A Strongly Secure and Efficient Certificateless Authenticated Asymmetric Group Key Agreement Protocol

Haiyan Sun\textsuperscript{a,}\textsuperscript{*}, Zengyu Cai\textsuperscript{b}, Jianwei Zhang\textsuperscript{a}, Ling Zhang\textsuperscript{a}, and Yong Gan\textsuperscript{c}

\textsuperscript{a}Software Engineering College, Zhengzhou University of Light Industry, Zhengzhou, 450002, China
\textsuperscript{b}School of Computer and Communication Engineering, Zhengzhou University of Light Industry, Zhengzhou, 450002, China
\textsuperscript{c}Zhengzhou Institute of Technology, Zhengzhou, 450044, China

Abstract

In Eurocrypt’2009, Wu et al. (2009) presented an important primitive named the asymmetric group key agreement (AGKA) protocol. In such a primitive, a group of users generate a common public encryption key, and each user only holds his own secret decryption key. Authenticated asymmetric group key agreement (AAGKA) protocols are a kind of AGKA protocol that can be secure against active attacks. AAGKA protocols in certificateless public key cryptography (CL-PKC) have some preponderance than those in identity-based cryptography and PKI cryptography. However, existing AAGKA protocols in CL-PKC only consider security against normal type adversaries, the weakest adversaries considered in CL-PKC literature. To solve this problem, an improved security model that considers security against super adversaries and a provably secure certificateless AAGKA protocol under the improved security model are proposed. Efficiency comparison shows that the proposed protocol is more efficient.

Keywords: asymmetric group key agreement; provable security; super adversaries; certificateless key cryptography

(Submitted on August 12, 2018; Revised on September 15, 2018; Accepted on October 8, 2018)

© 2018 Totem Publisher, Inc. All rights reserved.

1. Introduction

Group key agreement (GKA) protocols allow a group of users to agree on a common secret key through public channel, which serves as the encryption key of symmetric encryption. GKA protocols are widely used in group oriented applications, such as multicast communications and social communications. Since this primitive was proposed, many GKA protocols (e.g., [1–4]) have been built. However, GKA protocols are subject to a specific problem, i.e., any user outside the group cannot send messages to group internal users, since only they know the secret key. To address this problem, Wu et al. [5] presented an important primitive named the asymmetric group key agreement (AGKA) protocol. Generally speaking, in such a protocol, a batch of users can agree on a common public encryption key while their respective private decryption keys are different and secret. Naturally, AGKA protocols can be applied to the above group-oriented applications. Since this primitive was proposed, some AGKA protocols (e.g., [6–8]) have been proposed. However, these protocols are only secure against passive attacks. In practice, adversaries can mount powerful attacks, e.g., by modifying, replaying, or deleting messages. Based on this idea, the notion of an authenticated asymmetric group key agreement (AAGKA) protocol [9] is proposed, which ensures that any outside user cannot possibly gain a decryption key in the presence of the active attacker.

With the help of various public key cryptography (e.g. the PKI cryptography, identity-based cryptography [10], and certificateless cryptography [11]), AAGKA protocols can be realized. By the definition of trust hierarchy given in [12], AAGKA protocols under the PKI cryptography [9,13–17] can achieve trust level 3. However, there is a certificate management problem. Instead of certificates, users’ public keys are their identities, such as their email addresses or telephone numbers in identity-based AAGKA protocols [18–22]. Thus, identity-based AAGKA protocols eliminate the certificate management problems. However, identity-based AAGKA protocols are subject to a key escrow problem, since a trusted party Key Generator Centre (KGC) generates all users’ private keys. Certificateless AAGKA (CL-AAGKA) protocols [23–26] avoid the above key escrow and certificate management problem since KGC generates a component of the

\textsuperscript{*}Corresponding author.

E-mail address: sunhaiyan2520@163.com
user’s private key, and the user secretly generates the other component and derives his public key without any certificate.

Wei et al. [23] proposed the first CL-AAGKA protocol in 2012. However, there was no formal security definition for CL-AAGKA protocols in [23]. Lv et al. [24] proposed the first security model for CL-AAGKA protocols in 2014. However, it does not capture known-key security. Later, Zhang et al. [25] put forward a new security model that can capture known-key security, secrecy, and partial forward security. The security model in [26] is the same as that in [25]. However, adversaries in these security models own a parallel power as the “normal adversaries” [27], the weakest adversaries in certificateless public key cryptography literature. Similar to [27], if a CL-AAGKA protocol is secure against “super adversaries” and more powerful than “normal and strong adversaries”, the security level would be enhanced. In fact, super adversaries should be considered for CL-AAGKA protocols. For the existing AGKA protocols, messages used to generate group keys are all based on elements named batch multi-signature schemes, which means that it would be better if messages can be generated under the new public keys if replaced. Based on these, we design an improved security model and a strongly secure CL-AAGKA protocol.

In this paper, we propose an improved security model for CL-AAGKA, which considers the most powerful adversaries (i.e., super adversaries). On the basis of the Bilinear Diffie-Hellman Exponent (BDHE) assumption, we propose a provably secure CL-AGAKA protocol under the improved security model. Moreover, efficiency comparison shows that the proposed protocol is more efficient.

2. Preliminaries

2.1. Bilinear Maps

Let $P$ be the generator of an additive group $G$ of prime order $q$ and $G_T$ be a multiplicative group of $q$. A map $e : G \times G \rightarrow G_T$ is said to be a bilinear map if it meets the following requirements [26]:

1. **Bilinearity**: $e(aP, bQ) = e(P, Q)^{ab}$ for any $a, b \in \mathbb{Z}_q^*$, $Q \in G$.

2. **Non-degeneracy**: $e(P, P) \neq 1_{G_T}$.

3. **Computability**: $e$ is efficiently computable.

2.2. Complexity Assumption

$\ell$-Bilinear Diffie-Hellman Exponent ($\ell$-BDHE) Problem [26]: Given $P, Q,$ and $T_i = a'_i P$ in $G$ for $i = 1, 2, \ldots, \ell, \ell + 2, \ldots, 2\ell$, compute $e(P, Q)^{a'_{\ell+1}}$.

$\ell$-BDHE assumption means that for any polynomial-time algorithm $\mathcal{CH}$, $\Pr(\mathcal{CH}(P, Q, T_1, \ldots, T_{\ell+1}, \ldots, T_{2\ell})) = e(P, Q)^{a'_{\ell+1}}$ is negligible.

3. Our Improved Security Model

Based on the security models in [25-26], an improved security model for CL-AAGKA protocols is proposed in this section.

3.1. Notations

For the security model, we need some notations shown in Table 1.

**Definition 1** (Accepted) We say $\Pi^a_{\forall_i}$ is **accepted** if it possesses $ek^a_{\forall_i}$ ( $\neq$ null), $dk^a_{\forall_i}$ ( $\neq$ null), $sid^a_{\forall_i}$, and $pid^a_{\forall_i}$.

**Definition 2** (Partner) Two accepted instances $\Pi^a_{\forall_i}$ and $\Pi^a_{\forall_j}$ are called **partner** iff (a) $sid^a_{\forall_i} = sid^a_{\forall_j}$ and (b) $pid^a_{\forall_i} = pid^a_{\forall_j}$.
Table 1. Notations

<table>
<thead>
<tr>
<th>Notations</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$U_i$</td>
<td>The $i^{th}$ protocol participant</td>
</tr>
<tr>
<td>$\Pi^\pi_i$</td>
<td>An instance $\pi$ of participant $U_i$</td>
</tr>
<tr>
<td>$pid^\pi_i$</td>
<td>Partner ID of $\Pi^\pi_i$, the identity concatenation of all group members</td>
</tr>
<tr>
<td>$sid^\pi_i$</td>
<td>Session ID of $\Pi^\pi_i$</td>
</tr>
<tr>
<td>$tran^\pi_i$</td>
<td>The message concatenation from and to $\Pi^\pi_i$</td>
</tr>
<tr>
<td>$ek^\pi_i$</td>
<td>The encryption key of instance $\Pi^\pi_i$</td>
</tr>
<tr>
<td>$dk^\pi_i$</td>
<td>The decryption key of instance $\Pi^\pi_i$</td>
</tr>
<tr>
<td>$state^\pi_i$</td>
<td>The state of instance $\Pi^\pi_i$</td>
</tr>
<tr>
<td>$params$</td>
<td>The system parameter</td>
</tr>
<tr>
<td>$s$</td>
<td>The master key of KGC</td>
</tr>
<tr>
<td>$P_i$</td>
<td>The public key of $U_i$</td>
</tr>
<tr>
<td>$x_i$</td>
<td>The secret value of $U_i$</td>
</tr>
<tr>
<td>$D_i$</td>
<td>The partial private key of $U_i$</td>
</tr>
</tbody>
</table>

3.2. Security Model

The model is defined by the following game played between an active adversary $A$ and a challenger $C\mathcal{H}$. $A$ can be a Type I adversary $A_i$ or a Type II adversary $A_{ii}$. $A_i$ can replace any user’s public key as arbitrary in ignorance of KGC’s master key. $A_{ii}$ cannot replace the target user’s public key in advance of knowing the master key.

**Setup** $C\mathcal{H}$ obtains $s$ and the opened $params$ by running the Setup algorithm. Then, $C\mathcal{H}$ sends $params$ to $A_i$ or $params$ with $s$ to $A_{ii}$.

**Phase 1** $A$ performs the following queries with no sequence.

**Create** ($U_i$): $C\mathcal{H}$ creates $U_i$’s public/private key pair ($P_i, (D_i, x_i)$).

**Public-Key** ($U_i$): $C\mathcal{H}$ returns $P_i$.

**Partial-Private-Key** ($U_i$): $C\mathcal{H}$ returns $D_i$.

**Public-Key-Replacement** ($U_i, P'_i$): $C\mathcal{H}$ replaces $U_i$’s public key with $P'_i$ ($P'_i \neq P_i$) and updates $U_i$’s secret value to null. Note that null denotes an empty string.

**Secret-Value** ($U_i$): $C\mathcal{H}$ returns $x_i$.

**Super-Send** ($\Pi^\pi_i, \Delta$): Receiving message $\Delta$ is sent to $\Pi^\pi_i$, and $C\mathcal{H}$ returns corresponding messages. If $\Delta = (sid, pid)$, the protocol with session ID $sid$ and partner ID $pid$ is initiated. Here, messages should be generated under the new public keys if replaced.

**Encryption-Key** ($\Pi^\pi_i$): $C\mathcal{H}$ returns the encryption key $ek^\pi_i$. The encryption key should be generated under the replaced public keys if replaced.

**Super-Decryption-Key** ($\Pi^\pi_i$): $C\mathcal{H}$ returns the decryption key $dk^\pi_i$. The decryption key should be generated under the replaced public keys if replaced.
Phase 2. $\mathcal{A}$ issues the following Test query once after choosing two messages $(m_0, m_1)$ and an instance $\Pi_U^x$ with freshness (see Definition 3).

Test $(\Pi_U^x, (m_0, m_1))$: $\mathcal{CH}$ chooses a bit $b \in \{0,1\}$ uniformly at random, uses $ek_U^x$ to encrypt $m_b$, and then returns the obtained ciphertext $c$ to $\mathcal{A}$.

Response $\mathcal{A}$ returns a bit $b'$ for $b$. The advantage of $\mathcal{A}$ is defined to be $\text{Adv}(\mathcal{A}) = 2\text{Pr}[b' = b] - 1$, where $b' = b$ means $\mathcal{A}$ wins.

Definition 3 (Freshness) An accepted instance $\Pi_U^x$ is fresh if the conditions listed below happen.

1. Super-Decryption-Key query has never been carried out to $\Pi_U^x$, or any of his partners.

2. At least one participant $U_j \in \text{pid}_U^x$ has never been asked Partial-Private-Key $(U_j)$ by a type I adversary or has never been asked Public-Key-Replacement $(U_j, P'_j)$ or Secret-Value $(U_j)$ by a type II adversary.

3. If $\Pi_U^x$ is partnered with $\Pi_U^x$ and $U_j$ is corrupted, then $\mathcal{A}$, in the name of $\Pi_U^x$, can only forward messages generated by $\Pi_U^x$ to $\Pi_U^x$. Notice that $U_j$ is corrupted if a type I adversary has queried both Partial-Private-Key $(U_j)$ and Secret-Value $(U_j)$ or both Secret-Value $(U_j)$ and Public-Key-Replacement $(U_j, P'_j)$, or if a type II adversary has queried Public-Key-Replacement $(U_j, P'_j)$ or Secret-Value $(U_j)$.

Definition 4 (Semantic security) We say a CL-AAGKA protocol is semantically secure under Ind-ID-CPA, if, for any polynomial-time adversary $\mathcal{A}$, the advantage of $\text{Adv}(\mathcal{A})$ is negligible.

Remark 1 Known key secrecy, partial forward security, and super security are captured in the above security model. Our security model considers security against super adversaries under queries including Public-Key-Replacement, Super-Send, Encryption-Key, and Super-Decryption-Key. After Public-Key-Replacement queries, the challenger sets the new secret value to null, which means that $\mathcal{CH}$ has to correctly answer any query without knowing the new secret value, i.e., messages, the encryption key, and the decryption key all should be generated under the new public keys if replaced.

4. Our Proposed Protocol

A CL-AAGKA protocol motivated by [19,28] is proposed in this section. It consists of three stages (ten algorithms).

Phase 1 Generate the public/private keys of user $U_i$ with identity $ID_i \in \{0,1\}^\ast$.

Setup: On input a security parameter $k$, KGC performs what follows:

1. Choose a big prime $q$, an additive group $G$ with order $q$, one $G$'s generator $P$, and a multiplicative group $G_T$ with order $q$, and then construct a bilinear pairing $e: G \times G \rightarrow G_T$.
2. Randomly choose $s \in \mathbb{Z}_q^\ast$ and compute $P_{\text{pub}} = sP$.
3. Choose hash functions $H_1, H_3, H_4 : \{0,1\}^\ast \rightarrow G$, $H_2 : \{0,1\}^\ast \rightarrow \mathbb{Z}_q^\ast$, and $H_5 : G_T \rightarrow \{0,1\}^\ast$, in which $t$ is plaintexts’ max binary length.
4. Secretly keep the master key $s$ while publishing the system parameters $\text{params} = (G, G_T, e, P, P_{\text{pub}}, H_1, \cdots, H_5)$.

Partial-Key-Extract: The partial private key of user $U_i$ is set to $D_i = sH_i(ID_i)$ by KGC and is sent to $U_i$ via secure channel.
Secret-Value-Set: The secret value of user $U_i$ is set to $x_i \in Z_q^*$, which is selected by user $U_i$ at random.

Private-Key-Set: The full private key of user $U_i$ is set to $sk_i = (D_i, x_i)$.

Public-Key-Set: The public key of user $U_i$ is set to $P_i = x_i P$.

Phase 2 Achieve shared group keys.

Agreement: Protocol participant $U_i$, $(i=1, \ldots, n)$, with identity $ID_i$, full private key $sk_i = (D_i, x_i)$, and public key $P_i$ does as follows.

1) Stochastically pick $r_i \in Z_q^*$ and calculate $R_i = r_i P$.
2) Calculate $h_i = H_2(ID_i, R_i, P_i)$ and $Z = H_1(parms)$.
3) For $1 \leq j \leq n$, calculate $V_{ij} = D_j + x_j Z + (r_j + h_j x_j) H_4(j)$.
4) Publish $\sigma_i = (R_i, P_i, \{V_{ij}\}_{j \in \{1, 2, \ldots, n\}, i})$.

Encrypt-Key-Gen: Each participant $U_i$ first checks if

$$e(\sum_{j=1}^{n} V_{ij}, P) = e(\sum_{j=1}^{n} H_1(ID_j, P_{pub}), e(\sum_{j=1}^{n} P_j, Z) e(\sum_{j=1}^{n} (R_j + h_j P_j), H_4(i))$$

If it is equal, the group encryption key is set to $(W,Y)$, where

$$W = \sum_{i=1}^{n} (R_i + h_i P_i), Y = e(\sum_{i=1}^{n} H_1(ID_i), P_{pub}) e(\sum_{i=1}^{n} P_i, Z)$$

Note that any outside user can also compute $(W,Y)$, since these values are public.

Decrypt-Key-Gen: Each participant $U_i$ computes its group decrypt key as

$$dk_i = \sum_{j=1}^{n} V_{ij}$$

If $e(dk_i, P) = e(H_4(i), W) Y$ holds, $U_i$ accepts $dk_i$; otherwise, it aborts.

Note that if correctly formed, $e(dk_i, P) = e(H_4(i), W) Y$ must hold. The correctness is shown as follows.

$$e(dk_i, P) = e(\sum_{j=1}^{n} V_{ij}, P) = e(\sum_{j=1}^{n} (D_j + x_j Z + (r_j + h_j x_j) H_4(i), P)$$

$$= e(\sum_{j=1}^{n} D_j, P) e(\sum_{j=1}^{n} x_j Z, P) e(\sum_{j=1}^{n} (r_j + h_j x_j) H_4(i), P)$$

$$= e(\sum_{j=1}^{n} H_1(ID_j, P_{pub}), e(\sum_{j=1}^{n} P_j, Z) e(\sum_{j=1}^{n} (R_j + h_j P_j), H_4(i)) = Ye(H_4(i), W)$$

Phase 3 Encryption/Decryption.

Encrypt: For a plaintext $m$, an entity chooses a random number $\delta \in Z_q^*$, computes $c_1 = \delta P$, $c_2 = \delta W$, $c_3 = m \oplus H_5(Y^\delta)$, and outputs the ciphertext $c = (c_1, c_2, c_3)$.
**Decrypt:** Under $d_{k_i}$, $U_i$ computes
\[ m = c_3 \oplus H_3(e(d_{k_i}, c_3)e(-H_4(i), c_3)) \]

Note that the correctness of $m$ holds, since $e(d_{k_i}, P) = e(H_4(i), WY)$.

5. Security Proof

**Theorem 1** Our CL-AAGKA protocol is semantically secure under Ind-ID-CPA under the $\ell$-BDHE assumption.

This theorem follows from Lemma 1 and Lemma 2. The proofs of the two lemmas are similar. Here, we just give the proof of Lemma 1, to save space.

**Lemma 1** Under the $\ell$-BDHE assumption, for any polynomial time $\mathcal{A}_i$, $\text{Adv}(\mathcal{A}_i)$ is negligible.

**Lemma 2** Under the $\ell$-BDHE assumption, for any polynomial time $\mathcal{A}_i$, $\text{Adv}(\mathcal{A}_i)$ is negligible.

**Proof of Lemma 1.**

Now, we adopt proof by contradiction to prove Lemma 1. If there is one $\mathcal{A}_i$ such that $\text{Adv}(\mathcal{A}_i)$ is non-negligible, then there is an algorithm $\mathcal{CH}$ to solve the $\ell$-BDHE problem with a non-negligible probability. Specifically, $\mathcal{CH}$’s task is to compute $e(P, Q)^{\alpha i}$ with the help of $(P, Q, T_1, \ldots, T_6)$, where $T_i = \alpha^i P$, $i \in \{1, \ldots, \ell, \ell + 2, \ldots, 2\ell\}$ with some unknown $\alpha \in Z_q^\ast$.

Supposed that there are at most $\ell$ group participants for any session $sid_i$. For $sid_i$, the tuple $(sid_i, \gamma_{sid_i})$ is recorded by $\mathcal{CH}$, where $Pr[\gamma_{sid_i} = 1] = \eta$, $Pr[\gamma_{sid_i} = 0] = 1 - \eta$ ($\eta$ will be determined later).

**Setup** $\mathcal{CH}$ sets $P_{\text{pub}} = T_0 = \alpha P$ and selects $\text{params} = (G, G_\gamma, e, P, P_{\text{pub}}, H_1, H_2, \ldots, H_6)$ as the system parameters. Then, $\mathcal{CH}$ sends $\text{params}$ to $\mathcal{A}_i$.

**Query** The Test query can only be queried once by $\mathcal{A}_i$, while the others can be queried out of order. Every query needs a list to record the data. Suppose the maximum queried numbers of queries $H_5$, Partial-Private-Key, and Super-Decryption-Key are $q_{H_5}$, $q_{\text{pub}}$, and $q_{\text{DPK}}$, respectively.

$H_1(\text{ID}_i)$: $\mathcal{CH}$ maintains a list $L_{H_1}$ of tuples $(\text{ID}_i, \mu_i, id_i, \gamma_{H_1})$. Using index $\text{ID}_i$ to search for $L_{H_1}$, $id_i$ is returned if found; otherwise, $\mathcal{CH}$ perform the following steps:

1) Stochastically select $\gamma_{H_1} \in \{0, 1\}$ so that $Pr[\gamma_{H_1} = 1] = \eta$, $Pr[\gamma_{H_1} = 0] = 1 - \eta$.
2) Pick $\mu_i \in Z_q^\ast$, then set $id_i = \mu_i P$ if $\gamma_{H_1} = 0$, or $id_i = \mu_i P + T_0$ if $\gamma_{H_1} = 1$.
3) Add $(\text{ID}_i, \mu_i, id_i, \gamma_{H_1})$ to $L_{H_1}$ and respond with $id_i$.

$H_2(\text{ID}_i, R_i, P_i)$: $\mathcal{CH}$ maintains a list $L_{H_2}$ of tuples $(\text{ID}_i, R_i, P_i, h_i)$. Using index $(\text{ID}_i, R_i, P_i)$ to search for $L_{H_2}$, $h_i$ is returned if found; otherwise, $\mathcal{CH}$ randomly selects $h_i \in Z_q^\ast$, adds $(\text{ID}_i, R_i, P_i, h_i)$ to the list $L_{H_2}$, and outputs $h_i$.

$H_3($params$)$: $\mathcal{CH}$ maintains a list $L_{H_3}$ of tuples $(\text{params}, \beta, Z)$. Using index $\text{params}$ to search for $L_{H_3}$, $Z$ is returned if found; otherwise, $\mathcal{CH}$ randomly selects $\beta \in Z_q^\ast$, computes $Z = \beta P$, records $(\text{params}, \beta, Z)$ on the list $L_{H_3}$, and returns $Z$. 
\(H_4(j): \mathcal{C}H\) maintains a list \(L_{u_i}\) of tuples \((j,c_j,z_j)\). Using index \(j\) to search for \(L_{u_i}\), \(z_j\) is returned if found; otherwise, \(\mathcal{C}H\) does as follows:

1. Pick \(c_j \in Z_q^*\) at random, then set \(z_j = c_j P + T_j\) if \(j \leq \ell\); otherwise, set \(z_j = c_j P\).
2. Add the tuple \((j,c_j,z_j)\) to the list \(L_{u_i}\) and respond with \(z_j\).

\(H_5(\chi_i): \mathcal{C}H\) maintains a list \(L_{u_i}\) of tuples \((\chi_i,\omega_i)\). Using index \(\chi_i\) to search for \(L_{u_i}\), \(\omega_i\) is returned if found; otherwise, \(\mathcal{C}H\) randomly selects \(\omega_i \in \{0,1\}^*\), records \((\chi_i,\omega_i)\) on the list \(L_{u_i}\), and outputs \(\omega_i\).

Create \((U_i):\) Suppose \(U_i\)'s identity is \(ID_i\). \(\mathcal{C}H\) maintains a list \(L_{c}\) of tuples \((ID_i,\mu_i,D_1,x_i,P_i,\gamma_{H_{i}})\). Using index \(ID_i\) to search for \(L_{c}\), \(\mathcal{C}H\) does nothing if nothing found; otherwise, \(\mathcal{C}H\) obtains \((ID_i,\mu_i,\mu_i,\gamma_{H_{i}})\) by a \(H_i(ID_i)\) Query and then performs the following steps:

1. If \(\gamma_{H_{i}} = 0\), choose \(x_i \in Z_q^*\) at random, set \(P_i = x_i P\), set \(D_1 = \mu_i T_i\), and then add \((ID_i,\mu_i,D_1,x_i,P_i,\gamma_{H_{i}})\) to \(L_{c}\).
2. Otherwise, choose \(x_i \in Z_q^*\) at random, set \(P_i = x_i P\), set \(D_1 = \text{null}\), and then add \((ID_i,\mu_i,D_1,x_i,P_i,\gamma_{H_{i}})\) to \(L_{c}\).

In general, it is supposed that Create queries on related participants have been made before the following queries.

Public-Key \((U_i):\) Suppose \(U_i\)'s identity is \(ID_i\). \(\mathcal{C}H\) obtains \((ID_i,\mu_i,D_1,x_i,P_i,\gamma_{H_{i}})\) from \(L_{c}\) and then outputs \(P_i\).

Partial-Private-Key \((U_i):\) Suppose \(U_i\)'s identity is \(ID_i\). \(\mathcal{C}H\) obtains \((ID_i,\mu_i,D_1,x_i,P_i,\gamma_{H_{i}})\) from \(L_{c}\) and then does as follows:

1. If \(\gamma_{H_{i}} = 1\), abort (E1).
2. Else output \(D_1\) as the answer.

Secret-Value \((U_i):\) Suppose \(U_i\)'s identity is \(ID_i\). \(\mathcal{C}H\) obtains \((ID_i,\mu_i,D_1,x_i,P_i,\gamma_{H_{i}})\) from \(L_{c}\) and then outputs \(x_i\).

Public-Key-Replacement \((U_i,P_i')\): Suppose \(U_i\)'s identity is \(ID_i\). \(\mathcal{C}H\) obtains \((ID_i,\mu_i,D_1,x_i,P_i,\gamma_{H_{i}})\) from \(L_{c}\) and then updates \(P_i\) to \(P_i'\) and sets \(x_i = \text{null}\).

Super-Send \((\Pi_{U_i},\Lambda):\) \(\mathcal{C}H\) maintains a list \(L_{s}\) of tuples \((ID_i,sid_{U_i},r,V_{U_i})\). Assume \(\text{pid}_{U_i}^S = \{ID_1,ID_2,\ldots,ID_n\}\). \(\mathcal{C}H\) first recovers \(\gamma_{\text{sid}_{U_i}^S}\), searches for a tuple \((ID_i,\mu_i,D_1,x_i,P_i,\gamma_{H_{i}})\) in \(L_{c}\), submits \(\text{params}\) to \(H_4\) oracle and recovers \((\text{params},\beta,Z)\) from \(L_{H_{i}}\), submits \(j\) to \(H_4\) oracle and recovers \((j,c_j,z_j)\) from \(L_{H_{i}}\) for \(1 \leq j \leq n\); then does as follows:

- If \((\gamma_{\text{sid}_{U_i}^S},\gamma_{H_{i}}) = (0,0)\), generate the following answer:

1. Choose \(r_i,h_i \in Z_q^*\) and compute \(R_i = r_i P - h_i P\).
2. Set \(H_4(ID_i,R_1,P) = h_i\) and add \((ID_i,R_1,P,h_i)\) to the list \(L_{H_{i}}\).
3. For \(1 \leq j \leq n\), compute \(V_{U,i} = D_1 + \beta P + r_i c_j P + j T_i\).
4. Add \((ID_i,sid_{U_i},r,V_{U,i})\) to \(L_{s}\) and publish \((R_i,P_i,V_{U,i})_{i\in[1,\ldots,n]}\).

- Else if \((\gamma_{\text{sid}_{U_i}^S},\gamma_{H_{i}}) = (0,1)\), generate the following answer:
1) Choose \( r, h \in Z_q \) and compute \( R_i = rP - hP - \sum_{k=1}^{n} T_{(k,i)} \).

2) Set \( H_s(ID_s, R_i, P) = h \) and add \((ID_s, R_i, P, h)\) to the list \( L_{H_s} \).

3) For \( 1 \leq j \leq n \), compute \( V_{i,j} = \mu T_i + \beta P_i + c_i R_i + c_i h_j P_i + r T_j - \sum_{j=1}^{n} T_{(i,j)} \).

4) Add \((ID_s, sid_i^s, r_i, V_{i,j})\) to \( L_q \) and publish \((R_i, P_i, V_{i,j})_{j=1,2,...,n,j} \).

- Else if \((\gamma_{sid_i^s}, \gamma_{H_s}) = (1,0)\), generate the following answer:

1) Choose \( r, h \in Z_q \) and compute \( R_i = rP - hP - \sum_{k=1}^{n} T_{(k,i)} \).

2) Set \( H_s(ID_s, R_i, P) = h \) and add \((ID_s, R_i, P, h)\) to the list \( L_{H_s} \).

3) For \( 1 \leq j \leq n, i \neq j \), compute \( V_{i,j} = \mu T_i + \beta P_i + c_i R_i + c_i h_i P_i + r T_j + T_{(i,j)} \).

4) Add \((ID_s, sid_i^s, r_i, null)\) to \( L_q \) and publish \((R_i, P_i, V_{i,j})_{j=1,2,...,n,j} \).

- Else if \((\gamma_{sid_i^s}, \gamma_{H_s}) = (1,1)\), generate the following answer:

1) Choose \( r, h \in Z_q \) and compute \( R_i = rP - hP - \sum_{k=1}^{n} T_{(k,i)} \).

2) Set \( H_s(ID_s, R_i, P) = h \) and add \((ID_s, R_i, P, h)\) to the list \( L_{H_s} \).

3) For \( 1 \leq j \leq n, i \neq j \), compute \( V_{i,j} = \mu T_i + \beta P_i + c_i R_i + c_i h_i P_i + r T_j - \sum_{k=1}^{n} T_{(i,j)} \).

4) Add \((ID_s, sid_i^s, r_i, null)\) to \( L_q \) and publish \((R_i, P_i, V_{i,j})_{j=1,2,...,n,j} \).

**Encryption-Key** (\( \Pi_{U_i}^e \)): Assume \( tran_{i}^e = \{\sigma_1, \ldots, \sigma_n\} \), where \( \sigma_k = (R_k, P_k, V_{k,j})_{j=1,2,...,n,j} \). If state_{U_i} is not accepted, \( \mathcal{H} \) returns null; otherwise, \( \mathcal{H} \) outputs \((W, Y)\) where \( W = \sum_{k=1}^{n} (R_k + H_s(ID_s, R_k, P_k))P_k \).

\[ Y = e(\sum_{k=1}^{n} H_s(ID_s, P_{pk})\sigma_k, H_s(params)) \]

**Super-Decryption-Key** (\( \Pi_{U_i}^e \)): \( \mathcal{H} \) first recovers \( \gamma_{sid_i^e} \) corresponding to \( sid_i^e \) and then does as follows.

1) If \( \gamma_{sid_i^e} = 1 \), abort \((E2)\).

2) Else if \( \Pi_{U_i}^e \) is not accepted, return null.

3) Else obtain \( V_{i,j} \) from \( L_q \) and \( tran_{i}^e = \{\sigma_1, \ldots, \sigma_n\} \), where \( \sigma_k = (R_k, P_k, V_{k,j})_{j=1,2,...,n,j} \). Compute and output \( dk_i = \sum_{k=1}^{n} V_{k,j} \).

**Test** (\( \Pi_{U_i}^e, (m_0, m_i) \)): At some point, \( \mathcal{A} \) chooses a fresh instance \( \Pi_{U_i}^e \) and messages \( m_0 \) and \( m_i \). Assume \( pid_{U_i}^e = \{ID_{i}, ID_{i}, \ldots, ID_{i}\} \), \( tran_{i}^e = \{\sigma^e_1, \sigma^e_2, \ldots, \sigma^e_n\} \), where \( \sigma^e_k = (R^e_k, P^e_k, V_{k,j})_{j=1,2,...,n,j} \). \( \mathcal{H} \) does as follows:

1) For \( 1 \leq k \leq n \), recover \((ID_{i}^e, \mu_{i}^e, D_{i}^e, x_{i}^e, P_{i}^e, y_{i}^e)\), \((ID_{i}^e, \mu_{i}^e, id_{i}^e, y_{i}^e)\), \((ID_{i}^e, R_{i}^e, P_{i}^e, h_{i}^e)\) and \((k, c_{i}^e, z_{i}^e)\) from \( L_{C}, L_{H_i}, L_{H_i}, \) and \( L_{H_i} \), respectively. Recover \((params, \beta^e, Z^e)\) from \( L_{H_i} \).
2) If \( \gamma_{a_{24}} = 1 \) and there exists one and only one \( \gamma_{a_{k}}' = 1 \), turn Step (3). Otherwise, abort (E3).

3) Check if \( e(V'_{a_j}, P) = e(id_j, T_j)e(P'_j, Z')e(R'_j + b'_j P'_j, z'_j) \), where \( 1 \leq k \leq n, j = 1 \) if \( k = n \), \( j = k + 1 \) otherwise. If all the equations hold, turn Step (4). Otherwise, abort (E4).

4) For \( 1 \leq k \leq n \), recover \( r'_j, r'_2, \ldots, r'_n \) from \( L_k \), then \( e_{n-k} = (W', Y') = (\sum_{k=1}^{n} r'_j P, e(T_j + \sum_{k=1}^{n} \mu'_j P, T_j) e(\sum_{k=1}^{n} P'_j, \beta' P_j)) \).

5) Set \( c_1 = Q, c_2 = \sum_{k=1}^{n} r'_j Q \), pick \( \theta \in [0, 1]' \), and compute \( c_3 = m_b \oplus \theta \), where \( b \in [0,1] \).

6) Return \( c = (c_1, c_2, c_3) \). Note that if \( A_k \) has made a \( H_5 \) query on \( e(\alpha^{(s_1)} P + \sum_{i=1}^{n} \mu'_i T_i, Q) e(\sum_{k=1}^{n} P'_j, \beta' P_j) \), he can distinguish that if \( c \) is formed correctly.

**Response** Knowing the guess \( b' \in [0,1] \) from \( A_k \), \( CH \) chooses a tuple \( (\chi', \omega') \) from \( L_{n+1} \) and outputs the queried \( e(P, Q)^{\omega'} \) as

\[
\chi', e(-\sum_{k=1}^{n} \mu'_k T_i, Q)e(-\sum_{k=1}^{n} P'_k, \beta' Q)
\]

Now, we analyze the advantage of \( \text{Adv}(CH) \). \( CH \) succeeding means that \( A_k \) succeeds and \( CH \) indeed obtains the correct \( \chi' \) from \( L_{2n} \). Thus, \( \text{Adv}(CH) \geq \text{Adv}(A_k) \text{Pr}[CH \text{ not abort}] / q_{H_5} \). We can easily be aware that if all the events do not happen, \( CH \) will not abort. Therefore,

\[
\text{Pr}[CH \text{ not abort}] = \text{Pr}[E_1 \lor E_2 \lor E_3 \lor E_4] = \text{Pr}[E_1] \text{Pr}[E_2] \text{Pr}[E_3] \text{Pr}[E_4]
= (1-\eta)^{q_{psk}} (1-\eta)^{q_{psk} + q_{Dr} + \ell + 1} \eta^{2} (1-\eta)^{-n-1}
= \eta^{2} (1-\eta)^{q_{psk} + q_{Dr} + \ell + 1}
\geq \eta^{2} (1-\eta)^{q_{psk} + q_{Dr} + \ell - 1}
\]

When \( \eta = \frac{2}{q_{psk} + q_{Dr} + \ell + 1} \), \( \eta^{2} (1-\eta)^{q_{psk} + q_{Dr} + \ell - 1} \) has a maximum

\[
\frac{4}{(q_{psk} + q_{Dr} + \ell + 1)^{2}} (1-\frac{2}{q_{psk}} + \frac{2}{q_{Dr}} + \ell + 1)^{q_{psk} + q_{Dr} + \ell + 1} (1-\frac{2}{q_{psk}} + \frac{2}{q_{Dr}} + \ell + 1)^{-2}
\]

When \( q_{psk} + q_{Dr} \) is sufficiently large, we have \( \text{Pr}[CH \text{ not abort}] \geq \frac{4}{\text{e}^{2}(q_{psk} + q_{Dr} + \ell + 1)^{2}} \).

Hence,

\[
\text{Adv}(CH) \geq \frac{4}{q_{H_5}(q_{psk} + q_{Dr} + \ell + 1)^{2}} \text{Adv}(A_k)
\]

Then, \( \text{Adv}(CH) \) is non-negligible since \( \text{Adv}(A_k) \) is non-negligible. This completes the proof.

**6. Efficiency Comparison**

Up to now, protocols [23-26] are CL-AAGKA protocols. The results of efficiency comparison with them are listed in Table 2. Two high time complexity operations are considered, i.e., the pairing operation (denoted by \( P \)) and the scalar
multiplication operation in $G$ (denoted by $M$).

<table>
<thead>
<tr>
<th></th>
<th>[23]</th>
<th>[24]</th>
<th>[25-26]</th>
<th>Our protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost in Stage 2</td>
<td>$(2n+1)P + nM$</td>
<td>$nP + (n+1)M$</td>
<td>$8P + 3(n+1)M$</td>
<td>$6P + 2(n+1)M$</td>
</tr>
<tr>
<td>Time(ms)</td>
<td>$26.25n+11.25$</td>
<td>$15n+3.75$</td>
<td>$11.25n+101.25$</td>
<td>$7.5n+75$</td>
</tr>
</tbody>
</table>

In our protocol, the computation cost in key-agreement phase for a participant is $(n + 2)M$, and the computation cost in Encrypt-Key-Gen and Encrypt-Key-Gen phases for a participant is $6P + nM$. Thus, in our protocol the total computation cost in Stage 2 for a participant is $6P + 2(n + 1)M$. We use the experimental results in [29], i.e., the time consumption of $P$ and $M$ are 11.25ms and 3.75ms, respectively. According to Table 2, our protocol will be more efficient than other protocols [23-26] with an increase in group size.

7. Conclusions

An improved security model under super adversaries and a provably secure CL-AAGKA protocol are proposed. Efficiency comparison shows that our protocol is more efficient. Thus, it is more competitive in security and efficiency.

Acknowledgements

This work is supported by the National Natural Science Foundation of China (61502436, 61572445, 61672471), Science and Technology Program of Henan Province (172102210059, 162102210060), Plan for Scientific Innovation Talent of Henan Province (184200510010), Program for Innovative Research Team in Science and Technology in University of Henan Province (18IRTSTHN012), and Doctor Fund Project of Zhengzhou University of Light Industry (2014BSJ081).

References

8. X. Zhao, D. Wei, and H. Wang, “Asymmetric Group Key Agreement with Traitor Traceability,” in Proceedings of the 2010 International Conference on Computational Intelligence and Security, pp. 347-351, Nanning, China, December 2010
Haiyan Sun received her Ph.D. from Beijing University of Posts and Telecommunications in 2014. She currently works at Zhengzhou University of Light Industry. Her research interests include network security and cryptography.

Zengyu Cai received his M.S. degree from Northeast Normal University in 2006. He is currently an associate professor at Zhengzhou University of Light Industry. His research interests include trusted computing, plan recognition, and information security.

Jianwei Zhang received his Ph.D. from PLA Information Engineering University in 2010. He is currently a professor at Zhengzhou University of Light Industry. His research interests include broadband information networks and information security.

Ling Zhang received her Ph.D. from Beijing University of Posts and Telecommunications in 2014. She currently works at Zhengzhou University of Light Industry. Her research interests include intrusion detection and information security.

Yong Gan received his M.S. degree from Xi’an Microelectronics Technology Institute in 1989. He is currently a professor at Zhengzhou Institute of Technology. His research interests include information security and blockchain.