdot \delta_{T,i,k_i}, h_{T,i,k_i} = H_2(\xi, a_{T,i,k_i}, \delta_{T,i,k_i}) \mid i = 1, \ldots, s\}, g_T = g^{\sum_{i=1}^{s} r_{T,i}}, w_T = g_T^7.
\]

Based on the above information in \(gsk_e\) and \(gpk_T\), user \(e\) can obtain

\[
\{A_{e,a_{T,i,k_i},\delta_{T,i,k_i}} = A'_{e,a_{T,i,k_i},\delta_{T,i,k_i}} \cdot H_1(x_e, a_{T,i,k_i}, \delta_{T,i,k_i}, h_{T,i,k_i}) = g^{\frac{\alpha_{a_{T,i,k_i},\delta_{T,i,k_i}}}{7+x_e}} \mid i = 1, \ldots, s\}.
\]
Then, it can compute
\[ \hat{A}_{e,T} = \prod_{i=1}^{s} (A_{e,aT,i,k_i} \delta_{T,i,k_i})^{r_{T,i,k_i}} = g^{\sum_{i=1}^{s} r_{T,i,k_i}}. \]

It is satisfied that
\[ E(w_T g_T^{\hat{A}_{e,T}}, \hat{A}_{e,T}) = E(g_T, g_T); \]
that is, \((\hat{A}_{e,T}, x_e)\) and \((g_T, w_T)\) form a pair of private key and public key for the group signature scheme proposed by Boneh and Shacham.

Further,
\[
G_{\text{Verify}}(g_{pkT}, RL_T, G_{\text{Sign}}(g_{pkT}, g_{sk_e}, M), M) = BSG_{\text{Verify}}(\{g_T, w_T\}, RL_T, BSG_{\text{Sign}}(\{g_T, w_T\}, \{\hat{A}_{e,T}, x_e\}, M), M).
\]

Due to the correctness of the BSG scheme (i.e., property (3)), it holds that
\[ G_{\text{Verify}}(g_{pkT}, RL_T, G_{\text{Sign}}(g_{pkT}, g_{sk_e}, M), M) \iff gr_{T}[e] \notin RL_T. \]

Also due to \(gr_{T}[e] = \hat{A}_{e,T}\), property (11) holds.

Therefore, the correctness of the scheme is proved.

Traceability

\textbf{Theorem 5.} If the AdHocSign scheme for conjunction-only access structures is \((t', q_H, q_S, n, m, \epsilon)\)-traceable, then the AdHocSign scheme for general access structures is \((t, q_H, q_S, n, m, \epsilon)\)-traceable, where \(t' = O(t \cdot m \cdot N)\) and \(N\) is the maximum number of secrets associated with each attribute.

\textit{Proof.} In Appendix 3.

\textbf{Theorem 6.} If the \((q, t', \epsilon')\)-SDH assumption holds in \(G_1\), then the AdHocSign scheme for general access structures is \((t, q_H, q_S, n, m, \epsilon)\)-traceable, where \(n = q - 1\), \(\epsilon = 8n\sqrt{q_H} + 2n/p\), \(t' = O(t \cdot m \cdot N)\), and \(N\) is the maximum number of secrets associated with each attribute.

\textit{Proof.} The theorem can be inferred from Theorems 2 and 5.

Selfless-anonymity

\textbf{Theorem 7.} The AdHocSign scheme for general access structures is \((t, q_H, q_S, n, m, \epsilon)\)-selflessly-anonymous assuming the \((t', \epsilon')\) Decision Linear assumption holds in group \(G_1\) for \(\epsilon' = \Theta(1) \cdot (t + m \cdot n \cdot N),\) and \(N\) is the maximum number of secrets associated with each attribute.

\textit{Proof.} Similar to the proof of Theorem 3.
4.4 Overhead Analysis

Computational Costs The analysis is similar to that in Section 3.3. With the AdHocSign scheme for general access structures, a user needs to perform the following computations to sign a message regarding an access structure of form $T = DT_1 \land \cdots \land DT_s$ where each $DT_i$ is an attribute or a disjunction of attributes: (i) $s$ invocations of hash functions, (ii) $s$ exponential operations in $\mathbb{G}_1$, (iii) $2s$ multiplication operations in $\mathbb{G}_1$, and (iv) one invocation of $BSG\_Sign$ primitive. As shown in the results of system implementation and evaluation (Fig. 2 in Section 5), the cost of computing a private key (i.e., (i)-(iii)) is lower than that of $BSG\_Sign$ when $s$ is not large. Besides, the computational cost of verifying a signature is the same as that of $BSG\_Verify$ primitive.

Storage Costs vs. Number of Dynamically-defined Access Structures A user needs to store $N$ secrets, each of which is an element of $\mathbb{G}_1$, for every attribute owned by it. As we can see from the AccessStructureInit primitive, one secret of an attribute is consumed for each occurrence of the attribute in access structures. The larger is $N$, the more access structures can be constructed. Hence, there is a tradeoff between the storage cost at a user and the number of access structures that can be defined dynamically. In practice, as the size of a secret is small, a user is able to store a large number of secrets and thus the number of access structures defined dynamically can be large. Furthermore, when the secrets for an attribute have been used up, the authority can distribute new secrets to users; as the distribution occurs infrequently, the communication overhead incurred can still keep low.

5 Implementation

System Implementation We have implemented a proof-of-concept system composed of a client computer, a data host server, and a trusted authority server. The client computer is a desktop computer with 1.83 GHz Genuine Intel (R) processor and 3 GB of RAM. The data host server and the trusted authority server are workstation computers with two 2.13 GHz Intel Xeon (R) processors and 24 GB of RAM. The java paring-based cryptographic (jPBC) library [1] is used to implement the operations on bilinear groups. When using the jPBC libraries, we adopt the type A curves defined at [1].

Computational Cost of BSG Primitives As the group signature scheme [8] provides primitive operations for the proposed scheme, we first measure its computational overhead at the user side and the server side respectively. The signing primitive ($BSG\_Sign$) is conducted at user side, which takes about 1.65 seconds by average. The signature verification primitive ($BSG\_Verify$) is conducted at the server side. The measurement indicates that the server spends 0.28 seconds by average to check the validity of a signature and about $0.035L$ seconds by average to check if the signature signer is one of $L$ revoked users.

Computational Cost of UserInit Primitive The trusted authority server is responsible for executing the UserInit primitive to initialize a user. The time of initializing a user is linearly increased with the number of attributes that the user has. For each attribute $a$ owned by a user $e$, the authority needs to compute $A_{e,a}$ if the AdHocSign scheme is for conjunction-only access structures. When
the AdHocSign scheme is for general access structures, each attribute is associated with \( N \) secrets \( A_{e,a} \)

\[ A_{e,a,1}, \ldots, A_{e,a,N} \].

Fig. 1 shows the average computational cost for UserInit in the AdHocSign schemes for conjunction-only access structures, disjunction-only access structures, and general access structures, respectively. Here, \( N \) is fixed at 50 while the number of attributes owned by the user changes varies from 5 to 30. Fig. 1 (b) shows the average computational cost for UserInit in the AdHocSign scheme for general access structures, where the number of attributes owned by a user is fixed at 5 while \( N \) varies from 50 to 500.

**Computational Cost of AccessStructureInit Primitive** In the AdHocSign schemes for conjunction-only access structures, disjunction-only access structures and general access structures, the time for executing the AccessStructureInit primitive is almost the same, i.e., 40ms by average. This is because the execution of the primitive in these three cases all requires 2 exponential operations in \( G_1 \) to compute \( g_T \) and \( w_T \) for structure \( T \), which are the major time-consuming operations.

**Computational Cost for Deriving a Private Key in Signing Primitive** To sign a message, a client needs to derive the private key and then invoke the \( BSG\_Sign \) primitive. Fig. 2 shows the time for deriving a private key in the AdHocSign schemes for conjunction-only access structures, disjunction-only access structures and general access structures, respectively. As we can see, the computational time spent by the scheme for disjunction-only access structure does not change, while the time spend by the schemes for other two types of access structure increases linearly as the number of attributes in the conjunction-only access structure or the number of disjunction components in the general access structure increases.

### 6 Related Work

Attribute based encryption (ABE) \([2,13,5]\) provides fine-grained access control to shared data. With ABE, data are encrypted based on their access policies, where an access policy is a logical expression over attributes. A user is able to decrypt a piece of data if and only if she owns the access attributes that satisfy the access policy. Systems such as Persona \([4]\) have been proposed
to employ ABE for access control of shared outsourced data. However, ABE does not provide accountability. Though it ensures that only authorized users can access the data, it is hard to trace who have accessed the data.

With group signature schemes [3,8,6,12,11,14] every member of a group can sign a message anonymously on behalf of the group, a verifier of the signature can determine whether the signer is a valid member of the group without knowing the identity of the signer, but a trusted authority is able to trace out the identity of the signer when necessary. Built on top of the group signature schemes, accountable and privacy-preserving access control has been proposed for various scenarios [18,19,20,10,21,15]. In these systems, users with the same set of access rights are typically organized into a group. The groups typically need to be pre-defined and are hard to be established on the fly. Different from these works, our proposed AdHocSign is a new group signature scheme that not only inherits the features of accountable and anonymous signature generation/verification, but also enables on-the-fly formation of new groups according to a newly defined access structure (policy). If the AdHocSign is used together with ABE, accountable attribute-based fine-grained access control can be realized.

Khader proposed attribute based group signature schemes [16,17], with a similar design goal as that of AdHocSign. However, a signer in the schemes should inform the verifier of the attributes he/she owns for signature verification. On the contrary, AdHocSign does not let the verifier know what attributes are owned by a signer (unless the access structure is a conjunction-only expression of attributes), and therefore provides more privacy preservation.

7 Conclusion

We have presented a new group signature scheme (named AdHocSign) for dynamically formed groups, to support accountable and anonymous access to outsourced Data. Rigorous security analysis of the scheme has been conducted to prove its selfless-anonymity and traceability based on the hardness assumption of q-SDH and Decisional Linear problems. The scheme has also been implemented and the evaluation results show that its computational cost is comparable to that of a state-of-the-art group signature scheme.

References


Appendix 1: Proof of Theorem 2

Proof. This proof is to show: if there is a \( t \)-time algorithm \( \mathcal{A} \) that wins the traceability game in the AdHocSign scheme for conjunction-only access structures (called Game 1 hereafter), a \( t' \)-time algorithm \( \mathcal{B} \) can be constructed to solve the \( q \)-SDH problem, where \( t' = \Theta(1) \cdot (t + mq) \). Similar to the proof of traceability (i.e., Theorem 6.2) in Boneh and Shacham’s group signature scheme [8], this proof also proceeds in three parts: a framework through which \( \mathcal{B} \) interacts with \( \mathcal{A} \); instantiation of the framework for different types of algorithm (\( \mathcal{A} \)); and derivation of the conclusion.

Interaction Framework \( \mathcal{A} \) and \( \mathcal{B} \) are allowed to interact with each other as follows.

- **Initialization.** \( \mathcal{B} \) is given \( \mathbb{Z}_p, \mathbb{G}_1, g \in \mathbb{G}_1 \), bilinear mapping \( E : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2 \) and a \( q \)-SDH tuple. It creates \( n \) users and \( m \) attributes. For each user \( e \), \( \mathcal{B} \) creates a tuple \((x_e, A_{e,1}, \cdots, A_{e,m})\) where \( x_e \) could be a known element of \( \mathbb{Z}_p \) or placeholder (denoted as \( * \)), and each \( A_{e,a} \) could be a known element of \( \mathbb{G}_1 \) or placeholder (denoted as \( * \)). As to be elaborated later, users and attributes are initialized in different ways for different types of breaker algorithm.
- **Hash Queries.** \( \mathcal{A} \) can query hash functions. \( \mathcal{B} \) responds with random values while maintaining consistency.
- **Corruptions (i.e., Private Key Queries).** \( A \) can determine the set of attributes \( \{ a_{e,k} | k = 1, \cdots, n_e \} \), which is a subset of the attributes created by \( B \), for a certain user \( e \), and then request \( B \) for the private key \( gsk_e \) of \( e \). If \( B \) cannot provide the requested private key (i.e., \( x_e = \star \) or \( A_{e,a_{e,k}} = \star \) for some \( k \in \{ 1, \cdots, n_e \} \)), failure is declared and algorithm \( B \) exits. Otherwise, \( B \) computes and responds with the requested \( gsk_e \).

- **Access Structure Initializations.** \( A \) can define an access structure \( T \) and then request \( B \) for the public key \( gpk_T \) regarding \( T \). \( B \) computes and responds with the requested public key according to the \( CO\text{-}AccessStructureInit \) algorithm.

- **Signature queries.** \( A \) can request \( B \) for a user \( e \)'s signature \( \sigma_{M,T} \) on message \( M \) regarding an access structure \( T \). Here, initialization of \( T \) should have been requested by \( A \) before. \( B \) responds as follows: In the first step, if \( A_{e,a} \neq \star \) for each attribute \( a \) present in \( T \), it computes \( A_T \) as in the \( CO\text{-}Sign \) algorithm and then continues with the second step; otherwise, \( NULL \) is returned. In the second step, if \( x_e \neq \star \), \( \sigma_{M,T} = \text{BSG}\_\text{Sign}( \{ g_T, w_T \}, \{ A_{e,T}, x_e \}, M) \) is computed and returned; otherwise (i.e., \( x_e = \star \)), the following is executed to compute the signature.

1. A nonce \( r \) is picked from \( \mathbb{Z}_p \) uniformly at random, and \((u, v) = H_0(gpk, M, r) \) is queried, where \( H_0 \) is a hash function.
2. \( \alpha \) is picked from \( \mathbb{Z}_p \) uniformly at random, \( T_1 \leftarrow u^\alpha \) and \( T_2 \leftarrow \tilde{A}_{e,T}g_T^\alpha \) are set, and the \( Protocol \_1 \) simulator (in [7]) is run with values \((u, v, T_1, T_2)\).
3. The simulator returns a transcript \((u, v, T_1, T_2, R_1, R_2, R_3, c, s_\alpha, s_x, s_\delta)\), from which a group signature \( \sigma_{M,T} = (r, T_1, T_2, c, s_\alpha, s_x, s_\delta) \) is derived.
4. \( B \) patches the hash oracle at \((M, r, T_1, T_2, R_1, R_2, R_3)\) to equal \( c \). If this causes a collision, \( \sigma_{M,T} \) is returned to \( A \).

- **Output.** Finally, if \( A \) succeeds, it outputs a forged signature \( \sigma_{M,T} = (r, T_1, T_2, c, s_\alpha, s_x, s_\delta) \) on message \( M \) regarding access structure \( T \). \( B \) applies the revocation algorithm, with revocation tokens \( \tilde{A}_T = \{ \tilde{A}_{e,T} \} \) for each user \( e \) that owns attributes satisfying \( T \) \} to determine which \( \tilde{A}^* \) is encoded in \((T_1, T_2)\). For the forgery to be nontrivial, \( \tilde{A}^* \) cannot be on the revocation list \( RL^* \); also, if \( \tilde{A}^* \in \tilde{A}_T \), letting \( \tilde{A}^* = \tilde{A}_{e^*,T} \), \( A \) should have not issued a corruption query for user \( e^* \) or issued a signature query on \( M \) regarding \( T \) for \( e^* \). If \( \tilde{A}^* \) does not belong to \( \tilde{A}_T \), \( B \) outputs \( \sigma_{M,T} \). If \( \tilde{A}^* \in \tilde{A}_T \), letting \( \tilde{A}^* = \tilde{A}_{e^*,T} \), \( B \) outputs \( \sigma_{M,T} \) when \( x_{e^*} = \star \), or declares failure and exits otherwise.

As implied by the output phase, there are three types of successful forger \( A \).

- A Type I forger outputs a forgery \( \sigma_{M,T} \) that encodes some \( \tilde{A}^* \not\in \tilde{A}_T \), but \( \tilde{A}^* \in \tilde{A}_T \) where \( \tilde{A}_T = \{ \tilde{A}_{e,T} \} \) for every \( e \) that has been corrupted by \( A \). That is, \( \tilde{A}^* \) is the same as \( \tilde{A}_{e,T} \) of some user \( e \) who does not own all the attributes appearing in \( T \) and whose private key has been queried by \( A \).
- A Type II forger outputs a forgery \( \sigma_{M,T} \) that encodes an identity \( \tilde{A}^* \) such that \( \tilde{A}^* = \tilde{A}_{e^*,T} \) for some \( e^* \) that the forger has not issued corruption query.
- A Type III forger outputs a forgery \( \sigma_{M,T} \) that encodes an \( \tilde{A}^* \) that does not belong to either \( \tilde{A}_T \) or \( \tilde{A}_F \). That is, the forger outputs a forgery signature for a user not created by \( B \).

Next, we instantiate the above framework for each of the three types of forger.
Hence, we get a new SDH pair \((\sigma_\text{generality}, g_{pk})\) are equal, and the chances to trace the signature to every user are equal as well. Without loss of 

\[
A_1(x_1, A_{1,1}), (x_2, A_{2,1}), \cdots, (x_{q-2}, A_{q-2,1}), (x_q, A_{q,1}), 
\]

which are associated with a certain \(g_1 \in \mathbb{G}_1\). Note that SDH pair \((x_{q-1}, A_{q-1,1})\) is unknown. Based on each \(T_j\) for \(j = 2, \cdots, m\), similarly, we can obtain the following \(q - 1\) SDH pairs:

\[
(x_1, A_{1,j}), (x_2, A_{2,j}), \cdots, (x_{q-2}, A_{q-2,j}), (x_{q-1}, A_{q-1,j}), 
\]

each associated with a certain \(g_j \in \mathbb{G}_1\). In the later phases of Game 1, each \((x_i, A_{1,j})\) for \(i = 1, \cdots, q - 1\) and \(j = 1, \cdots, m\) is used as the private key for user \(i\) regarding attribute \(j\). Note that, the initialization phase has the computational complexity of \(\Theta(1) \cdot (mq)\).

A \((t, q_H, q_s, n, m, \epsilon)\)-Type I forger has the probability of \(\epsilon_f = \frac{\epsilon}{2(q-1)}\) to successfully output a forgery signature \(\sigma_{M,T}\) that traces to user \(q - 1\) regarding a certain access structure \(T\) where attribute 1 appears. This is because \(x_i\) for \(i = 1, \cdots, q - 1\) are randomly picked and all users appear the same to \(A\). The chances for \(A\) to win by forging a signature regarding a \(T\) with or without attribute 1 are equal, and the chances to trace the signature to every user are equal as well. Without loss of generality, let \(g_{pk_T} = (T, \{r_1, \cdots, r_m\}, g_T, w_T)\), where \(g_T = \prod_{j=1}^{m} g_j^r\), and \(w_T = g_T^t\).

As demonstrated by the Application of Forger part of the proof of traceability in [8], with probability \((\epsilon_f - 1/p)^2/(16q_H)\) = \((\epsilon/2(q-1) - 1/p)^2/(16q_H)\), a SDH pair \((x_{q-1}, \hat{A}_{q-1,T})\) can be derived from \(\sigma_{M,T}\). Since \(\hat{A}_{q-1,T} = \prod_{j=1}^{m} A_{q-1,j}^{r_j}\), we can compute \(A_{q-1} = (\hat{A}_{q-1,T}/\prod_{j=2}^{m} A_{q-1,j}^{r_j})^{1/r_1}\). Hence, we get a new SDH pair \((x_{q-1}, A_{q-1,1})\).

Therefore, using a \((t, q_H, q_s, n, m \epsilon)\)-Type I forger, the probability to solve the \(q\)-SDH problem is \((\epsilon/2(q - 1) - 1/p)^2/(16q_H)\) in time \(t'\) where \(t' = \Theta(1) \cdot (t + mq)\).

Instantiating and applying the Framework for Type III forger Against a \((t, q_H, q_s, n, m, \epsilon)\)-Type III forger \(A\), the initialization phase of the framework is instantiated as follows. \(B\) creates \(q - 1\) users and \(m\) attributes. \(q - 1\) numbers \(x_1, \cdots, x_{q-1}\) are picked from \(\mathbb{Z}_p\) uniformly at random to be associated with the \(q - 1\) users, respectively. Based on the given \(T_1\) in Eq. (13), a certain \(g_1\) can be obtained together with \(q - 1\) SDH pairs

\[
(x_1, A_{1,1}), (x_2, A_{2,1}), \cdots, (x_{q-1}, A_{q-1,1}), 
\]
where \( A_{i,1} = g_{i}^{-\alpha_{i}} \) for each \( i = 1, \ldots, q - 1 \). Furthermore, we pick \( m - 1 \) numbers \( \alpha_{2}, \ldots, \alpha_{m} \) from \( \mathbb{Z}_{p} \) uniformly at random. Then, for each \( j = 2, \ldots, m \), we can obtain \( g_{j} = g_{j,1}^{\alpha_{j}} \) and \( q - 1 \) SDH pairs

\[
(x_{1}, A_{1,j}), (x_{2}, A_{2,j}), \ldots, (x_{q-1}, A_{q-1,j}),
\]

where for each \( i = 1, \ldots, q - 1 \), \( A_{i,j} = A_{i,1}^{\alpha_{j}} \), and thus \( A_{i,j} = g_{i}^{-\alpha_{i}} \). In the later phases of the framework, each pair \((x_{i}, A_{i,j})\) is used as the private key of user \( i \) for attribute \( j \).

A \((t, q_{H}, q_{s}, n, m, \epsilon)\)-Type III forger has the probability of \( \epsilon_{III} = \epsilon \) to successfully output a forgery signature \( \sigma_{M,T} \) that traces to a user outside of the \( q - 1 \) created users regarding a certain access structure \( T \). Without loss of generality, let \( g_{pkT} = \langle T, \{ r_{1}, \ldots, r_{m} \}, g_{T}, w_{T} \rangle \), where \( g_{T} = \prod_{j=1}^{m} g_{j}^{r_{j}} \) and \( w_{T} = g_{T}^{q\cdot T} \).

As demonstrated by the Application of Forger part of the proof of traceability in [8], with probability \( (\epsilon_{III} - 1/p)^{2}/(16q_{H}) = (\epsilon - 1/p)^{2}/(16q_{H}) \), a SDH pair \((x_{q}, \hat{A}_{q,T})\) can be derived from \( \sigma_{M,T} \), where \( x_{q} \neq x_{i} \) for \( i = 1, \ldots, q - 1 \). Since \( \hat{A}_{q,T} = \prod_{j=1}^{m} A_{q,j} = \prod_{j=1}^{m} g_{j}^{r_{j}\alpha_{j}} \), where \( \alpha_{1} = 1 \), we can compute \( A_{q,1} = \hat{A}_{q,T}^{-\sum_{j=1}^{m} r_{j}\alpha_{j}} \). Hence, we get a new SDH pair \((x_{q}, A_{q,1})\).

Therefore, using a \((t, q_{H}, q_{s}, n, m, \epsilon)\)-Type III forger, the probability to solve the \( q \)-SDH problem is \( (\epsilon - 1/p)^{2}/(16q_{H}) \) in time \( t' \) where \( t' = \Theta(1) \cdot (t + mq) \).

### Instantiating and Applying the Framework for Type II forgery

Against a \((t, q_{H}, q_{s}, n, m, \epsilon)\)-Type II forger \( \mathcal{A} \), the initialization phase of the framework is instantiated as follows. \( \mathcal{B} \) first creates \( q - 2 \) users and \( m \) attributes as in the case for Type III forger. Particularly, for each user \( i \in \{ 1, \ldots, q - 2 \} \) and each attribute \( j \in \{ 1, \ldots, m \} \), \( g_{j} \) can be obtained together with \((x_{i}, A_{i,j})\) as above. In addition, \( A_{q,1} \) is picked from \( \mathbb{Z}_{p} \) uniformly at random, and \( A_{q,j} = A_{q,1}^{\alpha_{j}} \) is constructed for each \( j = 2, \ldots, m \). Then, another user \( q \) can be created, where each pair \((*, A_{q,j})\) is used as the private key of user \( q \) for attribute \( j \).

A \((t, q_{H}, q_{s}, n, m, \epsilon)\)-Type II forger has the probability of \( \epsilon_{II} = \epsilon/q \) to successfully output a forgery signature \( \sigma_{M,T} \) that traces to user \( q \) regarding a certain access structure \( T \), because the user \( q \) has the same appearance as other \( q - 1 \) users to \( \mathcal{A} \). Without loss of generality, let \( g_{pkT} = \langle T, \{ r_{1}, \ldots, r_{m} \}, g_{T}, w_{T} \rangle \), where \( g_{T} = \prod_{j=1}^{m} g_{j}^{r_{j}} \) and \( w_{T} = g_{T}^{q\cdot T} \).

As demonstrated by the Application of Forger part of the proof of traceability in [8], with probability \( (\epsilon_{II} - 1/p)^{2}/(16q_{H}) = (\epsilon - 1/p)^{2}/(16q_{H}) \), pair \((x_{q}, \hat{A}_{q,T})\) can be derived from \( \sigma_{M,T} \). Since \( \hat{A}_{q,T} = \prod_{j=1}^{m} A_{q,j} = \prod_{j=1}^{m} g_{j}^{r_{j}\alpha_{j}} \), where \( \alpha_{1} = 1 \), we can compute \( A_{q,1} = \hat{A}_{q,T}^{-\sum_{j=1}^{m} r_{j}\alpha_{j}} \). Hence, we get a new SDH pair \((x_{q}, A_{q,1})\).

Therefore, using a \((t - mq, q_{H}, q_{s}, n, m, \epsilon)\)-Type II forger, the probability to solve the \( mq \)-SDH problem is \( (\epsilon/q - 1/p)^{2}/(16q_{H}) \) in time \( t' \) where \( t' = \Theta(1) \cdot (t + mq) \).

### Summary

Consider the above three cases together, using a \((t, q_{H}, q_{s}, n, m, \epsilon)\) breaker algorithm of Game 1, the probability to solve a \( q \)-SDH problem is at least \( (\epsilon/2(q - 1) - 1/p)^{2}/(16q_{H}) \) within time \( \Theta(1) \cdot (t + mq) \). Therefore, if the SDH is \((q, t', \epsilon')\)-hard on \( \mathcal{G}_{1} \), the proposed signature scheme is \((t, q_{H}, q_{s}, n, m, \epsilon)\)-traceable, where \( n = q - 1, \epsilon = 8n\sqrt{\epsilon q_{H} + 2n/p} \), and \( t' = \Theta(1) \cdot (t + m \cdot q) \).
Appendix 2: Proof of Theorem 3

Proof. The proof is to show: if there is algorithm $A$ $(t, q_H, q_S, n, m, \epsilon)$-breaking the selfless anonymity of the AdHocSign scheme, algorithm $B$ can be constructed to break the Decision Linear assumption in $G_1$.

Specifically, $B$ is given as input a 6-tuple $(u_0, u_1, v, h_0 = v^n_0, h_1 = u^b_1, Z) \in G_1^6$ where $u_0, u_1$ and $v$ are picked from $G_1$ and $a$ and $b$ are picked from $\mathbb{Z}_p$, uniformly at random. $B$ decides whether $Z = v^{a+b} \in G_1$ or $Z$ is randomly picked from $G_1$, through the following interactions with $A$:

- **Initialization.** $B$ runs algorithm $CO\_Setup$ to get system parameters $\mathbb{Z}_p, G_1, E : G_1 \times G_1 \rightarrow G_2$, $g \in G_1$ and $\gamma \in \mathbb{Z}_p$. It creates $n$ users denoted as $1, \ldots, n$, from which two users $e_0$ and $e_1$ are randomly picked. For each user $e \in \{1, \ldots, n\} \setminus \{e_0, e_1\}$, $x_e$ is picked from $\mathbb{Z}_p$ uniformly at random. It defines $m$ attributes denoted as $1, \ldots, m$, and runs algorithm $CO\_AttributeInit$ to get secrets $\alpha_a \in \mathbb{Z}_p$ for each attribute $a \in \{1, \ldots, m\}$. For each user $e \in \{1, \ldots, n\} \setminus \{e_0, e_1\}$, its private key $gsk_e = (x_e, A_{e,1} = (g^{\alpha_1})^{1/(\gamma + x_e)}, \ldots, A_{e,m} = (g^{\alpha_m})^{1/(\gamma + x_e)})$. $B$ randomly picks $W$ from $G_1$. The private keys $gsk_{e_0}$ and $gsk_{e_1}$ are determined as follows:

$$gsk_{e_0} = \langle *, A_{e_0,1} = (ZW/v^a)^{\alpha_1}, \ldots, A_{e_0,1} = (ZW/v^a)^{\alpha_1} \rangle,$$

and

$$gsk_{e_1} = \langle *, A_{e_1,1} = (W^b)^{\alpha_1}, \ldots, A_{e_1,1} = (W^b)^{\alpha_1} \rangle.$$

Note that, if $Z = v^{a+b}$ in the given input of Decision Linear problem, it follows that

$$A_{e_0,1} = (ZW/v^a)^{\lambda_i} = (v^{a+b}W/v^a)^{\lambda_i} = (Wv^b)^{\lambda_i} = A_{e_1,1}.$$

- **Hash Queries.** Whenever $A$ queries the hash functions, $B$ responds with random values while ensuring consistency.

- **Pre-challenge Queries.** $A$ can issue corruption, access structure initialization and signing queries. $B$ responds as follows:
  - **Corruption.** If $A$ issues a corruption query on user $e \notin \{e_0, e_1\}$, $gsk_e$ is returned. Otherwise, failure is declared and $B$ exits.
  - **Access Structure Initialization.** If $A$ issues an access structure initialization query on access structure $T$, $B$ calls $CO\_AccessStructureInit$ to get $gpk_T$ and returns it.
  - **Signing.** If $A$ issues a signing query on a message $M$ regarding access structure $T$ for user $e$, $B$ first calls $CO\_AccessStructureInit(T)$ to get $gpk_T$. If $e \notin \{e_0, e_1\}$, $B$ calls $CO\_Sign(gpk_T, gsk_e, T)$ to get signature $\sigma$ and then returns it. Otherwise, suppose $gpk_T = \langle T, \{r_{jT,1}, \ldots, r_{jT,s_T}\}, g_T = \sum_{k=1}^{s_T} (r_{jT,k} \cdot \alpha_{jT,k}), w_T \rangle$. Let

$$\beta_T = \sum_{k=1}^{s_T} (r_{jT,k} \cdot \alpha_{jT,k}),$$

$$u'_0 = u_0^\beta_T, u'_1 = u_1^\beta_T, v' = v^\beta_T,$$

$$h'_0 = h_0^\beta_T, h'_1 = h_1^\beta_T, Z' = Z^\beta_T.$$

$B$ picks randomly $s, t, l$ from $G_p$, and makes the following assignment:
as follows:

A

the CD-Game in order to leverage
cal expressions of attributes), then a
t
and access structures are general (i.e., each can be represented as conjunction-of-disjunction logi-
n
system where there are

Proof.

Appendix 3: Proof of Theorem 5

According to the same reasoning as in the proof of selfless anonymity for the VLR group signature
scheme in [8], a signature on message \( M \) regarding \( T \) can be generated for private key \((x,\hat{A}_{c,T} = g^{\beta_T/(\gamma+x)})\).

– Challenge. \( \mathcal{A} \) outputs a message \( M \), an access structure \( T \), and two users \( e_0^* \) and \( e_1^* \) where it
wishes to be challenged. If \( \{e_0^*, e_1^*\} \neq \{e_0, e_1\} \), \( \mathcal{B} \) declares failure and exits. Otherwise, \( \mathcal{A} \)
picks a random bit \( b \) from \( \{0, 1\} \) uniformly at random and generates a signature \( \sigma^* \) under user
\( e_b^* \)'s key for \( M \) regarding \( T \) using the same method used to respond to signing queries in the
pre-challenge phase. It gives \( \sigma^* \) as the challenge to \( \mathcal{A} \).

– Restricted Queries. \( \mathcal{A} \) issues restricted queries. \( \mathcal{B} \) responds as in the pre-challenge phase.

– Output. \( \mathcal{A} \) outputs its guess \( b' \in \{0, 1\} \) for \( b \). If \( b = b' \) then \( \mathcal{B} \) outputs 0 (indicating that \( Z \) is
random in \( \mathcal{G}_1 \)); otherwise \( \mathcal{B} \) outputs 1 (indicating that \( Z = v^{a+b} \)).

According to the same reasoning as in the proof of selfless anonymity for the VLR group signature
scheme [8], it follows that \( \mathcal{B} \) can solve the Decision Linear problem with advantage at least
\[
\frac{1}{t'} \left( 1 - \frac{qs\mu}{\gamma + x} \right) = \frac{1}{t'} \left( 1 - \frac{qs\mu}{\gamma + x} \right).
\]

Assuming the \( (t', \epsilon') \) Decision Linear assumption holds in group \( \mathcal{G}_1 \) (i.e., no \( t' \)-time algorithm
can solve the Decision Linear problem with advantage at least \( \epsilon' \)), the ad hoc group signature
scheme in \( \mathcal{G}_1 \) is \( (t, q_H, q_S, n, m, \epsilon) \)-selflessly-anonymous, where \( \epsilon = \frac{1}{t'} \left( 1 - \frac{qs\mu}{\gamma + x} \right) \) and \( t' = \Theta(1) \cdot (t + m \cdot n) \), considering the initialization phase may take \( \Theta(mn) \) time.

Appendix 3: Proof of Theorem 5

Proof. This proof is to show: if there is a \( t \)-time algorithm \( \mathcal{A} \) winning the traceability game in a
system where there are \( n \) users and \( m \) attributes, each attribute is associated with \( N \) secret numbers,
and access structures are general (i.e., each can be represented as conjunction-of-disjunction logical
expressions of attributes), then a \( t' \)-time algorithm \( \mathcal{B} \) can be constructed to win the traceability
game in a system where there are \( n \) users and up to \( m \cdot N + 1 \) attributes, and access structures are
conjunction-only logical expressions of attributes. Here, \( t' = \Theta(1) \cdot (t + m \cdot N \cdot q) \). Hereafter, the
traceability games mentioned above are called CD-Game (with conjunction-of-disjunction access
structures) and CO-Game (with conjunction-only access structures), respectively.

Let \( \mathcal{B} \) play with a challenger (denoted as \( \mathcal{C} \)) in the CO-Game, and meanwhile play with \( \mathcal{A} \)
in the CD-Game in order to leverage \( \mathcal{A} \) to win the CO-Game. In the games, \( \mathcal{B} \) interacts with \( \mathcal{C} \) and \( \mathcal{A} \)
as follows:

– Initialization. At the end of the initialization phase of the CO-Game, \( \mathcal{C} \) provides to \( \mathcal{B} \) system
parameters: \( \mathcal{G}_1, g \in \mathcal{G}_1, E, \) the number of users \( n \) (users are named as 1, 2, \( \cdots \), \( n \)) and the
number of attributes \( m \cdot N + 1 \) (attributes are named as 0, 1, 2, \( \cdots \), \( m \cdot N \)). Based on the
received parameters, \( B \) initializes the CD-Game as follows. It also uses \( G_1, g \) and \( E \) as system parameters. Secret number \( \xi \in \mathbb{Z}_p \) is chosen randomly. \( n \) users (called users 1, 2, \ldots, \( n \)) are created and the number of attributes is set to \( m \) (attributes are named as \( 1, \ldots, m \)). For each attribute \( a \in \{1, \ldots, m\} \), two arrays (denoted as \( A_a[1..N] \) and \( R_a[1..N] \)) each with \( N \) elements are created to be associated to the attribute, and each entry of the arrays is initialized to empty. Finally, \( U \), the set of corrupted users, is initialized to \( \emptyset \).

- **Queries**. In the CD-Game, \( A \) can make queries to \( B \). As reactions, \( B \) makes queries to \( C \) and responds to \( A \) based on the responses from \( C \).
  - **Corruption**. When \( A \) in the CD-Game determines a set of attributes \( \{a_e,i|i = 1, \ldots, n_e\} \) for user \( e \) and requests for the private key of \( e \), \( B \) reacts by (1) making a corruption query for \( e \) at attribute 0 to \( C \) and thus obtaining \( x_e \), and (2) returning to \( A \) the following private key:
    \[
gsk_e = \langle x_e, \{ A_{e,a_e,i,j}, a_e \}_{i,j} \rangle = g^{H_3(\xi,x_e,a_e,i,j)}|i = 1, \ldots, n_e; j = 1, \ldots, N \rangle,
    \]
    where \( H_3(\cdot, \cdot, \cdot, \cdot) \) is a random oracle that outputs an element of \( \mathbb{Z}_p \) on any input of four elements of \( \mathbb{Z}_p \). (3) \( e \) is added to \( U \).
  - **Access Structure Initialization**. When \( A \) in the CD-Game requests for the public key \( gpk_T \) for a certain access structure \( T = DT_1 \land \cdots \land DT_s \), where each \( DT_i = a_{T,i,1} \lor \cdots \lor a_{T,i,s_i} \) is a disjunction-only access structure, \( B \) reacts as follows.
    * In the context of the CD-Game, for each \( DT_i \) appearing in \( T \): Let \( v_i = \sum_{j=1}^{s_i} 2^{a_{T,i,j} - 1} \).
      For each attribute \( a_{T,i,j} \) appearing in \( DT_i \), its associated array \( A_{a_{T,i,j}} \) is searched to locate \( v_i \). If \( v_i \) is not found, \( v_i \) is put into an empty entry of the array; if no empty entry can be found, the games end and failure is declared. Now, let \( a'_{T,i,j} \) be an index such that \( A_{a_{T,i,j}}[a'_{T,i,j}] = v_i \). After the above steps have been performed for each \( a_{T,i,j} \) of \( DT_i \), let \( a''_{T,i,j} = (a_{T,i,j} - 1) \cdot N + a'_{T,i,j} \).
    * In the CO-Game, \( B \) constructs a conjunction-only access structure \( T' = a''_1 \land \cdots \land a''_s \) and requests \( C \) for public key \( gpk_{T'} \) regarding \( T' \), and suppose that the public key received is
      \[
gpk_{T'} = \langle T', r_{T',1}, \cdots, r_{T',s}, g_{T'}, w_{T'} \rangle.
      \]
      * In the CD-Game, for every attribute \( a_{T,i,j} \) appearing in each \( DT_i \), \( B \) chooses a value for \( R_{a_{T,i,j}}[a'_{T,i,j}] \) from \( \mathbb{Z}_p \setminus \{0\} \) uniformly at random and computes \( r_{T,i,j} = r_{T',i} \cdot R_{a_{T,i,j}}[a'_{T,i,j}] \). It also computes \( h_{T,i,j} = H_2(\xi, a_{T,i,j}, a'_{T,i,j}) \), where \( H_2(\cdot, \cdot, \cdot, \cdot) \) is a random oracle which outputs an element of \( \mathbb{Z}_p \) on any input of three elements of \( \mathbb{Z}_p \).
      Finally, the following public key is returned to \( A \):
      \[
gpk_T = \langle T, \{(r_{T,i,j}, h_{T,i,j})|i = 1, \ldots, s_i; \text{ for each } i, j = 1, \ldots, s_i\}, g_{T'}, w_{T'} \rangle.
      \]
  - **Signing**. Whenever \( A \) in CD-Game requests for the signature of user \( e \) on message \( M \) regarding a certain access structure \( T \), \( B \) reacts in the following steps.
    * As in Access Structure Initialization, access structure \( T' \) in the CO-Game is obtained from the access structure \( T \) in the CD-Game.
    * In the CO-Game, \( B \) requests \( C \) for the signature of \( e \) on message \( M \) regarding \( T' \). Upon receiving the signature \( \sigma \), \( B \) returns the signature to \( A \) in the CD-Game.
  - **Hash Queries**. In response to \( H_1(e, a, a', h) \): If entry \( A_a[a'] \) is empty (i.e., the \( a' \)-th secret associated with attribute \( a \) has not been used in host preparation yet), an arbitrary value is returned. Otherwise:
* Let $DT = a_1 \lor \cdots \lor a_s$ (where $a_1 < \cdots < a_s$) such that $A_a[a'] = \sum_{j=1}^{s} 2^{a_j} - 1$. Array $A_{a_1}$ is searched to find out an index $a'_1$ such that $A_{a_1}[a'_1] = A_a[a']$. A compromise query for $e$ with attribute $a'_1$ is made to $C$.

* Upon receiving the private key $(x_e, A')$, the actual $a'_1$-th private key for $e$ on attribute $a$ is computed as $A_{e,a,a'} = (A')^{(1/R_a[a'])}$.

* The hash value is returned as $A_{e,a,a'} \cdot (A'_{e,a,a'})^{-1} = A_{e,a,a'} \cdot g^{-H_3(\xi,x_e,a,a')}$.

In response to queries on other hash functions (i.e., $H_0$, $H$, $H_1$ and $H_2$), algorithm $B$ returns random values while ensuring consistency.

- **Response.** If $A$ in CD-Game outputs $M^*$, a certain access structure $T$, a set $RL^*$ of revocation tokens and a signature $\sigma^*$, $B$ in CO-Game also outputs $M^*$, access structure $T'$ which is converted from $T$ as in the handling of Access Structure Initialization queries, $RL^*$ and $\sigma^*$.

According to the above strategy, if $A$ wins the CD-Game, $B$ must also win the CO-Game, because

- If $\sigma^*$ is accepted as a valid signature on $M^*$ regarding $T$ in the system where CD-Game is played, the signature is also accepted as a valid signature on $M^*$ regarding $T'$ in the system where CO-Game is played, where $T'$ is converted from $T$ as in the handling of access structure initialization queries.

- If $\sigma^*$ traces to some user $e$ outside of $U \setminus RL^*$ in the CD-Game, it also traces to the same user $e$ in the CO-Game.

- If the tracing fails in the CD-Game system (i.e., it cannot trace to any user), it cannot trace to any user either in the CO-Game because users in the two games are one-to-one correspondent.

- If $A$ never makes a signing query at $M^*$ for user $e$ on access structure $T$ in the CD-Game, $B$ never makes a signing query at $M^*$ for user $e$ on access structure $T'$ in the CO-Game, where $T'$ is converted from $T$ as in the handling of access structure initialization queries.

Also according to algorithm $B$, algorithm $B$ needs computational complexity of $O(m \cdot N)$ to handle each type of query issued by $A$. Hence, if $A$ needs time $t$, $q_S$ signing queries and $q_H$ queries on hash functions $H_0$ and $H$ to win the CD-Game, then algorithm $B$ needs time $t'$, $q_S$ signing queries and $q_H$ queries on $H_0$ and $H$ to win the CO-Game, where $t' = O(t \cdot m \cdot N)$. Therefore, if the AdHocSign scheme for conjunction-only access structures is $(t', q_S, q_H, n, m, \epsilon)$-traceable, then the AdHocSign scheme for general access structures is $(t, q_S, q_H, n, m, \epsilon)$-traceable.