Secure Password-Based Authenticated Group Key Agreement for Data-Sharing Peer-to-Peer Networks

Qiang Tang[1,*] and Kim-Kwang Raymond Choo[2,**]

[1] Information Security Group, Royal Holloway,
University of London, Egham, Surrey TW20 0EX, UK
[2] Information Security Institute, Queensland University of Technology,
GPO Box 2434, Brisbane, QLD 4001, Australia
qiang.tang@rhul.ac.uk, k.choo@qut.edu.au

Abstract. We explore authenticated group key agreement in data-sharing Peer-to-Peer networks. We first propose a novel password-based authenticated group key agreement protocol with key confirmation. We present a formal statement of its security in a variant of the Bresson *et al.* security model adapted for the password-based setting. A discussion of the limitations of our protocol in the case where the group size becomes large is then presented. We conclude the paper with an enhanced version of the protocol, using a CAPTCHA technique, designed to make it more robust against online password guessing attacks.

Keywords: P2P network, key agreement, provable security, CAPTCHA.

1 Introduction

Data-sharing Peer-to-Peer (P2P) systems such as Napster[1] and Gnutella[2], are becoming increasingly popular for sharing large amounts of data, in particular music files. Because of the increasing popularity and the potential future applications of such systems, they have attracted much attention from the research community. We can broadly classify data-sharing P2P systems into two categories based on their system architecture. In the first category, the system has a central server that requires a one-time off registration by users prior to using the data-sharing service. However, all subsequent data transfers are conducted among the users without the involvement of the server. One typical example of a data-sharing P2P system in this category is Napster. In the second category, there is no central server and all operations are conducted in a self-organised manner. In such a framework, the precise specifications are dependent on the

* This research has been partially supported by a Thomas Holloway Studentship.
** This work was partially funded by the Australian Research Council Discovery Project Grant DP0345775.

[1] http://www.napster.com/
[2] http://www.gnutella.com/

J. Zhou, M. Yung, and F. Bao (Eds.): ACNS 2006, LNCS 3989, pp. 162–177, 2006.

individual system, and an example of such a system is Gnutella. Systems in the first category are more easily monitored and controlled but not easily scalable whilst systems in the second category are scalable at the expense of management and security.. We refer the interested reader to recent work of Daswani, Garcia-Molina, and Yang [8] for a more comprehensive treatment of the open issues faced by existing data-sharing P2P systems.

In this paper, we primarily focus on the group key agreement issue in first category systems. Our results can be extended to second category systems if a trusted authentication server is added for the purpose of key establishment (i.e., authenticated key agreement) at the expense of the self-organised nature that defines category 2. However, authenticated key agreement allows the participants to authenticate each other when establishing a shared session key, and addresses some of the open issues in data-sharing P2P systems.

We first propose a novel password-based authenticated group key agreement protocol which provides key confirmation. This protocol is proven secure in an adapted version of the Bresson *et al.* model [4]. Like other password-based protocols in the literature, our protocol is vulnerable to online password guessing attacks when the group size becomes very large. As a counter-measure, we enhance our proposed protocol using a Completely Automated Public Turing Test to Tell Computers and Humans Apart (CAPTCHA) technique [2] and federated signature verification (both services are provided by the trusted authentication server).

The rest of this paper is organised as follows. We present our proposed password-based authenticated group key agreement protocol in Section 2. We then describe the security model in which we work in and present the security proof for our proposed protocol in Section 3. In Section 4, we describe the enhanced protocol and demonstrate its security against online password guessing attacks. We conclude this paper in Section 5.

2 A Novel Password-Based Group Key Agreement Protocol

2.1 Review of Password-Based Group Key Agreement

Since the seminal paper of Lomas, Gong, Saltzer, and Needham [12], many password-based key establishment schemes have been proposed. Some of the more recent password-based proposals include the group key agreement protocol of Bresson, Chevassut, and Pointcheval [4] derived from an earlier protocol [5]; the two-round key agreement protocol without key confirmation of Lee, Hwang, and Lee [11] and the group key agreement of Dutta and Barua [9][3]; and the Diffie-Hellman key exchange protocols of Byun and Lee [7][4]. However, to the best of our knowledge, much less attention has been devoted to password-based protocols

[3] Both protocols have recently been broken by Abdalla *et al.* [1].

[4] Tang and Chen [15] demonstrate that both protocols are vulnerable to several security problems.

in a group setting. In the next section, we present our proposed password-based authenticated group key agreement protocol.

2.2 Our Proposed Protocol

Let U_i $(1 \leq i \leq n, 3 \leq n)$ denote a set of participants sharing a secret password, π, selected from a password set, $\mathcal{PW}$. Each participant, U_i, possesses an associated identity, ID_i. We let ℓ be the security parameter. In our protocol, the following system parameters are public.

1. Two large prime numbers p and q, where $p = 2q + 1$.
2. Three collision-resistant one-way hash functions, $\mathcal{H}_0$, $\mathcal{H}_1$, and $\mathcal{H}_2$, where $\mathcal{H}_0 : \{0,1\}^* \rightarrow Z_p$, $\mathcal{H}_1 : \{0,1\}^* \rightarrow \{0,1\}^\ell$, $\mathcal{H}_2 : \{0,1\}^* \rightarrow \{0,1\}^\ell$.

We throughout assume that U_1 is the protocol initiator. Prior to the protocol execution, U_i $(1 \leq i \leq n)$ computes $g = \mathcal{H}_0(\pi||ID_u||x) \bmod p$, where $ID_u = ID_1||ID_2||\cdots||ID_n$ and $x \geq 0$ is the smallest integer that makes g a generator of a multiplicative subgroup G of order q in $GF(p)^*$.

In the protocol execution, the indices of the user names and the values exchanged between users are taken modulo n; and U_i $(1 \leq i \leq n)$ performs the following steps.

Stage 1: Message transfer and authentication.

1. U_i chooses a random s_i $(0 \leq s_i \leq q-1)$, and broadcasts $Z_i = g^{s_i}$.
2. After receiving every Z_j $(1 \leq j \leq n, j \neq i)$, U_i verifies that none of them equals 1. If the check succeeds, U_i computes and broadcasts $A_{i,i-1}$ and $A_{i,i+1}$, where $Z = Z_1||Z_2||\cdots||Z_n$, $A_{i,i-1} = \mathcal{H}_1(i||i-1||Z||g^{s_{i-1}s_i}||g||ID_u)$, and $A_{i,i+1} = \mathcal{H}_1(i||i+1||Z||g^{s_{i+1}s_i}||g||ID_u)$.
3. After receiving every $A_{j,j-1}$ and $A_{j,j+1}$ $(1 \leq j \leq n, j \neq i)$, U_i verifies the received values of $A_{i-1,i}$ and $A_{i+1,i}$ by recomputing them using s_i, Z, and the stored values of Z_{i-1} and Z_{i+1}. If the checks succeed, then U_i continues with the next stage. Otherwise, the protocol execution is terminated and a notification of failure broadcasted.

Stage 2: Key agreement and key confirmation.

1. U_i computes and broadcasts $X_i = (\frac{Z_{i+1}}{Z_{i-1}})^{s_i}$.
2. After receiving every X_j $(1 \leq j \leq n, j \neq i)$, U_i computes the keying material M_i as:

$$M_i = (Z_{i-1})^{ns_i} \prod_{j=0}^{n-2} (X_{i+j})^{n-1-j} = g^{ns_{i-1}s_i} \prod_{j=0}^{n-2} (\frac{g^{s_{i+j}s_{i+j+1}}}{g^{s_{i+j-1}s_{i+j}}})^{n-1-j}$$

$$= \prod_{j=1}^{n} g^{s_j s_{j+1}} = g^{\sum_{j=1}^{n} s_j s_{j+1}}$$

U_i then broadcasts its key confirmation message C_i, where

$$C_i = \mathcal{H}_1(i||ID_u||Z||A||X||M_i||g),\ X = X_1||X_2||\cdots||X_n, \text{ and}$$

$$A = A_{1,2}||A_{2,3}||\cdots||A_{n,1}||A_{1,n}||A_{2,1}||A_{3,2}||\cdots||A_{n,n-1}.$$

3. After receiving C_j $(1 \leq j \leq n, j \neq i)$, U_i checks whether the following equation holds:

$$C_j \stackrel{?}{=} \mathcal{H}_1(j||ID_u||Z||A||X||M_i||g).$$

If the check succeeds, U_i computes its session key as:

$$K_i = \mathcal{H}_2(ID_u||Z||A||X||M_i)$$

and concludes by computing a session identifier (SID). Note that the SID is defined to be the concatenation of the identities of all intended participants and the messages broadcast in every round of the ongoing protocol execution[5]. Otherwise, U_i terminates the protocol execution as a failure.

The above protocol is derived from the unauthenticated cyclic group key agreement protocol due to Burmester and Desmedt [6], which has been proven secure against a passive attacker under the Decisional diffie-Hellman (DDH) assumption. In our proposed protocol, the password is used to achieve authentication among participants. Note that U_i authenticates Z_{i-1} and Z_{i+1} in the first stage prior to computation and broadcasting of X_i. Without this authentication requirement, the protocol may be vulnerable to an offline dictionary attack.

3 Security Model and Security Analysis

We now describe the model in which we work, which is closely based on the model of Bresson *et al.* [4, 5]. The proposed model assumes that every protocol message will be broadcast to all the users. We then present a security proof for our proposed protocol in this security model.

3.1 Description of the Security Model

In the model, we denote the participants by U_i $(1 \leq i \leq n)$, each associated with a unique identity, ID_i. For any participant U_i, when the protocol is initiated, we say that a protocol instance of U_i is generated. In reality, when U_i starts a protocol execution, it knows some necessary information such as the identities of all the involved participants, the communication details, and the instance creation time. In this model, this necessary information is defined to be the participant instance identifier, which uniquely identifies the participant instance. If U_i is involved in an instance possessing an identifier id_i, we further define the participant instance to be an oracle $\Pi_i^{id_i}$, which is a probabilistic Turing machine processing the protocol messages on behalf of U_i. At any time, an oracle is in one of the following states:

[5] The SID is made public upon protocol completion, and the security of the protocol does not hinge on the difficulty of predicting a valid SID. In other words, anyone (including the attacker, $\mathcal{A}$) knows what a particular SID is.

- Active: the oracle is still waiting for inputs from other oracles, and the key agreement process has not finished.
- Accepted: the oracle has stopped and successfully generated the session key and a SID.
- Aborted: the oracle has stopped as a failure and has output an error message.

In the model, there exist a passive attacker and an active attacker. A passive attacker only eavesdrops on the communications, while an active attacker is allowed to intercept, delete, delay, and/or fabricate any messages at will. The security of a protocol is modelled by a series of games played between a challenger and the attacker. The challenger simulates the view of the attacker and answers all the queries of the following types asked by the attacker, $\mathcal{A}$.

- A Create query allows the attacker to initiate a new protocol instance.
- Upon receiving a Send query, the oracle will compute a response according to the protocol specification and a decision on whether to accept or reject, and return them to $\mathcal{A}$. If the oracle has either accepted with some session key or terminated, then this will be made known to $\mathcal{A}$.
- The Reveal query captures the notion of known key security. Upon receiving such a query, an oracle that has accepted and holds some session key, will return the session key to $\mathcal{A}$.
- The Test query is the only oracle query that does not correspond to any of $\mathcal{A}$'s abilities. If the oracle has accepted with some session key and is being asked a Test query, then, depending on a randomly chosen bit b, $\mathcal{A}$ is given either the actual session key or a session key drawn randomly from the session key distribution.

The definition of partnership is used in the definition of security to restrict the attacker's Reveal queries to accepted oracles that are not partners of the oracle whose key the attacker is trying to guess. Definition 1 describes the partnership definition.

Definition 1. *Two oracles, $\Pi_i^{id_i}$ and $\Pi_j^{id_j}$, for any $1 \leq i, j \leq n$ and $i \neq j$, are partners if and only if they accept and possess the same SID.*

We define a function Γ, which, on the input of an accepted oracle $\Pi_i^{id_i}$, returns $\Gamma(\Pi_i^{id_i}) = \sum_{j=1,j\neq i}^{n} \Psi_j(\Pi_i^{id_i})$, where $\Psi_j(\Pi_i^{id_i}) = 1$ if $\Pi_i^{id_i}$ has a partner oracle $(\Pi_j^{id_j})$, otherwise $\Psi_j(\Pi_i^{id_i}) = 0$. It is easy to see that the output of $\Gamma(\Pi_i^{id_i})$ equals the total number of participants that have at least one oracle partnered with $\Pi_i^{id_i}$.

Freshness is used to identify those session keys about which $\mathcal{A}$ ought not to know anything because $\mathcal{A}$ has not revealed any oracles that have accepted the key and has not corrupted any principals knowing the key. Definition 2 describes freshness, which depends on Definition 1.

Definition 2. *An oracle, $\Pi_i^{id_i}$, is said to be fresh if (1) $\Pi_i^{id_i}$ has accepted and has not been sent a* Reveal *query, and (2) no partner oracle of $\Pi_i^{id_i}$ (if such a partner exists) has been sent a* Reveal *query.*

3.1.1 Modelling Password Guessing Attacks

Without loss of generality, we assume that the password is chosen from the set $\mathcal{PW} = \{\pi_1, \pi_2, \cdots, \pi_m\}$, where π_i possesses the selection probability p_i and $p_i \leq p_j$ if $i > j$. It is easy to see that, after h tries and with no additional information, an attacker's advantage over the password (i.e. the largest probability the attacker can guess the correct password) is $\sum_{j=1}^{h+1} p_j$.

Password-based authenticated group key agreement protocols are designed to provide resilience against password guessing attacks, since passwords are usually of low entropy, and hence vulnerable to password guessing attacks. Password guessing attacks can be broadly categorised into online password guessing attacks and offline dictionary attacks. In an online password guessing attack, an attacker tries a guessed password by manipulating the inputs of one or more oracles. In an offline dictionary attack, an attacker exhaustively searches for the password by manipulating the inputs of one or more oracles. We remark that an offline dictionary attack presents a more subtle threat, as the adversary is able impersonate a legitimate party to initiate transactions without detection.

During the protocol execution, every oracle communicates with $n-1$ oracles. Hence, an attacker may try $n-1$ possible passwords by intervening in the inputs of only one oracle. Therefore, we regard a protocol to be secure if the attacker's advantage in guessing the "right" password is negligibly larger than the *evaluation probability* $\sum_{j=1}^{x(n-1)+1} p_j$ if x oracles aborted at the end of the following attack game. It should be noted that the *evaluation probability* $\sum_{j=1}^{x(n-1)+1} p_j$ can be replaced with any $\sum_{j=1}^{x(v)+1} p_j$ $(1 \leq v \leq n-1)$, where a smaller v means a stricter security requirement.

The attack game for modelling both kinds of password guessing attacks is carried out between the challenger and a polynomial-time attacker, $\mathcal{A}$, as follows:

Setup. The challenger generates a password, $\pi \in \mathcal{PW}$, and the public system parameters, *param*.

Challenge. The challenger runs $\mathcal{A}$ on the input of *param*. At some point, $\mathcal{A}$ terminates by outputting a guessed password π'. During its execution, $\mathcal{A}$ can make the following kinds of queries:

- $\mathsf{Create}(U)$, where U is any participant from the participant set.
- $\mathsf{Send}(\Pi, m)$, where Π is an active oracle and m is a message chosen by $\mathcal{A}$.
- $\mathsf{Reveal}(\Pi)$, where Π is an accepted oracle.

Suppose that there are x $(x < \frac{m}{n-1})$ aborted oracles at the end of the game. The attacker's advantage over the password in the game is defined to be $\mathsf{Adv}^{\mathcal{A}}(\pi) = \mathcal{F}(x, Pr(\pi = \pi'))$, where the function $\mathcal{F}$ is defined as follows: on the input of an integer a $(a < n)$ and a value b $(0 \leq b \leq 1)$, $\mathcal{F}(a, b)$ is computed as:

$$\mathcal{F}(a,b) = \begin{cases} 0, & \text{if } b \leq \sum_{j=1}^{a(n-1)+1} p_j \\ b - \sum_{j=1}^{a(n-1)+1} p_j, & \text{otherwise} \end{cases}$$

3.1.2 Modelling Attacks Against Key Authentication

The attack game against U_t $(1 \leq t \leq n)$ for key authentication is carried out between the challenger and a two-stage polynomial-time attacker $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ as follows:

Setup. The challenger generates a password $\pi \in \mathcal{PW}$ and the public system parameters *param*.

Phase 1. The attacker runs $\mathcal{A}_1$ on the input of *param*. $\mathcal{A}_1$ can make the following kinds of queries:

- $\mathsf{Create}(U)$, where U is any participant from the participant set.
- $\mathsf{Send}(\Pi, m)$, where Π is an active oracle and m is a message chosen by $\mathcal{A}_1$.
- $\mathsf{Reveal}(\Pi)$, where Π is an accepted oracle.

$\mathcal{A}_1$ terminates by making a $\mathsf{Test}(\Pi_t^{id_t})$ query, where $\Pi_t^{id_t}$ is a fresh oracle, and outputting some state information *state*.

Challenge. The challenger returns the output of $\mathsf{Test}(\Pi_t^{id_t})$.

Phase 2. The attacker runs $\mathcal{A}_2$ on the input of *state* and the output of the challenger. $\mathcal{A}_2$ can make the same kinds of query as those in Phase 1. But $\mathcal{A}_2$ is not allowed to make a Reveal query on the input of $\Pi_t^{id_t}$ or its partner oracle. $\mathcal{A}_2$ terminates by outputting a guess bit b'.

Suppose that there are x aborted oracles at the end of the game where $x < \frac{m}{n-1}$. The attacker's advantage in this game is defined to be $\mathsf{Adv}^{\mathcal{A}}(\mathsf{U_t}) = \mathcal{F}'(x, Pr(b = b'))$, where the function $\mathcal{F}'$ is defined as follows: on the input of an integer a $(a < n)$ and a value b $(0 \leq b \leq 1)$, $\mathcal{F}'(a, b)$ is computed as:

$$\mathcal{F}'(a,b) = \begin{cases} 0, & \text{if } b \leq \frac{1+\sum_{j=1}^{a(n-1)+1} p_j}{2} \\ b - \frac{1+\sum_{j=1}^{a(n-1)+1} p_j}{2}, & \text{otherwise} \end{cases}$$

3.1.3 Modelling Attacks Against Key Confirmation

The attack game against U_t $(1 \leq t \leq n)$ for key confirmation is carried out between the challenger and a two-stage polynomial-time attacker $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ as follows:

Setup. The challenger generates a password $\pi \in \mathcal{PW}$ and the public system parameters *param*.

Phase 1. The attacker runs $\mathcal{A}_1$ on the input of *param*. $\mathcal{A}_1$ can make the following kinds of queries:

- $\mathsf{Create}(U)$, where U is any participant from the participant set.
- $\mathsf{Send}(\Pi, m)$, where Π is an active oracle and m is a message chosen by $\mathcal{A}_1$.
- $\mathsf{Reveal}(\Pi)$, where Π is an accepted oracle.

$\mathcal{A}_1$ terminates by outputting an accepted oracle $\Pi_t^{id_t}$ and the state information *state*.

Challenge. The attacker continues running $\mathcal{A}_2$.

Phase 2. The attacker runs $\mathcal{A}_2$ on the input of *state*. $\mathcal{A}_2$ can make the same kinds of query as those in Phase 1. At some point, $\mathcal{A}_2$ terminates.

Suppose that, at the end of the game, there are x ($x < \frac{m}{n-1}$) aborted oracles and $\Pi_t^{id_t}$ has y partner oracles. The attacker's advantage in the game is defined to be $\mathsf{Adv}^{\mathcal{A}}(\mathsf{U_t}) = \mathcal{F}(x, Pr((y > n-1) \vee (\Gamma(\Pi_t^{id_t}) < n-1)))$.

Intuitively, if a protocol achieves key confirmation then it also guarantees mutual authentication which informally means that any legitimate accepted oracle confirms that the messages come from other legitimate oracles .

3.1.4 Security Definition

We first give a formal definition for negligible probability.

Definition 3. *The probability $P(\ell)$ is negligible if for any polynomial $f(\ell)$, where ℓ is the security parameter, there exists an integer N_f such that $P(\ell) \leq \frac{1}{f(\ell)}$ for all $\ell \geq N_f$.*

Informally, a secure authenticated key agreement protocol with key confirmation guarantees that only the legitimate oracle can possibly compute the session key. Any oracle that accepts can confirm that it has $n-1$ partner oracles which compute the same session key. A formal statement is as follows.

Definition 4. *A password-based authenticated group key agreement protocol with key confirmation is secure, if it satisfies the following requirements:*

1. *When the protocol is run in the presence of a probabilistic, polynomial-time (PPT) attacker, all partnered oracles compute the same session key.*
2. *An oracle computes a uniformly distributed session key regardless of the inputs from other oracles.*
3. *An active PPT attacker only has negligible advantage in the attack game modelling password guessing attacks.*
4. *An active PPT attacker only has negligible advantage in the attack game against U_t $(1 \leq t \leq n)$ for key authentication.*
5. *An active PPT attacker only has negligible advantage in the attack game against U_t $(1 \leq t \leq n)$ for key confirmation.*

3.2 Security Analysis

We assume that the DDH problem is hard, i.e. given a finite cyclic group G of prime order, a generator α of G, and group elements α^a and α^b, distinguish α^{ab} and α^c, where α^c is a random element in G. In our proof, $\mathcal{H}_0$, $\mathcal{H}_1$, and $\mathcal{H}_2$ are modelled as random oracles.

Theorem 1. *The proposed authenticated password-based group key agreement protocol is secure in the sense of Definition 4 in the random oracle model under the DDH assumption.*

Proof Sketch for Theorem 1. It is straightforward to verify that the proposed protocol satisfies the first and second requirements of Definition 4. Next we prove that the proposed protocol satisfies the remaining (three) requirements.

Claim. In the attack game modelling password guessing attacks, the attacker's advantage is negligible.

Proof. The attack game is simulated by the challenger as follows.

Setup. Given a security parameter, ℓ, the challenger generates a password, $\pi \in \mathcal{PW}$, and the public system parameters, $param = \{p, q, \mathcal{H}_0, \mathcal{H}_1, \mathcal{H}_2, ID_i (1 \leq i \leq n)\}$. The attacker simulates $\mathcal{H}_0$, $\mathcal{H}_1$, and $\mathcal{H}_2$ as follows.

- $\mathcal{H}_0$ queries: The challenger maintains a list of request message and hash value pairs. When receiving the attacker's request message, the challenger first checks its list to see whether the hash value for the request message has been queried. If the check succeeds, the challenger sends the stored value to the attacker. Otherwise, the challenger generates a random value and sends it to the attacker. In the meantime, the challenger stores the request message and hash value pair to the existing list.
- $\mathcal{H}_1$ and $\mathcal{H}_2$ queries: These two kinds of queries are simulated in exactly the same way as the $\mathcal{H}_0$.

Challenge. The attacker runs $\mathcal{A}$ on the input of $param$. At some point, $\mathcal{A}$ terminates by outputting a guessed password π'. During its execution, $\mathcal{A}$ can make the following kinds of queries:

- $\mathsf{Create}(U)$, , where U is any participant from the participant set.
- $\mathsf{Send}(\Pi, m)$, where Π is an active oracle and m is a message chosen by A_1.
- $\mathsf{Reveal}(\Pi)$, where Π is an accepted oracle.

At the end of the game, suppose that there are x aborted oracles. We analyse the following different cases in which the attacker may try a guessed password:

1. Suppose $\mathcal{A}$ acts passively without manipulating any oracle's input. In this case, $\mathcal{A}$ needs to compute some $g^{s_i s_{i+1}}$ or M_i $(1 \leq i \leq n)$, in order to test a guess. Computing $g^{s_i s_{+1}}$ based on g^{s_i} and $g^{s_{i+1}}$ is equivalent to solving the Computational Diffie-Hellman (CDH) problem[6] and the probability of computing M_i is also negligible based on the DDH assumption [6]. Therefore, in this case $\mathcal{A}$ can only succeed with a negligible probability.
2. Suppose $\mathcal{A}$ has not manipulated the input of any oracle in the first stage during its execution. In this case, $\mathcal{A}$ can only succeed with a negligible probability for similar reasons as in the first case.
3. Suppose $\mathcal{A}$ has manipulated the input of an oracle in the first stage during its execution. Without loss of generality, suppose $\mathcal{A}$ replaces Z_{i+1} sent to $\Pi_i^{id_i}$ $(1 \leq i \leq n)$ with Z'_{i+1}, where $Z'_{i+1} = (g')^s$, g' is computed based on a guessed password, and s is randomly chosen by $\mathcal{A}$. Then $\mathcal{A}$ postpones

[6] It is well known that CDH implies DDH.

forging $A_{i+1,i}$ until it obtains $A_{i,i+1}$. With $A_{i,i+1}$, $\mathcal{A}$ can test the guessed password by checking whether $A_{i,i+1} \stackrel{?}{=} h(i||i+1||Z'||(Z_i)^s||g'||ID_u)$ holds, where $Z' = Z_1||Z_2||\cdots||Z_i||Z'_{i+1}||Z_{i+2}||\cdots||Z_n$.
If $\pi' \neq \pi$, we can prove the following claims.

Claim. With the given information, $\mathcal{A}$ can only succeed in trying a different password with a negligible probability.

Proof. If $\mathcal{A}$ wishes to test another possible password, say π'', when its first guess is wrong ($\pi' \neq \pi$), then it must compute $(Z'_{i+1})^x$, where $x = \log_{g''}(Z_i)$ and g'' is computed based on π''. The computation can be abstracted as follows: Given $Z_i = (g'')^{x_1}$ and $Z'_{i+1} = (g'')^{x_2}$, where x_1 and x_2 are unknown, $\mathcal{A}$ computes $(g'')^{x_1 x_2}$. Based on the CDH assumption, $\mathcal{A}$ can only succeed with a negligible probability. □

Claim. $\Pi_i^{id_i}$ accepts with a negligible probability.

Proof. To forge a key authentication message for $\Pi_i^{id_i}$, the attacker needs to compute the value, $(Z'_{i+1})^{s_i}$, where $Z'_{i+1} = (g')^s$, $Z_i = g^{s_i}$, and $g' \neq g$. The computation can be abstracted as follows: Given $Z_i = (g)^{x_1}$ and $Z'_{i+1} = (g)^{x_2}$, where x_1 and x_2 are unknown, $\mathcal{A}$ computes $g^{x_1 x_2}$. Based on the CDH assumption, $\mathcal{A}$ can only succeed in forging an authentication message with a negligible probability. □

4. In order to exhaustively search for the password, $\mathcal{A}$ may also try to compute some M_i $(1 \leq i \leq n)$, which is computed by some oracle, say $\Pi_i^{id_i}$, and then re-computes $\Pi_i^{id_i}$'s key confirmation message C_i. Suppose $\Pi_i^{id_i}$ receives the messages $X_1, X_2, \cdots, X_{i-1}, X_i, \cdots, X_{n-1}, X_n$, where X_i is computed by itself. Based on the discussions in the third case, it is easy to see that the probability, that any of these messages is computed by a legitimate oracle using at least one forged message (note that X_j $(1 \leq j \leq n)$ is computed using two messages: Z_{j-1} and Z_{j+1}), is $\sum_{j=1}^{x+1} p_j$. As a result, the attacker can compute M_i with the probability $\sum_{j=1}^{x+1} p_j$ based on the results in [6].

Suppose that all the x aborted oracles resulting from the password guessing attacks, the attacker's advantage over the password is $\sum_{j=1}^{x+1} p_j + \mathcal{P}_1$, where $\mathcal{P}_1$ is negligible. As a result, in this game $\mathcal{A}$'s advantage $\mathsf{Adv}^{\mathcal{A}}(\pi) = \mathcal{F}(x, \sum_{j=1}^{x+1} p_j + \mathcal{P}_1)$ is also negligible and we have proved this claim. □

Claim. In the attack game against U_t $(1 \leq t \leq n)$ for key confirmation, the attacker's advantage is negligible.

Proof. The attack game is simulated by the challenger as follows.

Setup. Given a security parameter, ℓ, the challenger generates a password, $\pi \in \mathcal{PW}$, and the public system parameters, $param = \{p, q, \mathcal{H}_0, \mathcal{H}_1, \mathcal{H}_2, ID_i (1 \leq i \leq n)\}$. The attacker simulates $\mathcal{H}_0$, $\mathcal{H}_1$, and $\mathcal{H}_2$ as follows.

- $\mathcal{H}_0$ queries: The challenger maintains a list of request message and hash value pairs. When receiving the attacker's request message, the challenger first checks its list to see whether the hash value for the request message has been queried. If the check succeeds, the challenger sends the stored value to the attacker. Otherwise, the challenger generates a random value and sends it to the attacker. In the meantime, the challenger stores the request message and hash value pair to the existing list.
- $\mathcal{H}_1$ and $\mathcal{H}_2$ queries: These two kinds of queries are simulated in exactly the same way as the $\mathcal{H}_0$.

Phase 1. The attacker runs $\mathcal{A}_1$ on the input of *param*. $\mathcal{A}_1$ can make the following kinds of queries:

- $\mathsf{Create}(U)$, , where U is any participant from the participant set.
- $\mathsf{Send}(\Pi, m)$, where Π is an active oracle and m is a message chosen by $\mathcal{A}_1$.
- $\mathsf{Reveal}(\Pi)$, where Π is an accepted oracle.

$\mathcal{A}_1$ terminates by outputting an accepted oracle $\Pi_t^{id_t}$ and some state information *state*.

Challenge. The attacker continues running $\mathcal{A}_2$.

Phase 2. The attacker runs $\mathcal{A}_2$ on the input of *state*. $\mathcal{A}_2$ can make the same kinds of query as those in Phase 1. At some point, $\mathcal{A}_2$ terminates.

At the end of the game, suppose that $\Pi_t^{id_t}$ has y partner oracles and there are x aborted oracles. Next, we compute the probability of $y > n-1$ and $\Gamma(\Pi_t^{id_t}) < n-1$, respectively.

- In the proposed protocol, every participant U_i $(1 \leq i \leq n)$ generates and broadcasts $Z_i = g^{s_i}$, where s_i is a random number, outlined in the first stage. Hence, the probability that $y > n-1$ happens is negligible because Z_i is part of the SID.
- As required by the protocol, $\Pi_t^{id_t}$ should have succeeded in verifying $n-1$ key confirmation messages (which is computed based on SID and g (or π)) before it accepts in Phase 1. Suppose that $\Gamma(\Pi_t^{id_t}) < n-1$, then the attacker has forged at least one of the confirmation messages received by $\Pi_t^{id_t}$, which equivalently means the attacker has guessed the correct password. However, it is straightforward to verify that $\mathcal{F}(x, Pr(\Gamma(\Pi_t^{id_t}) < n-1))$ is negligible based on the first claim.

Combining the above two possibilities, in this game the attacker's advantage $\mathcal{F}(x, Pr(\Gamma(\Pi_t^{id_t}) < n-1) + Pr(y > n-1))$ is negligible. As a result, we have proved this claim. □

Claim. In the attack game against U_t $(1 \leq t \leq n)$ for key authentication, the attacker's advantage is negligible.

Proof. The attack game is simulated by the challenger as follows.

Setup. Given a security parameter, ℓ, the challenger generates a password, $\pi \in \mathcal{PW}$, and the public system parameters, $param = \{p, q, \mathcal{H}_0, \mathcal{H}_1, \mathcal{H}_2, ID_i (1 \leq i \leq n)\}$. The attacker simulates $\mathcal{H}_0$, $\mathcal{H}_1$, and $\mathcal{H}_2$ as follows.

- $\mathcal{H}_0$ queries: The challenger maintains a list of request message and hash value pairs. When receiving the attacker's request message, the challenger first checks its list to see whether the hash value for the request message has been queried. If the check succeeds, the challenger sends the stored value to the attacker. Otherwise, the challenger generates a random value and sends it to the attacker. In the meantime, the challenger stores the request message and hash value pair to the existing list.
- $\mathcal{H}_1$ and $\mathcal{H}_2$ queries: These two kinds of queries are simulated in exactly the same way as the $\mathcal{H}_0$.

Phase 1. The attacker runs $\mathcal{A}_1$ on the input of *param*. $\mathcal{A}_1$ can make the following kinds of queries:

- $\mathsf{Create}(U)$, , where U is any participant from the participant set.
- $\mathsf{Send}(\Pi, m)$, where Π is an active oracle and m is a message chosen by $\mathcal{A}_1$.
- $\mathsf{Reveal}(\Pi)$, where Π is an accepted oracle.

$\mathcal{A}_1$ terminates by making a $\mathsf{Test}(\Pi_t^{id_t})$ query, where $\Pi_t^{id_t}$ is a fresh oracle, and outputting some state information *state*.

Challenge. The challenger returns the output of $\mathsf{Test}(\Pi_t^{id_t})$.

Phase 2. The attacker runs $\mathcal{A}_2$ on the input of *state* and the output of the challenger. $\mathcal{A}_2$ can make the same queries as those in Phase 1. But $\mathcal{A}_2$ is not allowed to make a Reveal query on the input of $\Pi_t^{id_t}$ or its partner oracle. $\mathcal{A}_2$ terminates by outputting a guess bit b'.

At the end of the game, suppose that there are x aborted oracles. We have the following observations:

- Let $\mathcal{A}_S$, N_P, and $\overline{N_P}$ represent the events $\mathcal{A}$ succeeds, $\Gamma(\Pi_t^{id_t}) = n-1$, and $\Gamma(\Pi_t^{id_t}) \neq n-1$, the following equation holds:

$$\begin{aligned} Pr(\mathcal{A}_S) &= Pr(b = b'|N_P)Pr(N_P) + Pr(b = b'|\overline{N_P})Pr(\overline{N_P}) \\ &\leq Pr(b = b'|N_P)Pr(N_P) + Pr(\overline{N_P}) \end{aligned}$$

- The attacker's advantage is computed as follows:

$$\mathsf{Adv}^{\mathcal{A}}(\mathsf{U_t}) = \begin{cases} 0, & \text{if } Pr(\mathcal{A}_S) \leq \frac{1+\sum_{j=1}^{x(n-1)+1} p_j}{2} \\ Pr(\mathcal{A}_S) - \frac{1+\sum_{j=1}^{x(n-1)+1} p_j}{2}, & \text{otherwise} \end{cases}$$

- Based on the second claim, we know that either $Pr(\overline{N_P}) \leq \sum_{j=1}^{x(n-1)+1} p_j$ holds or $Pr(\overline{N_P}) - \sum_{j=1}^{x(n-1)+1} p_j$ is negligible.

Suppose that the attacker's advantage $\mathsf{Adv}^{\mathcal{A}}(\mathsf{U_t})$ in this game is non-negligible. Based on the above observations it is easy to see that $|Pr(b = b'|N_P) - \frac{1}{2}|$ is non-negligible, in contradiction of the security results in [6]. As a result, we have proved the claim. □

As a result, we have proved that the proposed protocol satisfies all the five security requirements so that it is secure under our definition. □

Conclusion of Proof for Theorem 1. As we have seen in the security proof, if n participants take part in the protocol execution $\mathcal{A}$ can try $2n$ possible passwords by simultaneously manipulating the messages sent to every participant. Moreover, $\mathcal{A}$ can mount the online password guessing attacks automatically by running a program. Consequently, an attacker can leverage this vulnerability, e.g., launches a successful online dictionary attack against the protocol when the group size n becomes very large. We remark that other password-based group key agreement protocols [5, 7, 11] also suffer from this security vulnerability.

4 Revisiting Online Password Guessing Attacks

As we have shown in the previous section, our proposed protocol suffers from automated online password guessing attacks (similar weaknesses can be found in other published password-based group key agreement protocols). In this section, we enhance the proposed protocol to make it more robust by using a CAPTCHA technique and federated verification provided by the server S.

4.1 CAPTCHA

Recent research on CAPTCHA techniques [2] suggests that CAPTCHA has many practical applications, e.g., sites such as Yahoo Mail and Hotmail use distorted image recognition to prevent automated account registration; online polls; search engine bots. CAPTCHA technique has also been deployed in combating spam and worms and preventing dictionary attacks [13]. Although distorted image recognition is most widely used, CAPTCHA means more than this – it is a test (any test) that can be automatically generated. Such tests generated are trivial to human but computationally hard for computer programs. Ahn *et al.* [2] studied two kinds of artificial intelligence (AI) problems and proposed several methods to construct CAPTCHA. However, we should note that the security of these AI problems, the foundation of the CAPTCHA, is based on the state of the art in pattern recognisation, and thus, heuristic. Several researchers have recently developed efficient methods to defeat some specific CAPTCHA. Despite this setback, CAPTCHA is still very popular.

4.2 Description of the Enhanced Protocol

In this enhanced protocol, we assume that S possesses a signature key pair (p_k, s_k) generated by a signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ which is unforgeable under an adaptive chosen message attack (defined in [3]), and a public-key encryption key pair (p'_k, s'_k) generated by a IND-CCA2 secure public-key encryption scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ [14]. We also suppose that S can generate and verify CAPTCHA. Other parameters are generated in the same way as in the original protocol.

In a protocol execution, S and U_i $(1 \leq i \leq n)$ perform as follows.

Stage 0: Execution of CAPTCHA

1. U_i sends a random number, $r_i \in \{0,1\}^\ell$, to S.
2. S sends a new CAPTCHA, pz, to U_1.

3. U_1 solves the CAPTCHA, pz, and sends the solution, sz, to S.
4. S checks whether sz is the correct solution of pz. If the check succeeds, S selects a random $s_{id} \in \{0,1\}^\ell$, and then broadcasts s_{id}, $R = r_1||r_2||\cdots||r_n$, and a signature $\mathsf{Sig}_{s_k}(0||ID_u||R||s_{id})$.
5. After receiving the messages, U_i first verifies the signature. If the verification succeeds, U_i continues; otherwise, U_i aborts the protocol execution.

Stage 1: Message transfer and authentication

1. U_i chooses a random s_i $(0 \leq s_i \leq q-1)$, and broadcasts $Z_i = g^{s_i}$.
2. After receiving every Z_j $(1 \leq j \leq n, j \neq i)$, U_i verifies that none of them equals 1. If the check succeeds, U_i selects two random number $r_i', r_i'' \in \{0,1\}^\ell$, and sends

$$E_i = \mathsf{Enc}_{p_k'}(ID_i||A_{i,i-1}||A_{i,i+1}||A_{i-1,i}||A_{i+1,i}||r_i'||r_i''||s_{id})$$

   to S, where Z and $A_{i,i-1}, A_{i,i+1}, A_{i,i-1}, A_{i,i+1}$ $(1 \leq i \leq n)$ are defined in the same way as in the original protocol.
3. After receiving every E_i $(1 \leq i \leq n)$, S first decrypts them to check whether s_{id} is in its memory, and then checks whether $A_{i,i+1}$ (provided by U_i and U_{i+1}) and $A_{i,i-1}$ (provided by U_{i-1} and U_i) are correct, for all $1 \leq i \leq n$. If the checks succeed, S broadcasts a signature $\mathsf{Sig}_{s_k}(1||ID_u||R||R'||s_{id})$, where $R' = r_1'||r_2'||\cdots||r_n'$. Otherwise S broadcasts a failure message. Simultaneously, S deletes s_{id} and the related data.
4. If receiving a failure message, U_i aborts the protocol execution. Otherwise, after receiving R' and the signature from S, U_i verifies whether the signature is correct. If the verification succeeds, U_i continues the steps in next stage; otherwise it aborts the protocol execution.

Stage 2: Key agreement and key confirmation
This stage is similar to the original protocol presented in Section 2.2, except that the parameter A, used in the computation of key confirmation message and the session key, is substituted by $R||R'||s_{id}$.

4.3 Security Analysis of the Enhanced Protocol

Differences between our enhanced and our original versions include the following: (1) an additional stage 0 is added in our enhanced version, and (2) all the authentication messages in the first stage of our enhanced version are verified by the server, S. We now argue that these two features are effective countermeasures for online password guessing attacks, without compromising the overall security of our original protocol.

1. Assuming the hardness of the CAPTCHA, the possibilities of automated online password guessing attacks is now significantly reduced, since we assume that only a human being is intelligent enough to pass the CAPTCHA. In other words, we are assured that the initiator involved in every initialisation

of the protocol execution, is actually a human being and not some computer programs. Moreover, assuming the security of the underlying signature scheme, we can now verify that U_1 has successfully solved a CAPTCHA if, and only if, we receive a valid signature. Therefore, an automated program is prevening from mounting automated online password guessing attacks if the underlying CAPTCHA used is sufficiently strong.

Another alternative (without using the CAPTCHA) is to require the initiator to solve a computational puzzle [10] prior to the protocol execution with each individual protocol participant. However, such an approach is computationally intensive and it is difficult to adjust the hardness of the puzzles if the participants have very different computing power.

2. The federated verification ensures that $\mathcal{A}$ can only test at most one password in any single protocol execution. Assuming the security of the underlying public encryption scheme, $\mathcal{A}$ cannot simultaneously mount the attack by intervening in the protocol execution because $\mathcal{A}$ is unable to recover the encrypted messages or try its guess with the encrypted messages. Therefore, to test the guess, $\mathcal{A}$ must submit the forged messages to S for verification. However, S will definitely fail the verification and provide no other information about the failure if $\mathcal{A}$ attempts more than one guess at any one time.

   The participants can indirectly verify the authentication messages by verifying the signature from S because of the presence of r_i's in the signature but they cannot identify which authentication message goes wrong in the event that the verification fails.

5 Conclusions

We have proposed a password-based authenticated group key agreement protocol, and proved it secure in a variant of the model by Bresson *et al.* [4]. We then discussed the limitations of our protocol, followed by an enhanced version of the protocol using CAPTCHA. Consequently, our enhanced version is more robust against online password guessing attacks.

Acknowledgements. The authors would like to thank Chris J. Mitchell for his invaluable and constructive feedbacks, and the anonymous reviewers for their insightful comments.

References

1. M. Abdalla, E. Bresson, O. Chevassut, and D. Pointcheval. Password-based Group Key Exchange in a Constant Number of Rounds. In *PKC 2006*, LNCS. Springer-Verlag, 2006. To Appear.
2. L. Ahn, M. Blum, N. Hopper, and J. Langford. CAPTCHA: Using hard AI problems for security. In *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 294–311. Springer-Verlag, 2003.
3. M. Bellare and G. Neven. Transitive Signatures Based on Factoring and RSA. In *ASIACRYPT 2002*, volume 2501 of *LNCS*, pages 397–414. Springer-Verlag, 2002.

4. E. Bresson, O. Chevassut, and D. Pointcheval. Group Diffie–Hellman Key Exchange Secure Against Dictionary Attacks. In *ASIACRYPT 2002*, volume 2501 of *LNCS*, pages 497–514. Springer-Verlag, 2002.
5. E. Bresson, O. Chevassut, D. Pointcheval, and J.-J. Quisquater. Provably Authenticated Group Diffie–Hellman Key Exchange. In *ACM CCS 2001*, pages 255–264, 2001.
6. M. Burmester and Y. Desmedt. A Secure and Efficient Conference Key Distribution System. In A. D. Santis, editor, *Pre-Proceedings of EUROCRYPT '94*, LNCS, pages 279–290, 1994.
7. J. Byun and D. Lee. N-Party Encrypted Diffie–Hellman Key Exchange Using Different Passwords. In *ACNS 2005*, volume 3531 of *LNCS*, pages 75–90. Springer-Verlag, 2005.
8. N. Daswani, H. Garcia-Molina, and B. Yang. Open Problems in Data-Sharing Peer-to-Peer Systems. In *ICDT 2003*, volume 2572 of *LNCS*, pages 1–15. Springer-Verlag, 2002.
9. R. Dutta and R. Barua. Password-based Encrypted Group Key Agreement. *International Journal of Network Security*, 3(1):30–41, 2006.
10. A. Juels and J. Brainard. Client Puzzles: A Cryptographic Defense Against Connection Depletion. In *NDSS 1999*, pages 151–165, 1999.
11. S. M. Lee, J. Y. Hwang, and D. H. Lee. Efficient Password-Based Group Key Exchange. In *TrustBus 2004*, volume 3184 of *LNCS*, pages 191–199. Springer-Verlag, 2004.
12. T. Lomas, L. Gong, J. Saltzer, and R. Needham. Reducing Risks from Poorly Chosen Keys. *ACM SIGOPS Operating Systems Review*, 23(5):14–18, 1989.
13. B. Pinkas and T. Sander. Securing Passwords Against Dictionary Attacks. In *ACM CCS 2002*, pages 161–170. ACM Press, 2002.
14. C. Rackoff and D. R. Simon. Non-Interactive Zero-Knowledge Proof of Knowledge and Chosen Ciphertext Attack. In *CRYPTO 1991*, volume 576 of *LNCS*, pages 433–444. Springer-Verlag, 1991.
15. Q. Tang and L. Chen. Weaknesses in Two Group Diffie–Hellman Key Exchange Protocols. Cryptology ePrint Archive: Report 2005/197, 2005.