Efficient, Pairing-Free, Authenticated Identity Based Key Agreement in a Single Round

S. Sree Vivek[1], S. Sharmila Deva Selvi[1], Layamrudhaa Renganathan Venkatesan[1], and C. Pandu Rangan[1]

[1] Theoretical Computer Science Lab, Department of Computer Science and Engineering, Indian Institute of Technology Madras, Chennai, India
[2] National Institute of Technology Tiruchi, Tiruchirapalli, India

Abstract. Ever since Shamir introduced identity based cryptography in 1984, there has been a tremendous interest in designing efficient key agreement protocols in this paradigm. Since pairing is a costly operation and the composite order groups must be very large to ensure security, we focus on pairing free protocols in prime order groups. We propose a new protocol that is pairing free, working in prime order group and having tight reduction to Strong Diffie Hellman (SDH) problem under the CK model. Thus, the first major advantage is that smaller key sizes are sufficient to achieve comparable security. Our scheme has several other advantages. The major one being the capability to handle active adversaries. All the previous protocols can offer security only under passive adversaries. Our protocol recognizes the corruption by an active adversary and aborts the process. Achieving this in single round is significantly challenging. Ours is the first scheme achieving this property. In addition to this significant property, our scheme satisfies other security properties that are not covered by CK model such as forward secrecy, resistance to reflection, key compromise impersonation attacks and ephemeral key compromise impersonation attacks.

Keywords: Identity Based Key agreement, Provable Security, General forking lemma, Tight reduction, Random Oracle Model, Forward Secrecy, Reflection attacks, Key Compromise Impersonation attacks.

1 Introduction

Symmetric key cryptography is a system in which both encryption and decryption is performed using the same key unlike asymmetric system in which each user maintains a public key and a private key. Symmetric key cryptography is much easier to implement and demands less processing than asymmetric. But the main disadvantage with symmetric key cryptography is the establishment of the shared secret between the entities that want to communicate. A secure way of setting up the shared secret key is mandatory. The first key-agreement protocol was defined by Whitfield Diffie and Martin Hellman in 1976. This was based on the public key setting. Under this model, each user has to get his public key certified by a Certification Authority (CA), which is a trusted third party that issues certificates validating each user's public key. The idea was to use public key cryptography for key establishment and symmetric key cryptography for further communication using the shared secret key. But in this paradigm, the overhead associated with the CA will be high.

Identity Based Cryptography was introduced by Shamir [15] in 1984. In this infrastructure, each user's public key is his identity. A trusted party known as the Private Key Generator (PKG) maintains a master public key, master secret key pair. The master public key is known to everyone and the master secret key is known only to the PKG. It generates the private key of each user with the user's identity and master secret key. After the introduction of identity based cryptography, many schemes were proposed based on this model. In identity based key agreement protocol each user first obtains his private key from the PKG and engages in an interactive protocol with another user to establish a shared secret key. There is no need to transfer the public key certificates during the process. This is the main advantage of identity based system. Moreover there is a flexibility to use any string as the identity. It can be the user's email id, social security number, location and other attributes. The identity can also have a temporal value linked to it and hence the private key derived from it is invalid after a period of time. Hence identity based key agreement(IDKA) protocols are preferred rather than their public key based counterparts. An important factor to be considered with respect to key agreement is bandwidth requirement and number of rounds. Since IDKA eliminate the need to transfer public key verification certificates, they tend to reduce the bandwidth requirement. These are useful in situations where there is a constraint on the available bandwidth. Other properties like forward secrecy, resistance to man-in-the-middle attacks, reflection attacks and key compromise impersonation attacks should also be satisfied.

2 Previous Work and Our Contribution

After the discovery of identity based cryptography by Shamir [15], a number of key agreement protocols were developed in the identity based paradigm. But most of them involved pairing and hence their practical implementation was not efficient. Hence, we do not consider pairing based schemes for our comparison. The protocols which did not involve pairing were those of Gunther [7] and Saeednia [14]. Fiore [5] proposed a key agreement protocol without pairing which was an improvement over the protocols of Gunther [7] and Saeednia [14]. Cao [3] proposed a pairing free key agreement protocol but this was vulnerable to key-offset attack and known session specific secret information attack as presented in [8] by Islam. The protocol presented in [8] and [3] involves an initial agreement on who initiates the key agreement protocol. Therefore we do not consider [8] and [3] in the comparison. We will consider the works of Gunther [7], Saeednia [14] and Fiore [5] for comparison purpose.

The previous works on identity based key agreement do not consider an active adversary. An active adversary is one which can extract the messages that are exchanged during key agreement and modify them arbitrarily during transit. In the scheme presented in [5], the adversary can extract the ephemeral component g^{t_i} sent by user i to user j and modify it to g^{t_x} and send it to j. Similarly it can capture g^{t_j} sent from j to i and modify it to g^{t_x} and send to i. The component Z_2 calculated by i will be $(g^{t_x})^{t_i}$ and the one computed by j will be $(g^{t_x})^{t_j}$. Thus the final shared secret key of i and j will not be in agreement. Similar attacks are possible in Gunther [7] and Saeednia [14]. Our protocol avoids this kind of an attack by a signature on the ephemeral components. We compare computational power based on the number of exponentiation operations. We assume that the ephemeral components are chosen from a pre-computed list and hence we do not consider the cost of computing the ephemeral components. In our scheme, we use a Schnorr group and hence the exponentiation operations are cheaper than [5], [7], [14] even though it involves more exponentiation operations. This is because in a Schnorr group the exponent is from a group ${Z_p}^*$ where size of p is 224 bits according to *http://www.keylength.com/en/4/.* The additional security features like forward secrecy, resistance to reflection attacks, key compromise impersonation and ephemeral key compromise impersonation with respect to [5], [7], [14] are presented in Table 1.1 and Table 1.2.

Tightness of Security Reduction: We develop the security proof of the scheme as a game between a challenger and an adversary. If the adversary is able to break the scheme in polynomial time, then the adversary is said to succeed. Using the adversary's success, the challenger develops a solution to the underlying hard problem instance. Since there exists no solution to the hard problem that can be computed in polynomial time, the scheme cannot be broken and is considered secure.

The security parameter is set such that the adversary is not able to break the scheme in polynomial time through brute-force methods or sub-exponential algorithms. The relative hardness of breaking the scheme to that of breaking the computational assumption can be loose or tight. The use of forking lemma in security proofs makes the reduction inefficient by imposing an increase in the size of the modulus q. Hence, if we eliminate the use of forking lemma, the same level of security can be achieved with a smaller size modulus q. This contributes to the reduction in the number of bits required to realize the cryptographic primitive. Based on the work of Goh [6], in any discrete log based system, if the adversary can break the scheme in 2^n steps then forking lemma implies that the underlying discrete log problem can be solved in 2^{2n} steps. Here n is the security parameter. Therefore the scheme should be implemented in a group where the discrete log problem is believed secure with a security parameter of $2n$. In any discrete log system of a prime field Z_q, a factor α increase in the security parameter implies a factor α^3 increase in the size of the modulus q. Therefore if forking lemma is used in the security reduction, the security parameter increase by a factor of 2. Thus the size of the modulus increases by a factor of 8.

Our Contribution: In this paper, we present an identity based key agreement protocol which can be proved secure under the Strong-Diffie Hellman (SDH) assumption without using forking lemma. Thus we are able to achieve a tight reduction to the Strong Diffie Hellman problem based on the random oracle model. This tight reduction feature enables a reduction in the communication overhead thus making it efficient when compared to existing schemes. Moreover, our scheme is resistant to a dynamic active adversary which is allowed to modify the components exchanged during the key agreement. The scheme performs a check which will detect any tampering done on the components. In this way, a fully authenticated key agreement protocol is achieved. The protocol also satisfies additional security properties like forward secrecy, resistance to reflection attacks and key compromise impersonation attacks. But this level of security can be achieved with a smaller group size since our security proof does not involve the use of forking lemma and a tight reduction to SDH is possible. Table 1.1 and Table 1.2 compares our scheme for key lengths with the existing schemes that use forking lemma for the proof. Let $||\mathbb{G}||$ denote the number of bits needed to represent a group element. We have to set $||\mathbb{G}|| = 224$ for elliptic curve groups and $||\mathbb{G}|| = 1024$ for multiplicative groups as per $http://www.keylength.com/en/4/$.

Table 1. Comparison Table - Efficiency

Scheme	**No of Rounds**	**Tightness**	**Exp**	**Communication Cost-Elliptic Curve Group**	**Communication Cost Multiplicative Group**
Gunther [7]	2	Not tight	4	2*(8*$\|\|\mathbb{G}\|\|$)= 2*(8*224)=3584	2*(8*$\|\|\mathbb{G}\|\|$)= 2*(8*2048)=32768
Saeednia [14]	1	Not tight	3	1*(8*$\|\|\mathbb{G}\|\|$)= 1*(8*224)=1792	1*(8*$\|\|\mathbb{G}\|\|$)= 1*(8*2048)=16384
Fiore [5]	1	Not tight	2	1*(8*$\|\|\mathbb{G}\|\|$)= 1*(8*224)=1792	1*(8*$\|\|\mathbb{G}\|\|$)= 1*(8*2048)=16384
Ours	1	Tight	4	2*($\|\|\mathbb{G}\|\|$)+1*(224)= 2*224+1*224=672	2*($\|\|\mathbb{G}\|\|$)+1*(224)= 2*2048+1*224=4320

Remark 1 : Table 1 depict the resistance to the specified attacks. $\surd$ represents resistance and $\times$ represents vulnerability.

Remark 2 : The communication overhead and exponentiations are calculated for a single user.

Remark 3 : The key length is chosen based on the standard definition in *http://www.keylength.com/en/4/*. It states that to ensure security till the year 2030, the size of the elliptic curve group modulus should be 224 bits and 2048 bits for multiplicative groups. The size of the hash value is to be 224 bits for both elliptic curve and multiplicative groups.

Remark 4 : Generally the ephemeral components like t_i, w_i and g^{t_i}, g^{w_i} used during the protocol execution (see Table 2) are chosen from a pre-computed list. Hence they are not considered in the number of exponentiations.

Remark 5 : In our scheme, the computations of u_{j2}, v_{j2} by user i and u_{i2}, v_{i2} by user j in Step 2 of Table 2 are specific to a pair of users. So these computations are done only once for a pair of users and do not involve any session specific parameters. Hence these exponentiations are not included in computing the complexity of the protocol.

Remark 6 : Exponentiations of the form ${g_0}^{e_0}.{g_1}^{e_1}...{g_{k-1}}^{e_{k-1}}$ can be counted as a single exponentiation as in [12].

Remark 7 : When realized in elliptic curve groups, the first three schemes [7], [14] and [5] in Table 1.1 involve the specified number of exponentiation operations with exponent and base in a group with modulus size $8 * 224 \approx 2^{11}$ bits. Our scheme involves exponentiation operations where the exponent is from a group with modulus of 224 bits and base is from an elliptic curve group with modulus size = 224 bits. The complexity of an exponentiation x^y is given by $O\left({log_2}^2 x.log_2 y\right)$. So the cost of one exponentiation in [5], [7], [14] will be $2^{2*11+8} = 2^{30}$. Cost of exponentiation in our scheme will be $2^{2*8+8} = 2^{24}$.

Remark 8 : When realized in multiplicative groups, the first three schemes [7], [14] and [5] involve the specified number of exponentiation operations with exponent and base in a group with modulus size $8*2048=2^{14}$ bits. Our scheme involves exponentiation operations where the exponent is from a Schnorr group with modulus of 224 bits and base is from a group with modulus size = 2048 bits. The complexity of an exponentiation x^y is given by $O\left({log_2}^2 x.log_2 y\right)$. So cost of one exponentiation in [5], [7], [14] will be $2^{2*14+14} = 2^{42}$. Cost of exponentiation in our scheme will be $2^{2*11+8} = 2^{30}$.

Table 2. Comparison with the existing schemes - Security

Scheme	**Forward Secrecy**	**Reflection Attacks**	**KCI**	**Ephemeral KCI**	**Dynamic adversary**
Gunther [7]	√	×	√	×	×
Saeednia [14]	√	√	√	×	×
Fiore [5]	√	√	√	×	×
Ours	√	√	√	√	√

The PKI-based MQV [10] protocol involves sending 2 group elements and 1.5 exponentiation operations to compute the shared secret key. But certificates need to be sent in this system. We do not consider RSA signatures for certificates because RSA uses composite modulus. If we take into account Schnorr signature for certification purpose, we will have 1 more group element and a Schnorr group element to be sent and the signing and verification process will involve 3 more exponentiation operations. But these exponentiations will have exponent in a Schnorr group which is realized on elliptic curves. Hence the total number of exponentiation operations will be 1.5 with exponent size 8*224 and 3 with exponent size 224. Moreover this scheme does not achieve tight reduction unlike our protocol. Thus our scheme is better than the PKI based scheme with certificates.

3 Identity Based Key Agreement

In this section, we will give the definition of an identity based key agreement protocol and the description of the security model.

3.1 Definition of Identity Based Key Agreement Protocol

In Identity Based Key Agreement protocol, each entity i is defined by a unique identity, ID_i. The PKG maintains master public key and master secret key and generates the private key S_i for each user. The protocol is defined as follows:

Setup: The PKG chooses the public parameters and the master secret key. The public parameters are open to all users and the master secret key is known only to the PKG.

Key Generation: The user i submits its identity ID_i to the PKG and the PKG constructs the private key S_i for the user with identity ID_i.

Key Agreement: In order to establish the shared secret key between two users A and B with identities ID_A and ID_B and secret keys S_A and S_B, the users engage in a session by exchanging components and eventually set up the shared secret key. Either user A or B could initiate the protocol.

3.2 Definition of the Security Model

The security of our identity based key agreement protocol is analyzed based on the Canetti-Krawczyk (CK) model for key agreement [2]. CK model does not cover forward secrecy, resistance to reflection and key compromise impersonation attacks. We provide these additional security features. Now we define certain terms associated with identity based key agreement and formally define the security model.

An instance of the protocol defined in Section 3.1 is called a *session*. The user or entity that initiates a session is called the *owner* and the other user is called the *peer*. The components exchanged between the owner and the peer constitute the *session state*. The shared secret key obtained is called the *session key*. On successful completion of a session, each entity outputs the *session key* and deletes the *session state*. Otherwise, the session is said to be in *abort* state and no *session key* is generated in this case. Each entity participating in a session assigns a unique identifier to that session. For example, A sets the unique identifier as (A, B, out, in) where B is its peer and out and in are respectively the components sent to B and received by A. If B holds a session (B, A, in, out), then both the sessions are said to be *matching sessions*. There are three types of adversary:

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

Since we prove our scheme secure against the Type III adversary, it is also secure against Type I and Type II because they are weaker adversaries compared to Type III. We allow the adversary to access some of the parties secret information, via the following attacks: party corruption, state-reveal queries and session-key queries. In party corruption phase, the adversary learns the private keys of the users. In a state-reveal query to a party running a session, the adversary learns the session state for that session. In shared secret key query phase, the adversary learns the shared secret key of a complete session. A session is called *exposed* if it or its *matching session* (if existing) is compromised by one of the attacks described above. The security model of the identity based key agreement is modeled as a following game between the challenger and the adversary:

Setup: The challenger sets up the public parameters and the master secret key. The public parameters are made known to the adversary whereas the master secret key is kept private with the challenger.

Party corruption: In this phase, the adversary can query the challenger for the private key of any user with identity ID_i. The challenger has to compute the private key S_i corresponding to ID_i and return the response to the adversary.

Session Simulation: In this phase, the adversary is allowed to ask the shared secret key queries. The adversary queries for a shared secret belonging to a session established between two users A and B. The adversary can also emulate as one of the users, either A or B and present the challenger with the session state corresponding to that user. The challenger has to generate the session state for the other user of the session and obtain the shared secret key corresponding to that session. The adversary can also query for the session secret key between the two parties A and B from the challenger, where the adversary does not impersonate any of the user. In this case the challenger has to generate the session state for both the users and obtain the shared secret key corresponding to that session and provide it to the adversary.

Test Session: The adversary chooses a test session among all the completed and unexposed sessions. The challenger will toss a random bit $b \in_R \{0, 1\}$. If

$b = 0$ the challenger will give the adversary the session key K_0 of the test session. Otherwise the challenger will take a random shared secret key K_1 and provide the adversary with K_1.

Guess: The adversary makes a guess δ as to which key K_0 or K_1 was given by the challenger. The adversary wins if $\delta = b$.

The identity based key agreement protocol is said to be secure if no polynomial-time adversary has non-negligible advantage in winning the above game, i.e., distinguishing K_0 from K_1.

Note: The Send query present in [9] is not required here since our protocol is single round and it is a 2-party protocol. The adversary has access to the components exchanged and can modify them as per its wish.

4 Preliminaries

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

Definition 1. ***Computation Diffie-Hellman Problem (CDHP) -*** *Given* $(g, g^a, g^b) \in \mathbb{G}^3$ *for unknown* $a, b \in \mathbb{Z}_q^*$*, where* $\mathbb{G}$ *is a cyclic prime order multiplicative group with* g *as a generator and* q *the order of the group, the CDH problem in* $\mathbb{G}$ *is to compute* g^{ab}*.*

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

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

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

Definition 2. ***Decisional Diffie-Hellman Problem (DDHP) -*** *Given* $(g, g^a, g^b, h) \in \mathbb{G}^4$ *for unknown* $a, b \in \mathbb{Z}_q^*$*, where* $\mathbb{G}$ *is a cyclic prime order multiplicative group with* g *as a generator and* q *the order of the group, the DDH problem in* $\mathbb{G}$ *is to check whether* $h \stackrel{?}{=} g^{ab}$*.*

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

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

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

Definition 3. ***(Strong Diffie Hellman Problem (SDHP)*** *[1]): Let* κ *be the security parameter and* $\mathbb{G}$ *be a multiplicative group of order* q*, where* $|q| = \kappa$*. Given* $(g, g^a, g^b) \in_R \mathbb{G}^3$ *and access to a Decision Diffie Hellman (DDH) oracle* $\mathcal{DDH}_{g,a}(.,.)$ *which on input* g^b *and* g^c *outputs* `True` *if and only if* $g^{ab} = g^c$*, the strong Diffie Hellman problem is to compute* $g^{ab} \in \mathbb{G}$*.*

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

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

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

Note: In pairing groups (also known as gap groups), the DDH oracle can be efficiently instantiated and hence the strong Diffie Hellman problem is equivalent to the Gap Diffie Hellman problem [13].

5 The Proposed Identity Based Key Agreement Protocol

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

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

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

Key Extract: An user i with identity ID_i submits its identity to the PKG. The PKG does the following to generate the private key of the user i.

- The PKG chooses $x_i \in_R {Z_p}^*$.
- It computes $u_{i1} = g^{x_i}$ and sets $h_i = H_1(ID_i)$.
- It computes $v_{i1} = {h_i}^{x_i}$.
- It chooses $r_i \in_R {Z_p}^*$, computes $u_{i2} = g^{r_i}$ and $v_{i2} = {h_i}^{r_i}$.
- It sets $c_i = H_2(ID_i, u_{i1})$, $b_i = H_3(ID_i, u_{i1}, v_{i1}, u_{i2}, v_{i2})$ and $e_i = H_4(ID_i, u_{i1}, v_{i1}, u_{i2}, v_{i2})$.
- It computes $d_{i1} = x_i + s_1 c_i$ where s_1 is the master secret key. It also calculates $d_{i2} = x_i + r_i b_i + s_2 e_i$.
- Finally it sends $\langle u_{i1}, v_{i1}, u_{i2}, v_{i2}, d_{i1}, d_{i2}, {h_i}^{s_2} \rangle$ to the user i.

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

Key Agreement: The two users i and j with identities ID_i and ID_j get their respective private keys from the PKG and choose ephemeral secret components $t_i, w_i \in_R {Z_p}^*$ and $t_j, w_j \in_R {Z_p}^*$ respectively and engage in a session as described in Table 2.

Table 3. Description of the Key Agreement protocol

User i	**User j**
1. Send $F_i = \langle u_{i1}, v_{i1}, d_{i2}, b_i, e_i, {h_i}^{s_2}, ID_i \rangle$, $V_i = \langle w_i + d_{i1}.H_5\left(g^{t_i}, g^{w_i}\right), g^{t_i}, g^{w_i} \rangle$ to j.	1. Send $F_j = \langle u_{j1}, v_{j1}, d_{j2}, b_j, e_j, {h_j}^{s_2}, ID_j \rangle$, $V_j = \langle w_j + d_{j1}.H_5\left(g^{t_j}, g^{w_j}\right), g^{t_j}, g^{w_j} \rangle$ to i.
2. (a) ***Check for correctness of*** F_j***:*** Compute $u_{j2} = \left(\frac{g^{d_{j2}}}{u_{j1} \cdot {y_2}^{e_j}}\right)^{{b_j}^{-1}}$ Compute $v_{j2} = \left(\frac{{h_j}^{d_{j2}}}{v_{j1} \cdot \left({h_j}^{s_2}\right)^{e_j}}\right)^{{b_j}^{-1}}$ ***Check 1 :*** Check if $b_j \overset{?}{=} H_3(ID_j, u_{j1}, v_{j1}, u_{j2}, v_{j2})$ $e_j \overset{?}{=} H_4(ID_j, u_{j1}, v_{j1}, u_{j2}, v_{j2})$ If not equal abort, else proceed. (b) ***Check for correctness of*** V_j***:*** ***Check 2 :*** Check if $\left[\frac{g^{\left(w_j + d_{j1} \cdot H_5\left(g^{t_j}, g^{w_j}\right)\right)}}{\left(g^{x_j}\right)^{H_5\left(g^{t_j}, g^{w_j}\right)} \left(y_1\right)^{c_j \cdot H_5\left(g^{t_j}, g^{w_j}\right)}}\right] \overset{?}{=} g^{w_j}$ where $c_j = H_2\left(ID_j, u_{j1}\right)$. If equal proceed to step 3, else abort.	2. (a) ***Check for correctness of*** F_i***:*** Compute $u_{i2} = \left(\frac{g^{d_{i2}}}{u_{i1} \cdot {y_2}^{e_i}}\right)^{{b_i}^{-1}}$ Compute $v_{i2} = \left(\frac{{h_i}^{d_{i2}}}{v_{i1} \cdot \left({h_i}^{s_2}\right)^{e_i}}\right)^{{b_i}^{-1}}$ ***Check 1 :*** Check if $b_i \overset{?}{=} H_3\left(ID_i, u_{i1}, v_{i1}, u_{i2}, v_{i2}\right)$ $e_i \overset{?}{=} H_4\left(ID_i, u_{i1}, v_{i1}, u_{i2}, v_{i2}\right)$ If not equal abort, else proceed. (b) ***Check for correctness of*** V_i***:*** ***Check 2 :*** Check if $\left[\frac{g^{\left(w_i + d_{i1} \cdot H_5\left(g^{t_i}, g^{w_i}\right)\right)}}{\left(g^{x_i}\right)^{H_5\left(g^{t_i}, g^{w_i}\right)} \left(y_1\right)^{c_i \cdot H_5\left(g^{t_i}, g^{w_i}\right)}}\right] \overset{?}{=} g^{w_i}$ where $c_i = H_2\left(ID_i, u_{i1}\right)$. If equal proceed to step 3, else abort.
3. ***Shared secret key generation:*** Compute $Z_1 = \left(u_{j1} {y_1}^{c_j} g^{t_j}\right)^{d_{i1} + t_i}$ $Z_2 = v_{i1} v_{j1}$ $Z_3 = \left(g^{t_j}\right)^{t_i}$. $Z = H_6\left(Z_1, Z_2, Z_3\right)$.	3. ***Shared secret key generation:*** Compute $Z_1 = \left(u_{i1} {y_1}^{c_i} g^{t_i}\right)^{d_{j1} + t_j}$ $Z_2 = v_{j1} v_{i1}$ $Z_3 = \left(g^{t_i}\right)^{t_j}$. $Z = H_6\left(Z_1, Z_2, Z_3\right)$.

Z is the shared secret key that is established between User i and User j.

Remark 9 : The protocol is asynchronous and consists of only one send per user per session. Hence the data transfer can occur in any order.

Remark 10 : The values in F_i are same for all sessions between a pair of users and is independent of the session.

Remark 11 : The values in V_i are freshly generated for every session in the following manner. In a preprocessing or a setup stage, the user i generates a large number of $\left(\beta, g^{\beta}\right)$ pairs and stores them in a table T_i. For each session, user i extracts two fresh pairs from the table T_i and uses them to generate components of V_i. For security reasons, we assume that

(a) immediately after generating the components of V_i, w_i is erased from the system.
(b) $w_i + d_{i1}.H_5\left(g^{t_i}, g^{w_i}\right)$ is computed in a secured way so that w_i and d_{i1} are not leaked to the adversary and only $w_i + d_{i1}.H_5\left(g^{t_i}, g^{w_i}\right)$ is available to the adversary.

Remark 12 : The components in F_i, V_i and F_j, V_j is required to be sent only for the first time key establishment between users i and j. For subsequent key establishments between i and j, only V_i and V_j need to be exchanged. So we have considered only the components of V_i in communication overhead in Table 1.1. The exponentiations done in Check for correctness of components of F_i is one time and hence it is not included in computation cost. We include only the exponentiations in check for correctness of components in V_i and shared secret key generation in Table 1.1. We have to discuss the size of F_i and V_i in the cases of multiplicative groups and elliptic curve groups. The sizes in multiplicative group of order p which has a subgroup of order q, $|p|$ denoting the number of bits in p, $|q|$ referring to the number of bits in q and $|ID_i|$ denoting the length of the identity of user i are $F_i = 3.|q| + 3.|p| + |ID_i|$ and $V_i = 1.|q| + 2.|p|$. In the case of elliptic curve of order p, the sizes are $F_i = 6.|p| + |ID_i|$ and $V_i = 3.|p|$

Remark 13 : The intuition behind using the component Z_3 is to eliminate $g^{t_i \cdot t_j}$ from Z_1 in the security proof to obtain the solution to the hard problem.

Remark 14 : ***Check 1*** is done to ensure that g and h_i are raised to the same exponent x_i. This is a crucial security requirement.

For valid components this check holds good. We prove it here.

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

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

The components that are recomputed are valid and hence the computation of $b_i = H_3\left(ID_i, u_{i1}, v_{i1}, u_{i2}, v_{i2}\right)$ will match the one obtained if not for any tampering during transfer.

Remark 15 : ***Check 2*** is done to ensure that a dynamic adversary cannot tamper the components exchanged and affect the shared secret key generation. It verifies the signature $w_i + d_{i1}.H_5\left(g^{t_i}, g^{w_i}\right)$ on g^{t_i}.

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

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

Proof: User i computes :
$Z_1 = \left(u_{j1} {y_1}^{c_j} g^{t_j}\right)^{d_{i1} + t_i} = \left(g^{(x_j + s_1 c_j + t_j)}\right)^{(d_{i1} + t_i)} = g^{(d_{j1} + t_j)(d_{i1} + t_i)}$, since $u_{j1} = g^{x_j}$ and $x_j + s_1 c_j = d_{j1}$.
User j computes:
$Z_1 = \left(u_{i1} {y_1}^{c_i} g^{t_i}\right)^{d_{j1} + t_j} = \left(g^{(x_i + s_1 c_i + t_i)}\right)^{(d_{j1} + t_j)} = g^{(d_{i1} + t_i)(d_{j1} + t_j)}$, since $u_{i1} = g^{x_i}$ and $x_i + s_1 c_i = d_{i1}$.

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

6 Security Proof

In this section, we give the security proof of the scheme presented in the previous section. The proof is modeled based on the CK-model. The scheme is proved secure in the random oracle model. The scheme is reduced to the Strong Diffie-Hellman (SDH) problem. Since the proof technique eliminates the use of forking lemma, we are able to achieve a tight reduction to the underlying hard problem. The security proof is modeled as a game between the challenger and the adversary.

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

$H_1 Oracle$: The challenger is queried by the adversary for the hash value of the identity ID_i. If the H_1 $Oracle$ was already queried with ID_i as input, the challenger returns the value computed before which is stored in the hash list L_{h1} described below. Otherwise the challenger tosses a *coin* τ_i where the $Pr(\tau_i = 0) = \alpha$. The output of this oracle is defined as:

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

where $k_i \in_R Z_p{}^*$. The challenger makes an entry in the hash list $L_{h1} = \langle h_i, ID_i, \tau_i, k_i \rangle$ for future use and returns h_i.

H_2 $Oracle$: The adversary queries the challenger with inputs (ID_i, u_{i1}). If the H_2 $Oracle$ was already queried with (ID_i, u_{i1}) as input, the challenger extracts the value c_i from the hash list L_{h2} described below and returns the value. Otherwise, the challenger chooses a random value $c_i \in_R Z_p{}^*$. It makes an entry in the hash list $L_{h2} = \langle c_i, u_{i1}, ID_i \rangle$ and returns c_i.

H_3 $Oracle$: The adversary queries the challenger with inputs $(ID_i, u_{i1}, v_{i1}, u_{i2}, v_{i2})$. If the H_3 $Oracle$ was already queried with $(ID_i, u_{i1}, v_{i1}, u_{i2}, v_{i2})$ as input, the challenger extracts the value b_i from the hash list L_{h3} described below and returns the value. Otherwise, the challenger chooses a random value $b_i \in_R Z_p{}^*$. It makes an entry in the hash list $L_{h3} = \langle b_i, ID_i, u_{i1}, v_{i1}, u_{i2}, v_{i2} \rangle$ and returns b_i.

H_4 $Oracle$: The adversary queries the challenger with inputs $(ID_i, u_{i1}, v_{i1}, u_{i2}, v_{i2})$. If the H_4 $Oracle$ was already queried with $(ID_i, u_{i1}, v_{i1}, u_{i2}, v_{i2})$ as input, the challenger extracts the value e_i from the hash list L_{h4} described below and returns the value. Otherwise, the challenger chooses a random value $e_i \in_R Z_p{}^*$. It makes an entry in the hash list $L_{h4} = \langle e_i, ID_i, u_{i1}, v_{i1}, u_{i2}, v_{i2} \rangle$ and returns e_i.

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

H_6 *Oracle* : The adversary queries the challenger with inputs (Z_1, Z_2, Z_3). If the H_4 *Oracle* was already queried with (Z_1, Z_2, Z_3) as input, the challenger extracts the value l_i from the hash list L_{h6} described below and returns the value. Otherwise, the challenger chooses a random value $l_i \in_R Z_p{}^*$. It makes an entry in the hash list $L_{h6} = \langle l_i, Z_1, Z_2, Z_3 \rangle$ and returns l_i.

Party corruption: The adversary presents the challenger with an identity ID_i and the challenger should return the private key of that entity. The challenger proceeds in the following way:

The challenger checks if the H_1 *Oracle* was already queried for ID_i. If yes and the corresponding $\tau_i = 1$, it *aborts*. Otherwise it extracts k_i, h_i from the list L_{h1} and proceeds to the next step. If ID_i was not queried before, the challenger runs the H_1 *Oracle* with ID_i as input. If $\tau_i = 1$, it *aborts*. Else the challenger chooses $k_i \in_R Z_p^*$, computes $h_i = g^{k_i}$, adds the tuple $\langle h_i, ID_i, \tau_i, k_i \rangle$ to the L_{h1} list.

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

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

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

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

- ***Test 1 :*** Check if $\frac{g^{d_{i1}}}{y_1^{H_2(ID_i, u_{i1})}} \overset{?}{=} u_{i1}$.

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

- ***Test 2 :*** Check if $\frac{g^{d_{i2}}}{{u_{i2}}^{H_3(ID_i,u_{i1},v_{i1},u_{i2},v_{i2})}.{y_2}^{H_4(ID_i,u_{i1},v_{i1},u_{i2},v_{i2})}} \overset{?}{=} u_{i1}$.

  This can be verified as $\frac{g^{x'_i+r'_i b_i+s_2 e_i}}{\left(g^{r'_i}.{y_1}^{c_i \cdot {b_i}^{-1}}\right)^{b_i}.g^{s_2 \cdot e_i}} = g^{x'_i - a.c_i} = g^{x'_i}.{y_1}^{-c_i} = u_{i1}$, as $b_i = H_3\left(ID_i, u_{i1}, v_{i1}, u_{i2}, v_{i2}\right)$ and $e_i = H_4\left(ID_i, u_{i1}, v_{i1}, u_{i2}, v_{i2}\right)$.
- ***Test 3 :*** Check if $\frac{h_i^{d_{i2}}}{{v_{i2}}^{H_3(ID_i,u_{i1},v_{i1},u_{i2},v_{i2})}.({h_i}^{s_2})^{H_4(ID_i,u_{i1},v_{i1},u_{i2},v_{i2})}} \overset{?}{=} v_{i1}$.

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

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

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

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

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

Case 1b: If $\tau_j = 1$, the challenger does not know the secret key corresponding to j and hence simulates the session key as follows:

1. The challenger first performs the check presented in the Step 2 of the Key Agreement protocol, on $\langle u_{i1}, v_{i1}, d_{i2}, b_i, e_i, {h_i}^{s_2}, g^{t_i}, w_i + d_{i1}.H_5\left(g^{t_i}, g^{w_i}\right), g^{w_i}, ID_i\rangle$.
2. The challenger generates the parameters $\langle u_{j1} = g^{x_j}, v_{j1} = {h_j}^{x_j}, d_{j2} = x_j + r_j.b_j + s_2.e_j, b_j, e_j, {h_j}^{s_2}, g^{t_j}, w'_j + x_j.f_j, g^{w'_j}.{y_1}^{-c_j \cdot f_j}, ID_j\rangle$, where $r_j, x_j, t_j, w'_j, f_j \in_R {Z_p}^*$, $h_j = H_1\left(ID_j\right)$, $b_j = H_3\left(ID_j, u_{j1}, v_{j1}, g^{r_j}, {h_j}^{r_j}\right)$ and $e_j = H_4\left(ID_j, u_{j1}, v_{j1}, g^{r_j}, {h_j}^{r_j}\right)$.
3. If H_5 was already queried with inputs $\left(g^{t_j}, g^{w'_j}.{y_1}^{-c_j \cdot f_j}\right)$, generate a fresh w'_j and recompute the last but two components. With very high probability, the new $\left(g^{t_j}, g^{w'_j}.{y_1}^{-c_j \cdot f_j}\right)$ will not result in a previously queried input set to H_5. Set $H_5\left(g^{t_j}, g^{w'_j}.{y_1}^{-c_j \cdot f_j}\right)$ as f_j.
4. The parameters generated by the challenger, $\langle u_{j1}, v_{j1}, d_{j2}, b_j, e_j, {h_j}^{s_2}\rangle$ will satisfy ***Check 1*** in Step 2 of Key Agreement. This is because the parameters $\langle u_{j1}, v_{j1}, d_{j2}, b_j, e_j, {h_j}^{s_2}\rangle$ are generated in the same way as the original scheme.

5. The parameters $\langle u_{j1}, v_{j1}, d_{j2}, b_j, e_j, {h_j}^{s_2} \rangle$ also satisfy ***Check 2*** in the Step 2 of Key Agreement of Section 5.
$$\frac{g^{w'_j + x_j \cdot f_j}}{(g^{x_j})^{H_5\left(g^{t_j}, g^{w'_j}.{y_1}^{-c_j \cdot f_j}\right)} . (y_1)^{c_j . H_5\left(g^{t_j}, g^{w'_j}.{y_1}^{-c_j \cdot f_j}\right)}} = g^{w'_j}.{y_1}^{-c_j \cdot f_j} = g^{w_j}.$$
6. Thus the parameters generated by the challenger, $\langle u_{j1}, v_{j1}, d_{j2}, b_j, e_j, {h_j}^{s_2} \rangle$ are consistent with that of the system.
7. The challenger sends the parameters to the adversary.
8. The challenger computes $\bar{Z}_1 = (g^{x_i}.{y_1}^{c_i}.g^{t_i})^{x_j + t_j}$ where $c_i = H_2(ID_i, u_{i1})$. It also computes $P_1 = (u_{i1}.{y_1}^{c_i}.g^{t_i})^{c_j}$ and $P_2 = y_1$ where $c_j = H_2(ID_j, u_{j1})$.
9. The challenger computes $Z_2 = v_{i1}.v_{j1}$ and $Z_3 = (g^{t_i})^{t_j}$.
10. The challenger is given access to the $DH(y_1, ., .)$ oracle, since we assume the hardness of Strong-Diffie Hellman problem. The challenger makes use of the $DH(y_1, ., .)$ Oracle to answer the query as follows:
    - The challenger finds a Z such that $DH\left(P_2, P_1, Z_1/\bar{Z}_1\right)$ (valid since $P_2 = y_1$) and $H_6(Z_1, Z_2, Z_3) = Z$, where $Z_2 = v_{i1}.v_{j1}$ and $Z_3 = (g^{t_i})^{t_j}$.
    - If a Z exists, the challenger returns Z as the shared secret key.
    - Otherwise the challenger chooses $Z \in_R {Z_p}^*$ and for any further query of the form (Z_1, Z_2, Z_3) to the H_6 *Oracle*, if $DH\left(P_2, P_1, Z_1/\bar{Z}_1\right)$, $Z_2 = v_{i1}.v_{j1}$ and $Z_3 = (g^{t_i})^{t_j}$, the challenger returns Z as the result to the query.

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

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

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

1. The challenger generates the values $\langle u_{j1} = g^{x_j}, v_{j1} = {h_j}^{x_j}, d_{j2} = x_j + r_j.b_j + s_2.e_j, b_j, e_j, {h_j}^{s_2}, g^{t_j}, w'_j + x_j.f_j, g^{w'_j}.{y_1}^{-c_j \cdot f_j}, ID_j \rangle$, where $r_j, x_j, t_j, w'_j, f_j \in_R {Z_p}^*$, $h_j = H_1(ID_j)$, $b_j = H_3(ID_j, u_{j1}, v_{j1}, g^{r_j}, {h_j}^{r_j})$ and $e_j = H_4(ID_j, u_{j1}, v_{j1}, g^{r_j}, {h_j}^{r_j})$ for user j.
2. The challenger also generates the values $\langle u_{i1} = g^{x_i}, v_{i1} = {h_i}^{x_i}, d_{i2} = x_i + r_i.b_i + s_2.e_i, b_i, e_i, {h_i}^{s_2}, g^{t_i}, w'_i + x_i.f_i, g^{w'_i}.{y_1}^{-c_i \cdot f_i}, ID_i \rangle$ with i's private key for user i.
3. If H_5 was already queried with inputs $\left(g^{t_j}, g^{w'_j}.{y_1}^{-c_j \cdot f_j}\right)$, generate a fresh w'_j and recompute the last but two components. With very high probability,

the new $\left(g^{t_j}, g^{w'_j}.{y_1}^{-c_j \cdot f_j}\right)$ will not result in a previously queried input set to H_5. Set $H_5\left(g^{t_j}, g^{w'_j}.{y_1}^{-c_j \cdot f_j}\right)$ as f_j.

4. Similarly if H_5 was already queried with inputs $\left(g^{t_i}, g^{w'_i}.{y_1}^{-c_i \cdot f_i}\right)$, generate a fresh w'_i and recompute the last but two components. With very high probability, the new $\left(g^{t_i}, g^{w'_i}.{y_1}^{-c_i \cdot f_i}\right)$ will not result in a previously queried input set to H_5. Set $H_5\left(g^{t_i}, g^{w'_i}.{y_1}^{-c_i \cdot f_i}\right)$ as f_i.
5. The challenger computes $\bar{Z}_1 = (g^{x_i}.{y_1}^{c_i}.g^{t_i})^{x_j+t_j}$ where $c_i = H_2(ID_i, u_{i1})$. It also computes $P_1 = (u_{i1}.{y_1}^{c_i}.g^{t_i})^{c_j}$ and $P_2 = y_1$ where $c_j = H_2(ID_j, u_{j1})$.
6. The challenger computes $Z_2 = v_{i1}.v_{j1}$ and $Z_3 = (g^{t_i})^{t_j}$.
7. The challenger is given access to the $DH(y_1, ., .)$ oracle, since we assume the hardness of Strong-Diffie Hellman problem. The challenger makes use of the $DH(y_1, ., .)$ Oracle to answer the query as follows:
   - The challenger finds a Z such that $DH\left(P_2, P_1, Z_1/\bar{Z}_1\right)$ (valid since $P_2 = y_1$) and $H_6(Z_1, Z_2, Z_3) = Z$, where $Z_2 = v_{i1}.v_{j1}$ and $Z_3 = (g^{t_i})^{t_j}$.
   - If a Z exists, the challenger returns Z as the shared secret key.
   - Otherwise the challenger chooses $Z \in_R {Z_p}^*$ and for any further query of the form (Z_1, Z_2, Z_3) to the H_6 *Oracle*, if $DH\left(P_2, P_1, Z_1/\bar{Z}_1\right)$, $Z_2 = v_{i1}.v_{j1}$ and $Z_3 = (g^{t_i})^{t_j}$.

Finally the challenger returns Z as the shared secret key.

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

1. The challenger generates the values $\langle u_{j1} = g^{x_j}, v_{j1} = {h_j}^{x_j}, d_{j2} = x_j + r_j.b_j + s_2.e_j, b_j, e_j, {h_j}^{s_2}, g^{t_j}, w'_j + x_j.f_j, g^{w'_j}.{y_1}^{-c_j \cdot f_j}, ID_j\rangle$, where $r_j, x_j, t_j, w'_j, f_j \in_R {Z_p}^*$, $h_j = H_1(ID_j)$, $b_j = H_3(ID_j, u_{j1}, v_{j1}, g^{r_j}, {h_j}^{r_j})$ and $e_j = H_4(ID_j, u_{j1}, v_{j1}, g^{r_j}, {h_j}^{r_j})$ for user j.
2. The challenger also generates the values $\langle u_{i1} = g^{x_i}, v_{i1} = {h_i}^{x_i}, d_{i2} = x_i + r_i.b_i + s_2.e_i, b_i, e_i, {h_i}^{s_2}, g^{t_i}, w'_i + x_i.f_i, g^{w'_i}.{y_1}^{-c_i \cdot f_i}, ID_i\rangle$, where $r_i, x_i, t_i, w'_i, f_i \in_R {Z_p}^*$, $h_i = H_1(ID_i)$, $b_i = H_3(ID_i, u_{i1}, v_{i1}, g^{r_i}, {h_i}^{r_i})$ and $e_i = H_4(ID_i, u_{i1}, v_{i1}, g^{r_i}, {h_i}^{r_i})$ for user i.
3. If H_5 was already queried with inputs $\left(g^{t_j}, g^{w'_j}.{y_1}^{-c_j \cdot f_j}\right)$, generate a fresh w'_j and recompute the last but two components. With very high probability, the new $\left(g^{t_j}, g^{w'_j}.{y_1}^{-c_j \cdot f_j}\right)$ will not result in a previously queried input set to H_5. Set $H_5\left(g^{t_j}, g^{w'_j}.{y_1}^{-c_j \cdot f_j}\right)$ as f_j.
4. Similarly if H_5 was already queried with inputs $\left(g^{t_i}, g^{w'_i}.{y_1}^{-c_i \cdot f_i}\right)$, generate a fresh w'_i and recompute the last but two components. With very high

probability, the new $\left(g^{t_i}, g^{w'_i}.{y_1}^{-c_i \cdot f_i}\right)$ will not result in a previously queried input set to H_5. Set $H_5\left(g^{t_i}, g^{w'_i}.{y_1}^{-c_i \cdot f_i}\right)$ as f_i.

5. The challenger computes $\bar{Z}_1 = (g^{x_i}.{y_1}^{c_i}.g^{t_i})^{x_j+t_j}$ where $c_i = H_2\left(ID_i, u_{i1}\right)$. It also computes $P_1 = (u_{i1}.{y_1}^{c_i}.g^{t_i})^{c_j}$ and $P_2 = y_1$ where $c_j = H_2\left(ID_j, u_{j1}\right)$.
6. The challenger computes $Z_2 = v_{i1}.v_{j1}$ and $Z_3 = (g^{t_i})^{t_j}$.
7. The challenger is given access to the $DH\left(y_1, ., .\right)$ oracle, since we assume the hardness of Strong-Diffie Hellman problem. The challenger makes use of the $DH\left(y_1, ., .\right)$ Oracle to answer the query as follows:
   – The challenger finds a Z such that $DH\left(P_2, P_1, Z_1/\bar{Z}_1\right)$ (valid since $P_2 = y_1$) and $H_6(Z_1, Z_2, Z_3) = Z$, where $Z_2 = v_{i1}.v_{j1}$ and $Z_3 = (g^{t_i})^{t_j}$.
   – If a Z exists, the challenger returns Z as the shared secret key.
   – Otherwise the challenger chooses $Z \in_R {Z_p}^*$ and for any further query of the form (Z_1, Z_2, Z_3) to the H_6 *Oracle*, if $DH\left(P_2, P_1, Z_1/\bar{Z}_1\right)$, $Z_2 = v_{i1}.v_{j1}$ and $Z_3 = (g^{t_i})^{t_j}$.

Finally the challenger returns Z as the shared secret key.

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

– The challenger passes the parameters $\langle u_{j1} = g^{x_j}, v_{j1} = {h_j}^{x_j}, d_{j2} = x_j + r_j.b_j + s_2.e_j, b_j, e_j,$
  ${h_j}^{s_2}, D.g^{-d_{j1}}, w_j + d_{j1}.H_5\left(D.g^{-d_{j1}}, g^{w_j}\right), ID_j\rangle$ to the adversary, where d_{j1} is the private key component associated with User j which is known to the challenger, $r_j, x_j, w_j \in_R {Z_p}^*$, $h_j = H_1\left(ID_j\right)$, $b_j = H_3\left(ID_j, u_{j1}, v_{j1}, g^{r_j}, {h_j}^{r_j}\right)$ and $e_j = H_4\left(ID_j, u_{j1}, v_{j1}, g^{r_j}, {h_j}^{r_j}\right)$. The parameters passed satisfy the checks as they are generated in the way similar to the scheme. $g^{t_j} = D.g^{-d_{j1}} = g^{b-d_{j1}}$.
– The challenger performs the checks specified in *Step* 2 of the **Key Agreement** algorithm described in *Section* 5 on $\langle u_{i1}, v_{i1}, d_{i2}, b_i, e_i, {h_i}^{s_2}, g^{t_i}, w_i + d_{i1}.H_5\left(g^{t_i}, g^{w_i}\right), g^{w_i}, ID_i\rangle$. If the checks pass, the challenger proceeds to next step. Else, it aborts.
– The challenger returns a $Z^* \in_R {Z_p}^*$ as the shared secret key. This won't be a valid shared secret key. But in order to find that this is invalid the adversary should have queried the H_6 *Oracle* with a valid tuple (Z_1, Z_2, Z_3). Thus the challenger computes $\bar{Z}_2 = (Z_2/v_{j1})^{{k_i}^{-1}}$ and $\bar{Z}_3 = Z_3.(g^{t_i})^{d_{j1}}$. The challenger also computes $S = \left(Z_1/\bar{Z}_2\bar{Z}_3\right)^{{c_i}^{-1}}$ where $c_i = H_2\left(ID_i, u_{i1}\right)$.
– Finally the challenger returns S as the solution for the CDH hard problem.

Lemma 3: The value returned by the challenger is the solution to the CDH instance of the SDH hard problem set in the beginning.

*Proof:*The challenger returns $S = \left(Z_1/\bar{Z}_2\bar{Z}_3\right)^{c_i^{-1}}$ where $c_i = H_2\left(ID_i, u_{i1}\right)$ as the solution to the hard problem.

- $S = \left(g^{(d_{i1}+t_i)(d_{j1}+b-d_{j1})}/\bar{Z}_2\bar{Z}_3\right)^{c_i^{-1}}$.
- $\overline{Z_2} = \left(Z_2/v_{j1}\right)^{k_i^{-1}} = \left(v_{i1}.v_{j1}/v_{j1}\right)^{k_i^{-1}} = v_{i1}{}^{k_i^{-1}} = \left(h_i{}^{x_i}\right)^{k_i^{-1}} = \left(g^{b.k_i}\right)^{x_i.k_i^{-1}} = g^{b.x_i}$.(**Note :** The component $h_i = \left(g^b\right)^{k_i}$ as $\tau_i = 1$.).
- $\overline{Z_3} = Z_3.\left(g^{t_i}\right)^{d_{j1}} = \left(g^{t_i}\right)^{(b-d_{j1})}.\left(g^{t_i}\right)^{d_{j1}} = g^{b.t_i}$.
- Therefore $S = \left(g^{(x_i+a.c_i+t_i)(d_{j1}+b-d_{j1})}/g^{b.x_i}.g^{b.t_i}\right)^{c_i^{-1}} = g^{ab}$.

Thus we have proved that the value returned by the challenger is solution to the CDH Problem. □

7 Additional Security Properties

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

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

Resistance to Reflection Attacks: A reflection attack occurs when an adversary can compromise a session in which the two parties have the same identity. A practical situation in which both parties with the same identity communicate is when a person wants to establish secure connection between her computers in the house and the one in the office. The proposed scheme is resistant to reflection attacks which can be proved by the techniques used in [5] and [11].

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

Resistance to Ephemeral Key Compromise Impersonation: Generally the users pick the ephemeral keys (t_i, g^{t_i}) from a pre-computed list in order to minimize online computation cost. But the problem with this approach is that the ephemeral components may be subjected to leakage. This attack considers the case when the adversary can make state-reveal queries even in the test session. [4] presents such an attack on the scheme presented by Fiore [5]. But our scheme is resistant to that type of an attack because when an adversary tries to impersonate a user j without knowing the private key of j (as in [4]), it cannot generate the components d_{j2} and the signature on g^{t_j} (We assume that w_i is erased immediately after the signature on g^{t_i} is computed and hence is not available to the adversary during state-reveal queries). Thus it is secure and resists ephemeral key compromise impersonation attack.

8 Conclusion

The main advantage of our scheme is that there is only a single round of communication between the pair of users and there is no predefined order in which messages are exchanged between the users. Moreover our scheme is secure against active adversary which can intercept and modify the messages as per will. The next advantage is that forking lemma is not used in the security reduction contributing to the tight reduction feature. This results in a reduction in the communication overhead. Our scheme also satisfies additional security attributes like forward secrecy, resistance to reflection attacks, key compromise impersonation attack and ephemeral key compromise impersonation attack. Finally our proof can also be modified to support security in the advanced CK+ model. This will be discussed in the full version of the paper.

References

1. Abe, M., Kiltz, E., Okamoto, T.: Compact cca-secure encryption for messages of arbitrary length. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 377–392. Springer, Heidelberg (2009)
2. Canetti, R., Krawczyk, H.: Analysis of key-exchange protocols and their use for building secure channels. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 453–474. Springer, Heidelberg (2001)
3. Cao, X., Kou, W., Du, X.: A pairing-free identity-based authenticated key agreement protocol with minimal message exchanges. Information Sciences 180(15), 2895–2903 (2010)
4. Cheng, Q., Ma, C.: Ephemeral key compromise attack on the ib-ka protocol. IACR Cryptology ePrint Archive 2009, 568 (2009)
5. Fiore, D., Gennaro, R.: Making the diffie-hellman protocol identity-based. In: Pieprzyk, J. (ed.) CT-RSA 2010. LNCS, vol. 5985, pp. 165–178. Springer, Heidelberg (2010)
6. Goh, E.-J., Jarecki, S.: A signature scheme as secure as the diffie-hellman problem. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 401–415. Springer, Heidelberg (2003)
7. Günther, C.G.: An identity-based key-exchange protocol. In: Quisquater, J.-J., Vandewalle, J. (eds.) EUROCRYPT 1989. LNCS, vol. 434, pp. 29–37. Springer, Heidelberg (1990)

8. Hafizul Islam, S.K., Biswas, G.P.: An improved pairing-free identity-based authenticated key agreement protocol based on {ECC}. Procedia Engineering 30, 499–507 (2012)
9. Katz, J., Yung, M.: Scalable protocols for authenticated group key exchange. Journal of Cryptology 20(1), 85–113 (2007)
10. Krawczyk, H.: HMQV: A high-performance secure diffie-hellman protocol. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 546–566. Springer, Heidelberg (2005)
11. Maurer, U.M., Wolf, S.: Diffie-hellman oracles. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 268–282. Springer, Heidelberg (1996)
12. Menezes, A., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography, ch. 14, pp. 617–618. CRC Press (1996)
13. Okamoto, T., Pointcheval, D.: The gap-problems: A new class of problems for the security of cryptographic schemes. In: Kim, K.-C. (ed.) PKC 2001. LNCS, vol. 1992, pp. 104–118. Springer, Heidelberg (2001)
14. Saeednia, S.: Improvement of gunther's identity-based key exchange protocol. Electronics Letters 36(18), 1535–1536 (2000)
15. Shamir, A.: Identity-based cryptosystems and signature schemes. In: Blakely, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 47–53. Springer, Heidelberg (1985)

Appendix

Key Sanity Check: After receiving the private key from the PKG in the key extract phase, the user performs the following check to ensure the correctness of the components of the private key.

The user first computes

$$c_i = H_2(ID_i, u_{i1})$$
$$b_i = H_3(ID_i, u_{i1}, v_{i1}, u_{i2}, v_{i2})$$
$$e_i = H_4(ID_i, u_{i1}, v_{i1}, u_{i2}, v_{i2})$$

Test 1: Check if $\frac{g^{d_{i1}}}{y_1^{H_2(ID_i, u_{i1})}} \stackrel{?}{=} u_{i1}$.

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

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

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

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

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

This can be verified as $\frac{h_i^{x_i + r_i \cdot b_i + s_2 \cdot e_i}}{({h_i}^{r_i})^{H_3(ID_i, u_{i1}, v_{i1}, u_{i2}, v_{i2})} . ({h_i}^{s_2})^{H_4(ID_i, u_{i1}, v_{i1}, u_{i2}, v_{i2})}} = {h_i}^{x_i} = v_{i1}$ where $b_i = H_3(ID_i, u_{i1}, v_{i1}, u_{i2}, v_{i2})$ and $e_i = H_4\,(ID_i, u_{i1}, v_{i1}, u_{i2}, v_{i2})$. Test 3 ensures the correctness of ${h_i}^{s_2}$. Test 2 and Test 3 ensures that g and h_i are raised to the same exponent x_i in u_{i1} and v_{i1} respectively.

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