Provably Secure Authenticated Tree Based Group Key Agreement

Abstract

We present a provably secure authenticated tree based key agreement protocol. The protocol is obtained by combining Boldyreva's multi-signature with an unauthenticated ternary tree based multi-party extension of Joux's key agreement protocol. The security is in the standard model as formalized by Bresson *et al.*. The proof is based on the techniques used by Katz and Yung in proving the security of their key agreement protocol.
Keywords : group key agreement, authenticated key agreement, bilinear pairing, provable security.

1 Introduction

Key agreement protocols are important cryptographic primitives. These protocols allow two or more parties to exchange information among themselves over an insecure channel and agree upon a common key. Diffie-Hellman proposed the first two-party single-round key agreement protocol in their seminal paper [11]. In one of the breakthroughs in key agreement protocols, Joux [14] proposed a single-round three party key agreement protocol that uses bilinear pairings. Earlier, Burmester and Desmedt [10] had proposed a multi-party two-round key agreement protocol. Katz and Yung [15] proved that a variant of this protocol is secure against a passive adversary in the standard model under decision Diffie-Hellman assumption. All these protocols are unauthenticated in the sense that an active adversary who has control over the channel can mount a man-in-the-middle attack to agree upon separate keys with the users without the users being aware of this.

Authenticated key agreement protocols allow two or more parties to agree upon a common secret key even in the presence of active adversaries. Authenticated key agreement protocols are the basic tools for group-oriented and collaborative applications such as, distributed simulation, multi-user games, audio or video-conferencing, and also peer-to-peer applications that are likely to involve a large number of users.

Recently, Katz and Yung [15] proposed the first scalable, constant round, authenticated group key agreement protocol achieving forward secrecy. The protocol is a variant of Burmester and Desmedt (BD) [10] key agreement protocol. Katz and Yung [15] provided a detailed proof of security in the security model formalized by Bresson *et al.* [9].

Tree based key agreement protocols are applicable in situations where the users can be naturally grouped into a hierarchical structure. Each leaf node in the tree corresponds to an individual user and each internal node corresponds to a user who acts as a representative for the set of users in the subtree rooted at that node. Typically the representative user has more computational resources than the other users in the subtree. In a tree based group key agreement protocol the set of all users in each subtree agree upon a common secret key. Thus all users in a subtree can securely communicate among themselves using this common secret key. This feature makes tree based key agreement protocols useful for certain applications and there have been quite a number of tree based key agreement protocols [2, 5, 16, 17, 18]. Barua, Dutta and Sarkar [4] presented a ternary tree key agreement protocol by extending the basic Joux protocol to multi-party setting and provided a proof of security against a passive adversary.

The main contribution of this paper is to obtain a provably secure authenticated tree based group key agreement protocol from the unauthenticated protocol of Barua, Dutta and Sarkar [4]. The security of the authenticated protocol is considered in the model formalized by Bresson *et al* [9]. Authentication is achieved by modifying the Katz-Yung [15] technique to tree based protocol. The main component of the Katz-Yung authenticated protocol is the use of a digital signature scheme. We also use a signature scheme, where each user signs every message that (s)he sends. However, in keeping with the tree structure of the protocol, we require that in each round each user sends the signed message to his/her representative. The representative then combines these signatures and sends the multi-signature to the other users. This results in savings in the total amount of communication.

To a certain extent, our work completes the task of extending Joux's three-party key agreement protocol to multi-party setting with a concrete security analysis against active adversaries in a formal security model. Moreover, a similar construction can be used to modify any tree based unauthenticated group key agreement protocol to an authenticated group key agreement protocol.

2 Preliminaries

In this section, we describe the required preliminaries. We use the notation $a \in_R S$ to denote that a is chosen randomly from the set S.

2.1 Cryptographic Bilinear Maps

Let G_1, G_2 be two groups of the same prime order q. We view G_1 as an additive group and G_2 as a multiplicative group. A mapping $e : G_1 \times G_1 \to G_2$ satisfying the following properties is called a cryptographic bilinear map:

Bilinearity : $e(aP, bQ) = e(P, Q)^{ab}$ for all $P, Q \in G_1$ and $a, b \in Z_q^*$.
Non-degeneracy : If P is a generator of G_1, then $e(P, P)$ is a generator of G_2.
Computablity : There exists an efficient algorithm to compute $e(P, Q)$ for all $P, Q \in G_1$.

Modified Weil Pairing [6] and Tate Pairing [3, 12] are examples of cryptographic bilinear maps.

2.2 Adversarial Model

The adversarial model that we follow is along the lines proposed by Bresson *et al.* [9] and used by Katz and Yung [15].

Let $\mathcal{P} = \{U_1, \ldots, U_n\}$ be a set of n (fixed) users. At any point of time, any subset of $\mathcal{P}$ may decide to establish a session key. Thus a user can execute the protocol several times with different partners. The adversarial model consists of allowing each user an unlimited number of instances with which it executes the protocol. We will require the following notions.

Π_U^i : i-th instance of user U.
sk_U^i : session key after execution of the protocol by Π_U^i.
sid_U^i : session identity for instance Π_U^i. We set $\mathsf{sid}_U^i = S = \{(U_1, i_1), \ldots, (U_k, i_k)\}$ such that $(U, i) \in S$ and $\Pi_{U_1}^{i_1}, \ldots, \Pi_{U_k}^{i_k}$ wish to agree upon a common key.
pid_U^i : partner identity for instance Π_U^i, defined by $\mathsf{pid}_U^i = \{U_1, \ldots, U_k\}$, such that $(U_j, i_j) \in \mathsf{sid}_U^i$ for all $1 \leq j \leq k$.
acc_U^i : 0/1-valued variable which is set to be 1 by Π_U^i upon normal termination of the session and 0 otherwise.

We will make the assumption that in each session at most one instance of each user participates. Further, an instance of a particular user participates in exactly one session. This is not a very restrictive assumption, since a user can spawn an instance for each session it participates in. On the other hand, there is an important consequence of this assumption. Suppose there are several sessions which are being concurrently executed. Let the session ID's be $S_1, \ldots, S_k$. Then for any instance Π_U^i, there is exactly one j such that $(U, i) \in S_j$ and for any $j_1 \neq j_2$, we have $S_{j_1} \cap S_{j_2} = \emptyset$. Thus at any particular point of time, if we consider the collection of all instances of all users, then the relation of being in the same session is an equivalence relation whose equivalence classes are the session IDs.

We assume that the adversary has complete control over all communications in the network. All information that the adversary gets to see is written in a transcript. So a transcript consists of all the public information flowing across the network. The following oracles model an adversary's interaction with the users in the network:

- $\mathsf{Send}(U, i, m)$: This sends message m to instance Π_U^i and outputs the reply (if any) generated by this instance. The adversary is allowed to prompt the unused instance Π_U^i to initiate the protocol with partners $U_2, \ldots, U_l$, $l \leq n$, by invoking $\mathsf{Send}(U, i, \langle U_2, \ldots, U_l \rangle)$.

- $\mathsf{Execute}((V_1, i_1), \ldots, (V_l, i_l))$: Here $\{V_1, \ldots, V_l\}$ is a non empty subset of $\mathcal{P}$. This executes the protocol between unused instances of players $\Pi_{V_1}^{i_1}, \ldots, \Pi_{V_l}^{i_l} \in \mathcal{P}$ and outputs the transcript of the execution.

- $\mathsf{Reveal}(U, i)$: This outputs session key sk_U^i.

- $\mathsf{Corrupt}(U)$: This outputs the long-term secret key (if any) of player U.

- $\mathsf{Test}(U, i)$: This query is allowed only once, at any time during the adversary's execution. A random $\mathsf{coin} \in \{0, 1\}$ is generated; the adversary is given sk_U^i if $\mathsf{coin} = 1$, and a random session key if $\mathsf{coin} = 0$.

An adversary which has access to the Execute, Reveal, Corrupt and Test oracles, is considered to be passive while an active adversary is given access to the Send oracle in addition. We say that an instance Π_U^i is *fresh* unless either the adversary, at some point, queried $\mathsf{Reveal}(U, i)$ or $\mathsf{Reveal}(U', j)$ with $U' \in \mathsf{pid}_U^i$ or the adversary queried $\mathsf{Corrupt}(V)$ (with $V \in \mathsf{pid}_U^i$) before a query of the form $\mathsf{Send}(U, i, *)$ or $\mathsf{Send}(U', j, *)$ where $U' \in \mathsf{pid}_U^i$.

Let Succ denote the event that an adversary $\mathcal{A}$ queried the Test oracle to a protocol XP on a fresh instance Π_U^i for which $\mathsf{acc}_U^i = 1$ and correctly predicted the coin used by the Test oracle in answering this query. We define

$$\mathsf{Adv}_{\mathcal{A},\mathsf{XP}} := |2\ \mathsf{Prob}[\mathsf{Succ}] - 1|$$

to be the advantage of the adversary $\mathcal{A}$ in attacking the protocol XP. The protocol XP is said to be a *secure unauthenticated group key agreement* (KA) protocol if there is no polynomial time *passive* adversary with non-negligible advantage. In other words, for every probabilistic, polynomial-time, 0/1 valued algorithm $\mathcal{A}$, $\mathsf{Adv}_{\mathcal{A},\mathsf{XP}} < \frac{1}{M^L}$ for every fixed $L > 0$ and sufficiently large integer M. We say that protocol XP is a *secure authenticated group key agreement* (AKA) protocol if there is no polynomial time *active* adversary with non-negligible advantage. Next we define

$\mathsf{Adv}_{\mathsf{XP}}^{\mathsf{KA}}(t, q_E)$:= the maximum advantage of any passive adversary attacking protocol XP, running in time t and making q_E calls to the Execute oracle.

$\mathsf{Adv}_{\mathsf{XP}}^{\mathsf{AKA}}(t, q_E, q_S)$:= the maximum advantage of any active adversary attacking protocol XP, running in time t and making q_E calls to the Execute oracle and q_S calls to the Send oracle.

2.3 Decision Hash Bilinear Diffie-Hellman (DHBDH) Problem

Let (G_1, G_2, e) be as in Section 2.1. We define the following problem.

Instance : (P, aP, bP, cP, r) for some $a, b, c, r \in_R Z_q^*$ and a one way hash function $H : G_2 \to Z_q^*$.
Solution : Output yes if $r = H(e(P,P)^{abc}) \bmod q$ and output no otherwise.

The DHBDH problem is defined in [4] and is a combination of the bilinear Diffie-Hellman(BDH) problem [6] and a variation of the hash Diffie-Hellman(HDH) problem [1]. The advantage of any probabilistic, polynomial time, 0/1-valued algorithm $\mathcal{A}$ in solving DHBDH problem in (G_1, G_2, e) is defined to be:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{DHBDH}} = \mathsf{Prob}[\mathcal{A}(P, aP, bP, cP, r) = 1] - \mathsf{Prob}[\mathcal{A}(P, aP, bP, cP, H(e(P,P)^{abc})) = 1]$$

The probabilities are computed over $a, b, c, r \in_R Z_q^*$.
DHBDH assumption : There exists no probabilistic, polynomial time, 0/1-valued algorithm which can solve the DHBDH problem with non-negligible probability of success. In other words, for every probabilistic, polynomial time, 0/1-valued algorithm $\mathcal{A}$, $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{DHBDH}} < \frac{1}{M^L}$ for every fixed $L > 0$ and sufficiently large integer M.

2.4 Unauthenticated Protocol

We describe the multi-party extension [4] of Joux's unauthenticated protocol in Section 3.1. This unauthenticated protocol UP executes a single Execute oracle. Since it does not involve any long term public/private keys, Corrupt oracles may simply be ignored and thus the protocol achieves forward secrecy. The protocol UP has been proved to be secure against passive adversary [4] under DHBDH assumption for a single Execute query. This proof can be extended for the case of multiple Execute queries by using standard hybrid argument techniques: If $\mathsf{Adv}_{\mathsf{UP}}^{\mathsf{KA}}(t, 1)$ is the advantage of the protocol UP for a single query to the Execute oracle, then with q_E queries to the Execute oracle, the advantage of UP is

$$\mathsf{Adv}_{\mathsf{UP}}^{\mathsf{KA}}(t, q_E) \leq q_E \ \mathsf{Adv}_{\mathsf{UP}}^{\mathsf{KA}}(t, 1).$$

2.5 Multi-signatures

We will use multi-signatures to obtain the authenticated protocol. Multi-signatures allow a group of users to sign a message, such that a verifier can verify that all users indeed signed the message. For our application, we require the signing protocol to be *non-interactive* as in [8]. In fact, any non-interactive multi-signature scheme can be used with our protocol though we note that currently the only such known scheme has been presented by Boldyreva [8] and is based on the Boneh-Lynn-Shacham [7] pairing based short signature scheme. Our description of multi-signatures is based on the protocol presented in [8].

Let $\mathcal{P} = \{U_1, \ldots, U_n\}$ be the set of users who will be involved in generation of signatures and multi-signatures. Let $G_1 = \langle P \rangle, G_2$ (groups of prime order q) and $e(,)$ be as defined in Section 2.1. Let $\mathsf{DSig} = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ be the BLS short signature scheme [7]. As part of this scheme each user U_i generates a signing key $sk_i \in Z_q^*$ and a verification key $pk_i = sk_i P$ by invoking the BLS key generation algorithm $\mathcal{K}$. The signature on a message m is generated by the i-th user as $\sigma = sk_i H_1(m)$, where H_1 is a hash function which maps arbitrary length strings to G_1. Verification of a message-signature pair (m, σ) is performed by checking whether $e(P, \sigma)$ is equal to $e(pk_i, H_1(m))$.

The Boldyreva multi-signature scheme is based on the BLS signature scheme and can be described as follows.

1. Let $L = \{U_{i_1}, \ldots, U_{i_k}\} \subseteq \mathcal{P}$ be the set of users who wish to sign a message m.
2. Each user $U_{i_j} \in L$ generates a signature σ_{i_j} on m using the BLS signing algorithm $\mathcal{S}$.
3. All signatures are transmitted (over a public channel) to a representative user $U \in L$.
4. U combines all the signatures to obtain $\sigma_L = \sigma_{i_1} + \cdots + \sigma_{i_k}$.
5. U outputs (L, σ_L) as the multi-signature of L on the message m.

The verification protocol runs as follows.

1. A user V receives the multi-signature (L, σ_L) on a message m.
2. V computes $pk_L = pk_{i_1} + \cdots + pk_{i_k}$, where $L = \{U_{i_1}, \ldots, U_{i_k}\}$.
3. V accepts if and only if the BLS verification algorithm $\mathcal{V}$ returns true for the public key pk_L on the message-signature pair (m, σ_L).

2.5.1 Security of Multi-signature Scheme

Formally, a multi-signature scheme consists of three algorithms $\mathsf{MSig} = (\mathcal{MK}, \mathcal{MS}, \mathcal{MV})$, where $\mathcal{MK}$ is the key generation algorithm; $\mathcal{MS}$ is the signature generation algorithm and $\mathcal{MV}$ is the signature verification algorithm. We say that a multi-signature scheme is secure against *existential forgery under chosen message attack* if the following task is computationally infeasible for the adversary:

The adversary is given a public key pk_1; then outputs $(n-1)$ pairs of public and secret keys $pk_2, \ldots, pk_n$ and $sk_2, \ldots, sk_n$ respectively; is allowed to run the multi-signature generation algorithm $\mathcal{MS}$ with user U_1 having public key pk_1 on messages of the adversary's choosing; finally has to produce a message m, a subset L of users with $U_1 \in L$ and a signature σ such that $\mathcal{MV}$ returns true on input (L, m, σ) and U_1 did not participate in multi-signature generation algorithm for message m.

We denote by $\mathsf{Succ}_{\mathsf{DSig}}(t)$ the maximum success probability of any adversary running in time t to forge signatures for the BLS signature scheme DSig. Similarly, by $\mathsf{Succ}_{\mathsf{MSig}}(t)$ the maximum success probability of any adversary running in time t to break the Boldyreva multi-signature scheme MSig.

Both the BLS short signature scheme and the Boldyreva multi-signature scheme have been proved to be secure in the random oracle model. Since we use both the signature schemes in our authenticated protocol, the complete security of our protocol is also in the random oracle model. However, we do not actually require the random oracle in our security proof. Thus if one can replace the BLS signature scheme and the Boldyreva multi-signature scheme with schemes which are secure in the standard model, then the security of our protocol is also obtained in the standard model.

3 The Protocol

We start by describing the requirements of the protocol, follow it up with a description of the unauthenticated protocol from [4] and finally describe the authenticated protocol.

Suppose a set of n users $\mathcal{P} = \{U_1, U_2, \ldots, U_n\}$ wish to agree upon a secret key. Let US be a subset of users. Each such user set US has a representative $\mathsf{Rep}(\mathsf{US})$ and for the sake of concreteness we take $\mathsf{Rep}(\mathsf{US}) = U_j$ where $j = \min\{k : U_k \in \mathsf{US}\}$. We use the notation $A[1, \ldots, n]$ for an array of n elements $A_1, \ldots, A_n$ and write $A[i]$ or A_i to denote the ith element of array $A[\,]$.

Let $G_1 = \langle P \rangle, G_2$ (groups of prime order q) and $e(,)$ be as described in Section 2.1. We choose a hash function $H : G_2 \to Z_q^*$. The public parameters are $\mathsf{params} = (G_1, G_2, e, q, P, H)$.

3.1 Unauthenticated Key Agreement Protocol of [4]

First we describe the idea behind the n-party key agreement protocol of Barua, Dutta and Sarkar [4] which is an extension of Joux's three-party single-round key agreement protocol to multi-party setting.

Let $p = \lfloor \frac{n}{3} \rfloor$ and $r = n \bmod 3$. The set of users is partitioned into three user sets $\mathsf{US}_1, \mathsf{US}_2, \mathsf{US}_3$ with respective cardinalities being p, p, p if $r = 0$; $p, p, p+1$ if $r = 1$; and $p, p+1, p+1$ if $r = 2$. This top down procedure is used recursively for further partitioning. Essentially a ternary tree structure is obtained. The lowest level 0 consists of singleton users having a secret key. The formal description of the protocol is given below.

procedure KeyAgreement$(l, \mathsf{US}[i+1, \ldots, i+l], \mathsf{S}[i+1, \ldots, i+l])$
1. **if** $(l = 2)$ **then**
2. **call** CombineTwo$(\mathsf{US}[i+1, i+2], \mathsf{S}[i+1, i+2])$;
3. return;
4. **end if**
5. **if** $(l = 3)$ **then**
6. **call** CombineThree$(\mathsf{US}[i+1, i+2, i+3], \mathsf{S}[i+1, i+2, i+3])$;
7. return;
8. **end if**
9. $p_0 = 0$; $p_1 = \lfloor l/3 \rfloor$; $p_3 = \lceil l/3 \rceil$; $p_2 = l - p_1 - p_3$;
10. $n_0 = 0$; $n_1 = p_1$; $n_2 = p_1 + p_2$;
11. **for** $j = 1$ to 3 **do in parallel**
12. $\widehat{\mathsf{US}}_j = \mathsf{US}[i + n_{j-1} + 1, \ldots, i + n_{j-1} + p_j]$;
13. **if** $p_j = 1$, **then** $\widehat{\mathsf{S}}_j = S[i + n_{j-1} + 1]$;
14. **else**
15. **call** KeyAgreement$(p_j, \widehat{\mathsf{US}}_j, \mathsf{S}[i + n_{j-1} + 1, \ldots, i + n_{j-1} + p_j])$;
16. Let $\widehat{\mathsf{S}}_j$ be the common agreed key among all members of $\widehat{\mathsf{US}}_j$;
17. **end if;**
18. **end for**;
19. **call** CombineThree$(\widehat{\mathsf{US}}[1,2,3], \widehat{\mathsf{S}}[1,2,3])$;

end KeyAgreement

procedure CombineTwo$(\mathsf{US}[1,2], \mathsf{S}[1,2])$
1. **do** Steps 2 and 3 **in parallel**
2. US_1 generates $\mathsf{S} \in_R Z_q^*$ and sends $\mathsf{S}P$ and $\mathsf{S}_1 P$ to US_2;
3. US_2 sends $\mathsf{S}_2 P$ to US_1;
4. **end do**;
5. **do** steps 6 and 7 **in parallel**
6. US_1 computes $H(e(\mathsf{S}_2 P, SP)^{\mathsf{S}_1})$;
7. US_2 computes $H(e(\mathsf{S}_1 P, SP)^{\mathsf{S}_2})$;
8. **end do**;

end CombineTwo

procedure CombineThree$(\mathsf{US}[1,2,3], \mathsf{S}[1,2,3])$
1. **for** $i = 1$ to 3 **do in parallel**
2. Let $\{j, k\} = \{1, 2, 3\} \setminus \{i\}$;
3. Rep(US_i) sends $\mathsf{S}_i P$ to all members $\mathsf{US}_j \cup \mathsf{US}_k$;

4. **end for**;
5. **for** $i = 1$ to 3 **do in parallel**
6. let $\{j, k\} = \{1, 2, 3\} \setminus \{i\}$;
7. each member of US_i computes $H(e(\mathsf{S}_j P, \mathsf{S}_k P)^{\mathsf{S}_i})$;
8. **end for**;

end CombineThree

The start of the recursive protocol KeyAgreement is made by the following statements:

start main
1. $\mathsf{US}_j = \{U_j\}$ for $1 \leq j \leq n$;
2. User j chooses a secret $s_j \in_R Z_q^*$;
3. User j sets $\mathsf{S}[j] = s_j$;
4. **call** KeyAgreement$(n, \mathsf{US}[1, \ldots, n], \mathsf{S}[1, \ldots, n])$.

end main

The values $s_1, \ldots, s_n$ are session specific and determine the final common key for the users. Note that CombineTwo is invoked only for two individual users, (i.e. $|\mathsf{US}_1| = |\mathsf{US}_2| = 1$), whereas CombineThree is invoked for three individual users as well as for three groups of users. In CombineThree the common agreed key of user sets $\mathsf{US}_1, \mathsf{US}_2, \mathsf{US}_3$ is $H(e(P, P)^{\mathsf{S}_1\mathsf{S}_2\mathsf{S}_3})$ and in CombineTwo the common agreed key of the two users in the singleton sets $\mathsf{US}_1, \mathsf{US}_2$ is $H(e(P, P)^{\mathsf{S}_1\mathsf{S}_2\mathsf{S}})$.

The protocol described above allows $U_1, \ldots, U_n$ to agree upon a common key. The same protocol can be used by an arbitrary subset of $\{U_1, \ldots, U_n\}$ to agree upon a common key. This feature will be explained in more details for the authenticated protocol.

3.2 Authenticated Key Agreement Protocol

Authentication is obtained by incorporating signature schemes into the unauthenticated protocol of Section 3.1. More specifically, we use the BLS short signature and the Boldyreva multi-signature scheme based on it. As part of the two signature schemes each user U_i chooses a signing and a verification key sk_i(or sk_{U_i}) and pk_i(or pk_{U_i}) respectively.

An important issue in authenticating the protocol is that of session ID. Recall that Π_U^i is the ith instance of user U. Suppose instances $\Pi_{U_{i_1}}^{d_1}, \ldots, \Pi_{U_{i_k}}^{d_k}$ wish to agree upon a common key. According to our definition, $\mathsf{sid}_{U_{i_j}}^{d_j} = \{(U_{i_1}, d_1), \ldots, (U_{i_k}, d_k)\}$. At the start of a session, $\Pi_{U_{i_j}}^{d_j}$ need not know the entire set $\mathsf{sid}_{U_{i_j}}^{d_j}$. This set is built up as the protocol proceeds. Of course, we assume that $\Pi_{U_{i_j}}^{d_j}$ knows the pair (U_{i_j}, d_j). Clearly, $\Pi_{U_{i_j}}^{d_j}$ knows U_{i_j}. Knowledge of d_j can be maintained by U_{i_j} by keeping a counter which is incremented when a new instance is created. Each instance keeps the partial information about the session ID in a variable psid_U^i. Before the start of a session an instance $\Pi_{U_{i_j}}^{d_j}$ sets $\mathsf{psid}_{U_{i_j}}^{d_j} = \{(U_{i_j}, d_j)\}$. On the other hand, we do assume that any instance $\Pi_{U_{i_j}}^{d_j}$ knows its partner ID $\mathsf{pid}_{U_{i_j}}^{d_j}$, i.e., the set of users with which it is partnered in the particular session.

Another issue is the numbering of the messages. Any instance Π_U^i sends out a finite number of messages, which can be uniquely numbered by Π_U^i based on their order of occurrence. The number of instances participating in a session determines the leaf level of the ternary key agreement tree which in turn uniquely determines the ternary tree structure. Once the ternary tree structure is defined, the numbering of the messages is also defined uniquely. More specifically, if US is a user set at some intermediate point in the

execution of a session and $U = \mathsf{Rep}(\mathsf{US})$, then all instances in US knows the number of the next message to be sent out by U.

We now explain the signing part of the protocol. All messages are exchanged as part of CombineTwo and CombineThree and hence signing is also introduced as part of CombineTwo and CombineThree. The call to CombineTwo and CombineThree in Steps 2 and 6 of KeyAgreement involve individual users whereas the call to CombineThree in Step 19 of KeyAgreement are on three sets of users. In the first case, we require only the BLS signature scheme, whereas in the second case we require the multi-signature scheme.

We now describe the modified versions of CombineTwo and CombineThree. In these algorithms, US will represent sets of instances (as opposed to sets of users as in Section 3.1). In case US is a singleton set (as in AuthCombineTwo and AuthCombineThree-A) we will identify US with the instance it contains. We also define $\mathsf{Rep}(\mathsf{US}) = \Pi_{U_j}^{d_j}$ where $j = \min\{k : \Pi_{U_k}^{d_k} \in \mathsf{US}\}$.

AuthCombineTwo(US[1,2],S[1,2])

1. **perform** Steps 2,3 and 4 **in parallel** with Steps 5,6, and 7;
2. $\mathsf{US}_1 = \Pi_{U_1}^{d_1}$ generates $\mathsf{S} \in_R Z_q^*$ and forms $m_1 = (\mathsf{S}P, \mathsf{S}_1 P)$;
3. US_1 computes $\sigma_1 = \mathcal{S}(sk_{U_1}, \mathsf{US}_1|1|m_1)$;
4. US_1 sends $\mathsf{US}_1|1|m_1|\sigma_1$ to US_2;
5. $\mathsf{US}_2 = \Pi_{U_2}^{d_2}$ sets $m_2 = \mathsf{S}_2 P$;
6. US_2 computes $\sigma_2 = \mathcal{S}(sk_{U_2}, \mathsf{US}_2|1|m_2)$;
7. US_2 sends $\mathsf{US}_2|1|m_2|\sigma_2$ to US_1;
8. **end**;
9. **for** $i = 1, 2$ **do in parallel**
10. set $i' = 3 - i$; let $\mathsf{US}_i = \Pi_{U_i}^{d_i}$; $\mathsf{US}_{i'} = \Pi_{U_{i'}}^{d_{i'}}$;
11. US_i verifies $\sigma_{i'}$ on $\mathsf{US}_{i'}|1|m_{i'}$ using $pk_{U_{i'}}$ and algorithm $\mathcal{V}$;
12. **if** verification fails, **then** US_i sets $\mathsf{acc}_{U_i}^{d_i} = 0$, $\mathsf{sk}_{U_i}^{d_i} = \mathsf{NULL}$ and aborts;
13. **else** US_i sets $\mathsf{psid}_{U_i}^{d_i} = \mathsf{psid}_{U_i}^{d_i} \cup \{(U_{i'}, d_{i'})\}$;
14. **end for**;
15. US_1 computes $H(e(\mathsf{S}_2 P, \mathsf{S}P)^{\mathsf{S}_1}$;
16. US_2 computes $H(e(\mathsf{S}_1 P, \mathsf{S}P)^{\mathsf{S}_2}$;

end AuthCombineTwo

There are two versions of CombineThree for the authenticated protocol.

AuthCombineThree-A(US[1, 2, 3],S[1, 2, 3])

1. **for** $i = 1, 2, 3$ **do in parallel**
2. set $\{j, k\} = \{1, 2, 3\} \setminus \{i\}$;
3. $\mathsf{US}_i (= \Pi_{U_i}^{d_i})$ computes $\sigma_i = \mathcal{S}(sk_{U_i}, \mathsf{US}_i|1|\mathsf{S}_i P)$;
4. US_i sends $\mathsf{US}_i|1|\mathsf{S}_i P|\sigma_i$ to US_j and US_k;
5. **end for**;
6. **for** $i = 1, 2, 3$ **do in parallel**
7. set $\{j, k\} = \{1, 2, 3\} \setminus \{i\}$; let $\mathsf{US}_i = \Pi_{U_i}^{d_i}$; $\mathsf{US}_j = \Pi_{U_j}^{d_j}$; $\mathsf{US}_k = \Pi_{U_k}^{d_k}$;
8. US_i verifies
9. • σ_j on $\mathsf{US}_j|1|\mathsf{S}_j P$ using pk_{U_j};
10. • σ_k on $\mathsf{US}_k|1|\mathsf{S}_k P$ using pk_{U_k};
11. **if** any of the above verification fails **then**
12. US_i sets $\mathsf{acc}_{U_i}^{d_i} = 0$, $\mathsf{sk}_{U_i}^{d_i} = \mathsf{NULL}$ and aborts;

13. **else**
14. US_i sets $\mathsf{psid}_{U_i}^{d_i} = \mathsf{psid}_{U_i}^{d_i} \cup \{(U_j, d_j), (U_k, d_k)\}$;
15. US_i computes $H(e(\mathsf{S}_j P, \mathsf{S}_k P)^{\mathsf{S}_i})$;
16. **end if**;
17. **end for**;

end AuthCombineThree-A

The next authenticated variation of CombineThree uses multi-signature. We need a notation to describe the algorithm. Suppose $\mathsf{psid}_U^i = \{(U_{i_1}, d_1), \ldots, (U_{i_k}, d_k)\}$ is the (partial) session ID for instance Π_U^i. Then $\mathsf{First}(\mathsf{psid}_U^i)$ is defined to be the set $\{U_{i_1}, \ldots, U_{i_k}\}$. In the next algorithm, we use the notation $\Pi_V^{d_V}$ to denote an instance of a user V. The value of the instance number d_V is not uniquely determined by V. However, it is unique for one particular session. Since one particular invocation of the next algorithm will involve only one session, there will be no confusion in the use of the notation d_V.

AuthCombineThree-B(US[1,2,3],S[1,2,3])

1. **for** $i = 1, 2, 3$ **do in parallel**
2. set $\{j, k\} = \{1, 2, 3\} \setminus \{i\}$; $\Pi_U^{d_U} = \mathsf{Rep}(\mathsf{US}_i)$;
3. **for** each $\Pi_V^{d_V} \in \mathsf{US}_i$ **do in parallel**
4. $\Pi_V^{d_V}$ computes $\sigma_V = \mathcal{S}(sk_V, \mathsf{psid}_V^{d_V} | t_i | \mathsf{S}_i P)$, where t_i is number of next message to be sent by $\Pi_U^{d_U}$;
5. $\Pi_V^{d_V}$ sends (V, σ_V) to U;
6. **end for**;
7. $\Pi_U^{d_U}$ computes the multi-signature $\sigma_{\mathsf{US}_i} = \mathcal{MS}(\{\sigma_V : V \in \mathsf{US}_i\})$;
8. $\Pi_U^{d_U}$ sends $\mathsf{psid}_U^{d_U} | t_i | \mathsf{S}_i P | \sigma_{\mathsf{US}_i}$ to $\mathsf{US}_j \cup \mathsf{US}_k$;
9. **end for**;
10. **for** $i = 1, 2, 3$ **do in parallel**
11. set $\{j, k\} = \{1, 2, 3\} \setminus \{i\}$; $\Pi_{W_j}^{d_{W_j}} = \mathsf{Rep}(\mathsf{US}_j)$; $\Pi_{W_k}^{d_{W_k}} = \mathsf{Rep}(\mathsf{US}_k)$;
12. **for** each $\Pi_V^{d_V} \in \mathsf{US}_i$ **do in parallel**
13. $\Pi_V^{d_V}$ receives $\mathsf{psid}_{W_j}^{d_{W_j}} | t_j | \mathsf{S}_j P | \sigma_{\mathsf{US}_j}$ from W_j and $\mathsf{psid}_{W_k}^{d_{W_k}} | t_k | \mathsf{S}_k P | \sigma_{\mathsf{US}_k}$ from W_k;
14. Let $F_j = \mathsf{First}(\mathsf{psid}_{W_j}^{d_{W_j}})$ and $F_k = \mathsf{First}(\mathsf{psid}_{W_k}^{d_{W_k}})$;
15. $\Pi_V^{d_V}$ verifies
16. • F_j and F_k are subsets of $\mathsf{pid}_V^{d_V}$;
17. • t_j and t_k are the next expected message numbers from $\Pi_{W_j}^{d_{W_j}}$ and $\Pi_{W_k}^{d_{W_k}}$ respectively;
18. • σ_{US_j} is the multi-signature of F_j on $\mathsf{psid}_{W_j}^{d_{W_j}} | t_j | \mathsf{S}_j P$;
19. • σ_{US_k} is the multi-signature of F_k on $\mathsf{psid}_{W_k}^{d_{W_k}} | t_k | \mathsf{S}_k P$;
20. **if** any of the above verification fails, **then**
21. $\Pi_V^{d_V}$ sets $\mathsf{acc}_V^{d_V} = 0$, $\mathsf{sk}_V^{d_V} = \mathsf{NULL}$ and aborts;
22. **else**
23. $\Pi_V^{d_V}$ computes $H(e(\mathsf{S}_j P, \mathsf{S}_k P)^{\mathsf{S}_i})$;
24. $\Pi_V^{d_V}$ sets $\mathsf{psid}_V^{d_V} = \mathsf{psid}_V^{d_V} \cup \mathsf{psid}_{W_j}^{d_{W_j}} \cup \mathsf{psid}_{W_k}^{d_{W_k}}$;
25. **end if**;
26. **end for**;
27. **end for**;

end AuthCombineThree-B

The calls of CombineTwo and CombineThree from KeyAgreement are modified as follows:

Step 2 : change to **call** AuthCombineTwo($\mathsf{US}[i+1, i+2], \mathsf{S}[i+1, i+2]$);
Step 6 : change to **call** AuthCombineThree-A($\mathsf{US}[i+1, i+2, i+3], \mathsf{S}[i+1, i+2, i+3]$);
Step 19 : change to **call** AuthCombineThree-B($\widehat{\mathsf{US}}[1,2,3], \widehat{\mathsf{S}}[1,2,3]$);

The start of the protocol between instances $\Pi_{U_1}^{d_1}, \ldots, \Pi_{U_k}^{d_k}$ is made as follows:

start session($\Pi_{U_{i_1}}^{d_1}, \ldots, \Pi_{U_{i_k}}^{d_k}$)
1. **for** $j = 1$ to k
2. instance $\Pi_{U_{i_j}}^{d_j}$ chooses a secret key $s_j \in_R Z_q^*$ and sets $\mathsf{S}[j] = s_j$;
3. set $\mathsf{US}_j = \{\Pi_{U_{i_j}}^{d_j}\}$; $\mathsf{psid}_{U_{i_j}}^{d_j} = \{(U_{i_j}, d_j)\}$;
4. **end for**;
5. **call** KeyAgreement($k, \mathsf{US}[1, \ldots, k], \mathsf{S}[1, \ldots, k]$).

end session

The following property is easy to verify from the description of the protocol.

Proposition 1 *Suppose* US *is a set of instances occurring in* AuthCombineThree-B *in some session. Then for any two instances* $\Pi_{U_1}^{i_1}$ *and* $\Pi_{U_2}^{i_2}$ *participating in the session with* $\Pi_{U_1}^{i_1}, \Pi_{U_2}^{i_2} \in \mathsf{US}$*, we have* $\mathsf{psid}_{U_1}^{i_1} = \mathsf{psid}_{U_2}^{i_2}$. *Moreover, after the completion of a session in which* Π_U^i *participates, we have* $\mathsf{psid}_U^i = \mathsf{sid}_U^i$.

One consequence of the first statement of Proposition 1 is that in AuthCombineThree-B, each user in a user set computes the signature on the same message. Further, the second statement assures us that the partial session IDs finally "grow" into full session IDs.

Note that our protocol does not include key verification between users. This can be achieved at extra computational and communication cost.

4 Security Analysis

The goal is to show that the modification described in Section 3.2, converts the protocol of Section 3.1 into an authenticated key agreement protocol. Our proof technique is based on the proof technique used by Katz and Yung [15]. The idea behind the proof is the following. Assuming that DSig and MSig are secure, we can convert any adversary attacking the authenticated protocol into an adversary attacking the unauthenticated protocol. There are some technical differences between our proof and that of [15].

1. The Katz-Yung technique is a generic technique for converting *any* unauthenticated protocol into an authenticated protocol. On the other hand, we concentrate on one particular protocol. Hence we can avoid some of the complexities of the Katz-Yung proof.

2. Our protocol involves a multi-signature scheme whereas Katz-Yung requires only a signature scheme.

3. Katz-Yung protocol uses random nonces whereas our protocol does not.

4. In our unauthenticated protocol, there are no long term secret keys. Thus we can avoid the Corrupt oracle queries and can trivially achieve forward secrecy.

Theorem 2 *The protocol* AP *described in Section 3.2 satisfies the following:*

$$\mathsf{Adv}^{\mathsf{AKA}}_{\mathsf{AP}}(t, q_E, q_S) \leq \mathsf{Adv}^{\mathsf{KA}}_{\mathsf{UP}}(t', q_E + q_S/2) + |\mathcal{P}|\ \mathsf{Succ}_{\mathsf{DSig}}(t') + |\mathcal{P}|\ \mathsf{Succ}_{\mathsf{MSig}}(t')$$

where $t' \leq t + (|\mathcal{P}|q_E + q_S)t_{\mathsf{AP}}$*, where* t_{AP} *is the time required for execution of* AP *by any one of the users.*

Proof : Let $\mathcal{A}'$ be an adversary which attacks the authenticated protocol AP. Using this we construct an adversary $\mathcal{A}$ which attacks the unauthenticated protocol UP. We first have the following claim.
Claim : Let Forge be the event that a signature (either of DSig or of MSig) is forged by $\mathcal{A}'$. Then

$$\mathsf{Prob}[\mathsf{Forge}] \leq |\mathcal{P}|\ \mathsf{Succ}_{\mathsf{MSig}}(t') + |\mathcal{P}|\ \mathsf{Succ}_{\mathsf{DSig}}(t').$$

Proof of Claim: Let E_1 be the event that $\mathcal{A}'$ forges a multi-signature, i.e., it makes a query of the type $\mathsf{Send}(V, i, \mathsf{psid}^k_U|j|SP|\sigma)$ such that user V uses algorithm $\mathcal{MV}$ to verify σ to be the multi-signature of $\mathsf{First}(\mathsf{psid}^k_U)$ on $\mathsf{psid}^k_U|j|SP$ and this message-signature pair was not previously returned by $\mathcal{A}$.

Let E_2 be the event that $\mathcal{A}'$ makes a query of the type $\mathsf{Send}(V, i, Y)$ where Y has the form $Y = \Pi^{d_k}_{U_k}|1|SP|\sigma_{U_k}$ with $\mathcal{V}(\mathsf{pk}_{U_k}, \Pi^{d_k}_{U_k}|1|SP, \sigma_{U_k}) = 1$. Then clearly $\mathsf{Forge} = E_1 \vee E_2$.

First consider the event E_2. Using $\mathcal{A}'$, we construct an algorithm $\mathcal{F}$ that forges a signature for DSig as follows: Given a public key pk, algorithm $\mathcal{F}$ chooses a random $U \in \mathcal{P}$ and sets $pk_U = pk$. The other public keys and private keys for the system are generated honestly by $\mathcal{F}$. The forger $\mathcal{F}$ simulates all oracle queries of $\mathcal{A}'$ by executing protocol AP itself, obtaining the necessary signatures with respect to pk_U, as needed, from its signing oracle. Thus $\mathcal{F}$ provides a perfect simulation for $\mathcal{A}'$. If $\mathcal{A}'$ ever outputs a new valid message/signature pair with respect to $pk_U = pk$, then $\mathcal{F}$ outputs this pair as its forgery. The success probability of $\mathcal{F}$ is equal to $\frac{\mathsf{Prob}[E_2]}{|\mathcal{P}|}$ and hence $\mathsf{Prob}[E_2] \leq |\mathcal{P}|\ \mathsf{Succ}_{\mathsf{DSig}}(t')$.

Next consider the event E_1. Using $\mathcal{A}'$, we may construct an algorithm $\mathcal{F}$ that forges a multi-signature for the scheme MSig as follows: Given a public key pk_1, algorithm $\mathcal{F}$ chooses a random $U \in \mathcal{P}$, sets $pk_U = pk_1$ and honestly generates all other public/private keys for the system and outputs them. The forger $\mathcal{F}$ simulates all oracle queries of $\mathcal{A}'$ in the natural way by executing protocol AP itself, obtaining the necessary signatures with respect to pk_U, as needed, from its signing oracle and thus providing a perfect simulation for $\mathcal{A}'$. Now, if $\mathcal{A}'$ ever outputs a valid multi-signature σ on a message m for a set of users $L = \{U_{i_1}, \ldots, U_{i_k}\}$, which was not obtained from the signing oracle, then $\mathcal{F}$ outputs the message m, σ and L as its forgery. The success probability of $\mathcal{F}$ is equal to $\frac{\mathsf{Prob}[E_1]}{|\mathcal{P}|}$ and hence $\mathsf{Prob}[E_1] \leq |\mathcal{P}|\ \mathsf{Succ}_{\mathsf{MSig}}(t')$.

Then $\mathsf{Prob}[\mathsf{Forge}] = \mathsf{Prob}[E_1 \vee E_2] \leq \mathsf{Prob}[E_1] + \mathsf{Prob}[E_2] \leq |\mathcal{P}|\ \mathsf{Succ}_{\mathsf{MSig}}(t') + |\mathcal{P}|\ \mathsf{Succ}_{\mathsf{DSig}}(t')$, yielding the result of the Claim. ■(of Claim)

Now we describe the construction of the passive adversary $\mathcal{A}$ attacking UP that uses adversary $\mathcal{A}'$ attacking AP. Adversary $\mathcal{A}$ uses a list tlist. It stores pairs of session IDs and transcripts in tlist.

Adversary $\mathcal{A}$ generates the verification/signing keys pk_U, sk_U for each user $U \in \mathcal{P}$ and gives the verification keys to $\mathcal{A}'$. If ever the event Forge occurs, adversary $\mathcal{A}$ aborts and outputs a random bit. Otherwise, $\mathcal{A}$ outputs whatever bit is eventually output by $\mathcal{A}'$. Note that since the signing and verification keys are generated by $\mathcal{A}$, it can detect occurrence of the event Forge. $\mathcal{A}$ simulates the oracle queries of $\mathcal{A}'$ using its own queries to the Execute oracle. We provide details below.
Execute queries: Suppose $\mathcal{A}'$ makes a query $\mathsf{Execute}((U_{i_1}, d_1), \ldots, (U_{i_k}, d_k))$. This means that instances $\Pi^{d_1}_{U_{i_1}}, \ldots, \Pi^{d_k}_{U_{i_k}}$ are involved in this session. $\mathcal{A}$ defines $S = \{(U_{i_1}, d_1), \ldots, (U_{i_k}, d_k)\}$ and sends the execute query to its Execute oracle. It receives as output a transcript T of an execution of UP. It appends (S, T) to tlist. Adversary $\mathcal{A}$ then expands the transcript T for the unauthenticated protocol into a transcript T' for the authenticated protocol according to the modification described in Section 3.2. It returns T' to $\mathcal{A}'$.

Send queries: The first send query that $\mathcal{A}'$ makes to an instance is to start a new session. We will denote such queries by Send_0 queries. To start a session between unused instances $\Pi_{U_{i_1}}^{d_1}, \ldots, \Pi_{U_{i_k}}^{d_k}$, the adversary has to make the following send queries.

$$\begin{array}{c} \mathsf{Send}_0(U_{i_1}, d_1, \langle U_{i_2}, \ldots, U_{i_k}\rangle); \\ \mathsf{Send}_0(U_{i_2}, d_2, \langle U_{i_1}, U_{i_3} \ldots, U_{i_k}\rangle); \\ \vdots \\ \mathsf{Send}_0(U_{i_k}, d_k, \langle U_{i_1}, \ldots, U_{i_{k-1}}\rangle). \end{array}$$

Note that the above queries may be made in any order. When all the above queries have been made, $\mathcal{A}$ sets $S = \{(U_{i_1}, d_1), \ldots, (U_{i_k}, d_k)\}$ and makes an $\mathsf{Execute}$ query to its own execute oracle. It receives a transcript T in return and stores (S, T) in the list tlist.

Assuming that signatures (both DSig and MSig) cannot be forged, any subsequent Send query (i.e., after a Send_0 query) to an instance Π_U^i is a properly structured message with a valid signature. For any such Send query, $\mathcal{A}$ verifies the query according to the algorithm of Section 3.2. If the verification fails, $\mathcal{A}$ sets $\mathsf{acc}_U^i = 0$ and $\mathsf{sk}_U^i = \mathsf{NULL}$ and aborts Π_U^i. Otherwise, $\mathcal{A}$ performs the action to be done by Π_U^i in the authenticated protocol. This is done in the following manner: $\mathcal{A}$ first finds the unique entry (S, T) in tlist such that $(U, i) \in S$. From T, it finds the message which corresponds to the message sent by $\mathcal{A}'$ to Π_U^i. From the transcript T, adversary $\mathcal{A}$ finds the next public information to be output by Π_U^i. If this involves computation of a multi-signature and all the individual signatures have not yet been received by Π_U^i (using Send queries from $\mathcal{A}'$), then there is no output to this Send query. In all other cases, $\mathcal{A}$ returns the next public information to be output by Π_U^i to $\mathcal{A}'$.

Reveal/Test queries : Suppose $\mathcal{A}'$ makes the query $\mathsf{Reveal}(U, i)$ or $\mathsf{Test}(U, i)$ for an instance Π_U^i for which $\mathsf{acc}_U^i = 1$. At this point the transcript T' in which Π_U^i participates has already been defined. Now $\mathcal{A}$ finds the unique pair (S, T) in tlist such that $(U, i) \in S$. Assuming that the event Forge does not occur, T is the unique unauthenticated transcript which corresponds to the transcript T'. Then $\mathcal{A}$ makes the appropriate Reveal or Test query to one of the instances involved in T and returns the result to $\mathcal{A}'$.

As long as Forge does not occur, the above simulation for $\mathcal{A}'$ is perfect. Whenever Forge occurs, adversary $\mathcal{A}$ aborts and outputs a random bit. So $\mathsf{Prob}_{\mathcal{A}',\mathsf{AP}}[\mathsf{Succ}|\mathsf{Forge}] = \frac{1}{2}$. Now

$$\begin{aligned} \mathsf{Adv}_{\mathcal{A},\mathsf{UP}} &:= 2\,|\mathsf{Prob}_{\mathcal{A},\mathsf{UP}}[\mathsf{Succ}] - 1/2| \\ &= 2\,|\mathsf{Prob}_{\mathcal{A}',\mathsf{AP}}[\mathsf{Succ} \wedge \overline{\mathsf{Forge}}] + \mathsf{Prob}_{\mathcal{A}',\mathsf{AP}}[\mathsf{Succ} \wedge \mathsf{Forge}] - 1/2| \\ &= 2\,|\mathsf{Prob}_{\mathcal{A}',\mathsf{AP}}[\mathsf{Succ} \wedge \overline{\mathsf{Forge}}] + \mathsf{Prob}_{\mathcal{A}',\mathsf{AP}}[\mathsf{Succ}|\mathsf{Forge}]\ \mathsf{Prob}_{\mathcal{A}',\mathsf{AP}}[\mathsf{Forge}] - 1/2| \\ &= 2\,|\mathsf{Prob}_{\mathcal{A}',\mathsf{AP}}[\mathsf{Succ} \wedge \overline{\mathsf{Forge}}] + (1/2)\mathsf{Prob}_{\mathcal{A}',\mathsf{AP}}[\mathsf{Forge}] - 1/2| \\ &= 2\,|\mathsf{Prob}_{\mathcal{A}',\mathsf{AP}}[\mathsf{Succ}] - \mathsf{Prob}_{\mathcal{A}',\mathsf{AP}}[\mathsf{Succ} \wedge \mathsf{Forge}] + (1/2)\mathsf{Prob}_{\mathcal{A}',\mathsf{AP}}[\mathsf{Forge}] - 1/2| \\ &\geq |2\ \mathsf{Prob}_{\mathcal{A}',\mathsf{AP}}[\mathsf{Succ}] - 1| - |\mathsf{Prob}_{\mathcal{A}',\mathsf{AP}}[\mathsf{Forge}] - 2\ \mathsf{Prob}_{\mathcal{A}',\mathsf{AP}}[\mathsf{Succ} \wedge \mathsf{Forge}]| \\ &\geq \mathsf{Adv}_{\mathcal{A}',\mathsf{AP}} - \mathsf{Prob}[\mathsf{Forge}] \end{aligned}$$

The adversary $\mathcal{A}$ makes an $\mathsf{Execute}$ query for each $\mathsf{Execute}$ query of $\mathcal{A}'$. Also $\mathcal{A}$ makes an $\mathsf{Execute}$ query for each session started by $\mathcal{A}'$ using Send queries. Since a session involves at least two instances, such an $\mathsf{Execute}$ query is made after at least two Send queries of $\mathcal{A}'$. Hence the number of such $\mathsf{Execute}$ queries is at most $q_S/2$, where q_S is the number of Send queries made by $\mathcal{A}'$. Hence the total number of $\mathsf{Execute}$ queries made by $\mathcal{A}$ is at most $q_E + q_S/2$, where q_E is the number of $\mathsf{Execute}$ queries made by $\mathcal{A}'$. Also since $\mathsf{Adv}_{\mathcal{A},\mathsf{UP}} \leq \mathsf{Adv}_{\mathsf{UP}}^{\mathsf{KA}}(t', q_E + q_S/2)$ by assumption, we obtain:

$$\mathsf{Adv}_{\mathsf{AP}}^{\mathsf{AKA}} \leq \mathsf{Adv}_{\mathsf{UP}}^{\mathsf{KA}}(t', q_E + q_S/2) + \mathsf{Prob}[\mathsf{Forge}].$$

This yields the statement of the theorem. ∎

5 Efficiency

Efficiency of a protocol is measured by communication and computation cost. Communication cost involves counting total number of rounds needed and total number of messages transmitted through the network during a protocol execution. Computation cost counts total scalar multiplications, pairings, group exponentiations *etc.*.

Let Y be the total number of singleton user sets in level 1 and set $R(n) = \lceil \log_3 n \rceil$.

For the unauthenticated protocol [4], number of rounds required is $R(n)$ and total messages sent is $< \frac{5}{2}(n-1)$. The computation cost of this protocol is as follows :
total number of scalar multiplications in G_1 is $< \frac{5}{2}(n-1)$, total pairings required is $nR(n)$ and total group exponentiations in G_2 is $nR(n)$.

Note that in the authenticated protocol, the representative of a user set with more than one user creates a multi-signature after it collects the basic (BLS) signatures from the other users in that user set. This makes the representative to wait for accumulating the basic signatures. In the first round of the authenticated protocol, no multi-signature is required because each user set is a singleton set with a user itself being the representative and only a basic signature on the transmitted message is sent by the representative. The authenticated protocol additionally requires the followings :

1. The number of rounds increases by $R(n) - 1$.
2. Total number of basic (BLS) signatures computed is $nR(n)$.
3. Total number of additional messages (basic signatures) communicated is $n[R(n)-1] - \frac{3}{2}(3^{R(n)-1} - 1)$.
4. Total bilinear aggregate signatures computed, communicated and verified is $\frac{3}{2}(3^{R(n)-1} - 1) - Y$.
5. Total number of basic signatures verified is n.

This protocol involves no basic signature verification above level 1, only verification of multi-signatures are required. If any of the basic signature used in multi-signature creation on a message is improper, then this can be detected during multi-signature verification. This feature reduces the cost of verification for the authenticated protocol. Moreover, only representatives are required to have more computation power than other users. They are allowed to create and communicate multi-signatures which saves the total amount of communications.

6 Conclusion

We have described an authentication mechanism to authenticate the tree based key agreement protocol proposed by Barua, Dutta and Sarkar [4] in a standard fomalized security model. The bilinear pairing based multi-signature by Boldyreva [8] is used and the formal security model of Bresson *et al.* [9] is adopted. Using the proof technique of Katz and Yung [15], we get a provably secure authenticated tree based key agreement protocol.

References

[1] M. Abdalla, M. Bellare and P. Rogaway. *DHIES : An encryption scheme based on the Diffie-Hellman problem*, CT-RSA 2001 : 143-158.

[2] G. Ateniese, M. Steiner, and G. Tsudik. *New Multiparty Authenticated Services and Key Agreement Protocols*, Journal of Selected Areas in Communications, 18(4):1-13, IEEE, 2000.

[3] P. S. L. M. Barreto, H. Y. Kim and M. Scott. *Efficient algorithms for pairing-based cryptosystems.* Advances in Cryptology - Crypto '2002, LNCS 2442, Springer-Verlag (2002), pp. 354-368.

[4] R. Barua, R. Dutta, P. Sarkar. *Extending Joux Protocol to Multi Party Key Agreement.* Indocrypt 2003, Also available at http://eprint.iacr.org/2003/062.

[5] K. Becker and U. Wille. *Communication Complexity of Group Key Distribution.* ACMCCS '98.

[6] D. Boneh and M. Franklin. *Identity-Based Encryption from the Weil Pairing.* In Advances in Cryptology - CRYPTO '01, LNCS 2139, pages 213-229, Springer-Verlag, 2001.

[7] D. Boneh, B. Lynn, and H. Shacham. *Short Signature from Weil Pairing*, Proc. of Asiacrypt 2001, LNCS, Springer, pp. 213-229, 2001.

[8] A. Boldyreva. Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. Public Key Cryptography 2003: 31-46.

[9] E. Bresson, O. Chevassut, and D. Pointcheval. *Dynamic Group Diffie-Hellman Key Exchange under Standerd Assumptions.* Advances in Cryptology - Eurocrypt '02, LNCS 2332, L. Knudsen ed., Springer-Verlag, 2002, pp. 321-336.

[10] M. Burmester and Y. Desmedt. *A Secure and Efficient Conference Key Distribution System.* In A. De Santis, editor, Advances in Cryptology EUROCRYPT '94, Workshop on the theory and Application of Cryptographic Techniques, LNCS 950, pages 275-286, Springer-Verlag, 1995.

[11] W. Diffie and M. Hellman. *New Directions In Cryptography*, IEEE Transactions on Information Theory, IT-22(6) : 644-654, November 1976.

[12] S. Galbraith, K. Harrison and D. Soldera. *Implementing the Tate Pairing*, Algorithm Number Theory Symposium - ANTS V, LNCS 2369, Springer- Verlag (2002), pp. 324-337.

[13] I. Ingemarsson, D. T. Tang, and C. K. Wong. *A Conference Key Distribution System*, IEEE Transactions on Information Theory 28(5) : 714-720 (1982).

[14] A. Joux. *A One Round Protocol for Tripartite Diffie-Hellman*, ANTS IV, LNCS 1838, pp. 385-394, Springer-Verlag, 2000.

[15] J. Katz and M. Yung. *Scalable Protocols for Authenticated Group Key Exchange*, In Advances in Cryptology - CRYPTO 2003.

[16] Y. Kim, A. Perrig, and G. Tsudik. *Simple and Fault-tolerant Key Agreement for Dynamic Collaborative Groups.* In S. Jajodia, editor, 7th ACM Conference on Computation and Communication Security, pages 235-244, Athens, Greece, Nov. 2000, ACM press.

[17] Y. Kim, A. Perrig, and G. Tsudik. *Tree based Group Key Agreement.* Report 2002/009, http://eprint.iacr.org, 2002.

[18] M. Steiner, G. Tsudik, M. Waidner. *Diffie-Hellman Key Distribution Extended to Group Communication*, ACM Conference on Computation and Communication Security, 1996.