Practical Authenticated Key Agreement using Passwords

Taekyoung Kwon

School of Computer Engineering, Sejong University, Seoul 143-747, Korea
E-mail: tkwon@sejong.ac.kr

Abstract. Due to the low entropy of human-memorable passwords, it is not easy to conduct password authenticated key agreement in a secure manner. Though there are many protocols achieving this goal, they may require a large amount of computation specifically in the augmented model which was contrived to resist server compromise. Our contribution in this paper is two fold. First, we propose *a new practical password authenticated key agreement protocol* that is efficient and generic in the augmented model. Our scheme is considered from the practical perspective (in terms of efficiency) and is provably secure under the Diffie-Hellman intractability assumptions in the random-oracle model. Our second contribution is more realistic and generic; *a conceptually simple but novel password guessing attack* which can be mounted on every three-pass password-based protocol unless care is taken in both the design and implementation phases. This is due to the server's failure to synchronize multiple simultaneous requests. Experimental results and possible prevention methods are also discussed.

1 Introduction

User authentication is necessary for the typical case that a human being resides as a client and tries to log on to a remote server machine. The server must be able to determine the user's identity reliably over a public or private channel. Password authentication is one of such methods, in which simply the user memorizes a (short) password while the server maintains a user profile that associates the user name and the password verifying information. The intrinsic problem with this method is the memorable password, associated with each user, has low entropy, so that it is not easy to protect the password information against the notorious password guessing attacks by which attackers could search the relatively small space of human-memorable passwords.

Since a pioneering method that resists the password guessing attacks was introduced to cryptographic protocol developers [24], there has been a great deal of work for password authenticated key agreement, preceded by EKE [5], on the framework of Diffie-Hellman [10]. Readers are referred to [15] for complete references. Compared to the typical authenticated key agreement, the password-based schemes are more expensive due to the low entropy of passwords, specifically in the augmented model which was contrived to resist server compromise. Provable security is important but tends to make the schemes harder to be practical in some cases. From the theoretical perspective, several methods that are much more expensive but provably secure in the standard model, were presented [12, 18, 19]. From the practical perspective, the practice-oriented security models are applied for examining the security of protocols [1–3, 7]. For example, EKE2 and AuthA are provably secure in both the random oracle and ideal cipher models [3, 4, 8], while PAK and PAK-Z (that improves the efficiency of PAK-X impressively by specifying a generic digital signature) are in the random oracle model [7, 25, 26]. However, it is (arguably) still expensive to assume ideal ciphers or digital signatures along with many costly operations on them, while PAK-Y is reasonably efficient with Schnorr signature in terms of computational costs [4, 26, 30].

At present, SPEKE [16], SRP [32], PAK [26], and AMP [21] are being discussed by the IEEE P1363 Standard Working Group and more recently by the ISO/IEC JTC 1/SC 27 group as practical protocols for standardization on password-based public key cryptographic techniques [13, 14]. Among them, PAK and SPEKE are 'three-pass' protocols, while AMP and SRP are 'four-pass' protocols. The standardization work is valuable in many aspects; for instance, a new attack called the 'two-for-one' guessing attack[1] against the four-pass protocols was found and resolved in the process [13, 31]. Any preference between three-pass and four-pass is still open for password-based protocols while typical authenticated key agreement such as STS and SIGMA is three-pass [11, 20].

In this paper, our contribution is two fold from the practical perspective.

1) An efficient three-pass password-based protocol in the augmented model
2) A generic password-guessing attack against three-pass protocols

A password-based protocol designed in the augmented model can resist server compromise. In other words, an adversary who compromised a password profile from a server cannot impersonate a user without launching dictionary attacks. For this additional property, the related protocols (for example, A-EKE, AMP, AuthA, B-SPEKE, PAK-Z, and SRP) are more expensive than those are not (for instance, EKE, EKE-2, SPEKE, and PAK) in the augmented model [6, 21, 4, 17, 26, 32]. We observe that the existing provably-secure schemes are still expensive in the augmented model in terms of the amount of computation, and that it is desirable to minimize the number of message passes and the size of message blocks for practice on expensive communication channels. So we design a new three-pass password-based protocol in the augmented model with both security and efficiency in mind. We achieve this goal interestingly by a composition under the careful observation of the existing schemes discussed by the IEEE P1363 Standard Working Group, say without losing the presumed level of security. We call the protocol TP-AMP and prove its security in the random oracle model.

On developing the new three-pass password-based protocol, we find a conceptually simple but novel password guessing attack which can be mounted on every three-pass password-based protocol by exploiting a small window of vulnerability resulting from a standard technique to resist on-line guessing attacks, say from counting the number of failed requests. Our attack is due to the server's failure to synchronize multiple simultaneous requests, and is unavoidable in three-pass protocols unless special care is taken in both the design and implementation phases. We call this attack a *many-to-many* (or *parallel*) guessing attack[2] because an active attacker can validate as many password guesses as (s)he makes server instances invoked concurrently, regardless of its upper limit of on-line guessing. A prototype of the proposed protocol is implemented to show how our attack works and is prevented. We first consider this attack and possible resolution in the literature.

This paper is organized as follows. In the following section, the so-called TP-AMP protocol (our first contribution) is presented. In Section 3, the many-to-many guessing attack (our second contribution) is described in more detail. In Section 4, security and efficiency of TP-AMP are discussed. Finally this paper is concluded in Section 5. Appendix A and B are necessary for providing a formal security argument in more details.

[1] An active attacker can validate two password guesses in one impersonation attempt. The first attack against SRP was discovered by D. Bleichenbacher in 2000, while the similar attack on AMP was by M. Scott [31]. However, both protocols were fixed to resist respective attacks by each original author [13].

[2] We first introduced this attack at IEEE P1363.2 meeting and also discussed a few names for it.

Table 1. Basic Notation

C	Client (User)	S	Server
π	Password	τ_C	Transformed password for C
$\leftarrow$	Derivation	$\stackrel{\mathrm{R}}{\leftarrow}$	Random selection
κ, ℓ	Security parameters	q	Prime of size κ
r	Integer co-prime to q	p	Prime of size ℓ such that $p = rq + 1$
$\mathbb{Z}_p^*$	Multiplicative group of p	$\mathbb{G}_q$	q-order subgroup of $\mathbb{Z}_p^*$
g, ζ	Generator of $\mathbb{G}_q$	h_i, H_i	Random oracles
α, β	Agreed values	sk_i	Session key

2 A Practical Protocol

2.1 Preliminaries

Our principal motivation comes from the fact that password-based protocols designed in the augmented model are much less efficient than those are not in that model, in terms of either computation or communication costs. When we regard PAK as a fundamental structure for three-pass protocols due to its simplicity and clarity, we can easily observe that its augmentation such as PAK-X, PAK-Y, and PAK-Z are far from its intrinsic nature and get much more complicated in the augmented model [7, 25, 26]. AMP and SRP show better performance in that model but in four passes [21, 32]. So, our basic idea is to make AMP squeezed into PAK or PAK augmented by AMP, since AMP is another protocol that can be computed very efficiently over various numerical groups [21]. However, a simple composition is not sufficient, and consequently we obtain a new practical protocol by more careful consideration on them.

The reason for choosing PAK rather than EKE2 is obviously that the former can formally be proved by postulating the random oracles only, while the latter requires the additional assumption of ideal cipher [7, 26, 3]. However, EKE2 or similar schemes that are proved sufficiently secure, can also be applied to constructing the practical augmented protocol in the way of our composition. In that sense, our construction is quite generic.

In Table 1, we enumerate the notation, in part, to be used in the remaining of this paper. Additional ones will be self-contained in each part of this paper. Let κ be a general security parameter (say 160 bits) and ℓ be a special security parameter for public keys (1024 or 2048 bits). A client C and a server S should agree on algebraic parameters[3] related to Diffie-Hellman key agreement such as p, q, and g. Define $\bar{\mathbb{G}}_q = \{g^x \bmod p | x \in \mathbb{Z}_q^*\}$ where $|\bar{\mathbb{G}}_q| = q - 1$. Let us often omit 'mod p' from the expressions that are obvious in $\mathbb{Z}_p^*$. Let $\{0,1\}^*$ denote the set of finite binary strings and $\{0,1\}^n$ the set of binary strings of length n. We then define random oracles such that h_i: $\{0,1\}^* \rightarrow \{0,1\}^\kappa$ and H_i: $\{0,1\}^* \rightarrow \{0,1\}^\ell$. Note that $H_i(\cdot)$ is a specific random oracle which outputs in a pre-defined subgroup ($\mathbb{G}_q$) only. Their practical instances are defined as $h_i(\cdot) = h(i, \cdot, i)$ and $H_i(\cdot) = (h(i, \cdot, i))^{\frac{p-1}{q}} \mod p$ or $H_i(\cdot) = \zeta^{h(i,\cdot,i)} \mod p$ where $h(\cdot)$ is a strong one-way hash function and ζ is another generator of $\mathbb{G}_q$. Let $\mathsf{ACCEPTABLE}(\cdot)$ denote an acceptable function which may return true if its pre-image satisfies the given security properties, as defined in Section 2.3. Readers who are not familiar with the legacy protocols, are referred to the previous work of [7, 13, 21, 25, 26].

[3] In spite that PAK, in general, does not require $\gcd(r, q)$=1 and only PAK-R requires it for further randomization, we recommend to use a *secure prime* such that each factor of r except 2 is of size at least κ or a *safe prime* such that $r = 2$ for $p = rq + 1$ as discussed in [21, 23, 32, 29]. They satisfy $\gcd(r, q)$=1. Specifically, we observe that TP-AMP shows the best performance with a secure prime, while PAK-Y does with a safe prime and "arbitrarily" smaller exponents [28].

$\mathsf{client}[C,\pi]$		$\mathsf{server}[C,\tau_C(\gamma',\nu)]$ on S
		$\langle \gamma' = \gamma^{-1} \bmod p, \nu = g^u \bmod p \rangle$
$x \xleftarrow{\mathrm{R}} \mathbb{Z}_q^*$		
$\gamma \leftarrow H_0(C,\pi)$		
$m \leftarrow g^x\gamma \bmod p$		
	$\xrightarrow{C,m}$	
		abort if $\neg\mathsf{ACCEPTABLE}(C,m)$
		$y \xleftarrow{\mathrm{R}} \mathbb{Z}_q^*$
$\gamma' \leftarrow \gamma^{-1} \bmod p$		$\mu \leftarrow \nu^y \bmod p$
$u \leftarrow h_1(C,\pi)$		$\beta \leftarrow (m\gamma' g^m)^y \bmod p$
$w \leftarrow u^{-1}(x+m) \bmod q$		$k_1 \leftarrow h_2(C,S,m,\mu,\beta,\gamma')$
	$\xleftarrow{\mu,k_1}$	
$\alpha \leftarrow \mu^w \bmod p$		
$k_1' \leftarrow h_2(C,S,m,\mu,\alpha,\gamma')$		
abort if $k_1 \neq k_1'$		
$k_2 \leftarrow h_3(C,S,m,\mu,\alpha,\gamma')$		$k_2' \leftarrow h_3(C,S,m,\mu,\beta,\gamma')$
	$\xrightarrow{k_2}$	
		abort if $k_2 \neq k_2'$ or time is out
$sk_C \leftarrow h_4(C,S,m,\mu,\alpha,\gamma')$		$sk_S \leftarrow h_4(C,S,m,\mu,\beta,\gamma')$

Fig. 1. TP-AMP (Three-Pass AMP Protocol)

2.2 Proposed Protocol - TP-AMP

TP-AMP stands for the Three-Pass Authenticated key agreement via Memorable Passwords and is depicted in Figure 1. Let us borrow the name AMP from [21] for our basic motivation.

Protocol Setup On the registration phase, a user chooses a name C and a password π while the server S saves user's profile $\langle C, \tau_C \rangle$ in its stable storage where $\gamma = H_0(C,\pi)$, $\gamma' = \gamma^{-1} \bmod p$, $u = h_1(C,\pi)$, $\nu = g^u$, and $\tau_C = \langle \gamma', \nu \rangle$. For convenience, S is assumed as an IP address of the server machine.

Protocol Run A user may type C and π into the client machine. The client (C on behalf of the user from now on) then chooses x at random from $\mathbb{Z}_q^*$ (not $\mathbb{Z}_p^*$), and computes γ in order to obtain $m = g^x\gamma$. The client sends ($\rightarrow$) a commitment message $\langle C, m \rangle$ to the server.

$$1.\ C \rightarrow S : C, g^x\gamma$$

After or before sending message 1, the client could compute γ' and the user's *amplified password* such that $w = u^{-1}(x+m) \bmod q$ by obtaining $u = h_1(C,\pi)$, and keeps them while waiting for message 2. In practice, we can hash m so that we have $q|h(m)$ with negligible probability.

Upon receiving message 1, the server should abort it if $\mathsf{ACCEPTABLE}$ (C,m) returns false. Otherwise, the server fetches $\langle C, \tau_C \rangle$ from its storage and chooses y at random from $\mathbb{Z}_q^*$ so as to obtain $\mu = \nu^y$. The server then computes $\beta \equiv (m\gamma' g^m)^y \equiv g^{(x+m)y} (\bmod\ p)$ and $k_1 = h_2(C,S,m,\mu,\beta,\gamma')$, and sends a challenge message $\langle \mu, k_1 \rangle$ to the client.

$$2.\ S \rightarrow C : \nu^y, h_2(C,S,m,\mu,\beta,\gamma')$$

After or before sending message 2, the server could compute $k_2' \leftarrow h_3(C, S, m, \mu, \beta, \gamma')$ and keeps it while waiting for message 3. The server should abort if time is run out.

Upon receiving message 2, the client raises μ to the amplified password so that $\alpha \equiv \mu^w \equiv g^{y(x+m)} (\mathrm{mod}\ p)$, and computes $k_1' = h_2(C, S, m, \mu, \alpha, \gamma')$. If k_1 is not equal to k_1', the client should abort this session. Otherwise, the client computes $k_2 = h_3(C, S, m, \mu, \beta, \gamma')$ and sends a response message k_2 to the server.

$$3.\ C \rightarrow S : h_3(C, S, m, \mu, \alpha, \gamma')$$

After or before sending message 3, the client could compute a session key such that $sk_C = h_4(C, S, m, \mu, \alpha, \gamma')$ and deletes any other ephemeral values.

Upon receiving message 3, the server should abort this session if k_2 is not equal to k_2'. Otherwise, the server should compute a session key such that $sk_S = h_4(C, S, m, \mu, \beta, \gamma')$ and deletes any other ephemeral values.

As a result, the client and the server could authenticate each other using the passwords and agree on the same session key $sk_C (= sk_S)$ because $\alpha \equiv \beta \equiv g^{(x+m)y} (\mathrm{mod} p)$.

2.3 Small Discussion

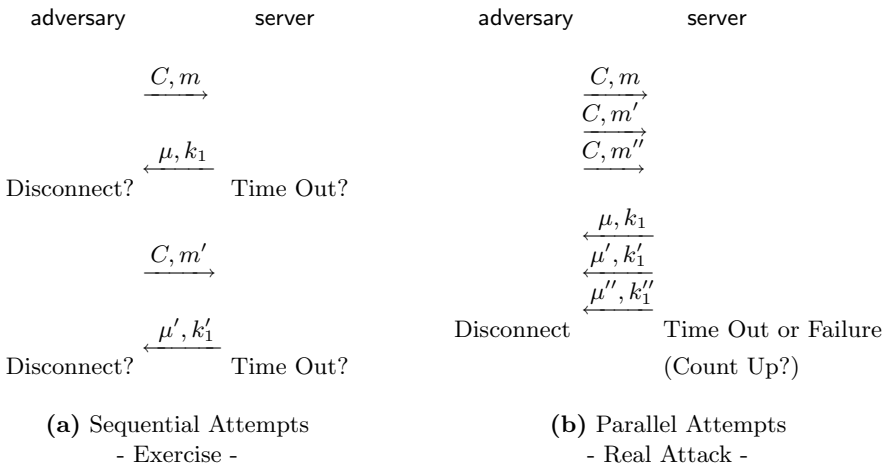
One can easily see that message 1 is extracted from PAK while message 2 and session key are motivated by AMP. This protocol performs simple computation in three passes and works in the augmented model where τ_C is defined as $\langle \gamma', \nu \rangle$. For efficiency, it would be better to hash m when we compute β and w, say $\beta = (m\gamma' g^{h(m)})^y$ and $w = u^{-1}(x + h(m)) \bmod q$ for a strong one-way hash function $h(\cdot)$. Also, we could modify $\beta = (m\gamma' g^{h(m,\mu)})^y$ and $w = u^{-1}(x + h(m, \mu)) \bmod q$ for easier security proof but a client cannot compute w before receiving μ in this case. For more efficiency, we recommend to use a secure prime for TP-AMP rather than a safe prime. Security and efficiency of the proposed protocol will be discussed in Section 4.

In the legitimate protocol run, g^x and ν^y are assumed not to be trivial values such as 0 and 1 as in the Diffie-Hellman relatives. We need to define a *failure count* that must be manipulated by the server and increased by one when $k_2 \neq k_2'$. The server should abort further requests of the client if the (subsequent) failure count exceeds its pre-defined limit, δ. This is a standard technique for resisting on-line guessing attacks. We also need to define the special function called ACCEPTABLE$(\cdot)$ since the server should abort when it returns false upon receiving $\langle C, m \rangle$. An example of the function follows:

ACCEPTABLE$(\cdot)$

INPUT: $\langle C, m \rangle$
OUTPUT:
Return *false*
  if C is being served by another instance; /* See Section 3 */
  else if the failure count of C is greater than or equal to its limit δ;
  else if $q|m$; /* Check if $m \notin Z_p^*$ only when hashing m before raising g */
Return *true* otherwise;

Note that the first condition (for resisting the many-to-many guessing attacks in the next section) can be considered in very flexible ways, for example, an IP address instead of C, and can be substituted by a more effective way in the future. This function is valid for authentication sessions only. Note also that $q|m$ means q divides m, but it might be enough to assure $m \in Z_p^*$ only when we hash m for β and w in the protocol.



(a) Sequential Attempts
- Exercise -

(b) Parallel Attempts
- Real Attack -

Fig. 2. Basic Concept of Many-to-Many Guessing Attacks

3 Many-to-Many Guessing

3.1 A Real World Attack

It is widely recognized that three-pass (say, smaller-pass) protocols are favorable to the channel efficiency for authenticated key agreement. However, care must be taken for password authenticated key agreement in a practical sense.

Let us glance over Theorem 1, in advance, that is introduced in Section 4 and proved in Appendix B. There exists an adversarial advantage that is bounded by $\frac{q_{se}}{N}$. The similar results can be found from the closely related work [3, 8, 26]. These advantages imply that the adversary is reduced to a simple online guessing attacker that can easily be detected and prevented from exceeding the pre-defined limit, δ, on the number of sequential on-line trials allowed by the server's policy. For example, an adversary posing as a user C sends an arbitrary message $\langle C, m\rangle$ to the server, based on her guessed password. The server may respond with $\langle \mu, k_1\rangle$ in the three-pass protocols while only μ in the four-pass protocols. Then, the adversary is assumed to check her guess with probability bounded by $\frac{q_{se}}{N}$ under the limit δ in three-pass protocols. Is this standard assumption really true?

Unfortunately, the answer is No! This classical prevention method can be fooled out of making the adversarial advantage much larger and in some cases disclosing a password, in a surprisingly simple way. Figure 2 depicts the possible bad events. Our attack is motivated from the fact that the server is typically implemented as a multi-threaded or multi-process application for handling many user requests simultaneously, and that the three-pass password-based protocol is not an exception. As summarized in Figure 2-(a), the adversary is able to exercise the real attack (that is described in Figure 2-(b)), for example, in order to approximate the maximum amount of time the server may wait for the third message k_2. The adversary then starts simultaneous authentication sessions, which the server processes independently in separate threads, and in that amount of time, is able to drive many different initiating messages based on different password guesses concurrently to the server. The adversary may get as many replies as allowed in that time boundary, by exceeding δ obviously. Figure 2-(b) abbreviates this idea. It could be a real world attack from the automated (and multi-threaded) adversary. The server instances must respond to each request and wait for the replies k_2

from the adversary who can even disconnect without answering, for example, by manually unplugging the network cable or automatically manipulating the transport layer.

As a result, the adversary is able to gather many triples, $\langle m, \mu, k_1 \rangle$, and mount the further guessing attacks off-line. The adversary is able to check *many* password guesses over δ while the server may notice *many* guessing attempts bounded by δ afterwards. So we call this simple attack the *many-to-many guessing attack*[4]. The window of vulnerability can be thought of as

$$O(T) = t_S + t_C + 2t_{CS} + \varepsilon + (\delta - 1)\epsilon$$

where 1) t_S is the time between receiving $\langle C, m \rangle$ and sending out $\langle \mu, k_1 \rangle$ in the server, 2) t_C is the time between receiving $\langle \mu, k_1 \rangle$ and sending out k_2 in the client, 3) t_{CS} is the time delay for exchanging messages over a communication channel, 4) ε is the (most influential) additional waiting time defined by the server considering t_C and t_{CS}, and 5) ϵ is the (most negligible) amount of time that defines the average time difference between one request and another subsequent one (so, $(\delta - 1)\varepsilon$ must be the time between the first notice of a failed attempt and the last one under δ). We address that this window of vulnerability is not negligible and is sufficient to allow the adversary to gather as many triples as she can fool the protocol out of being over δ and disclosing the password in the worst case.

3.2 Possible Prevention

We designed the ACCEPTABLE$(\cdot)$ function in Section 2.3 so as to return false if C (or a corresponding IP address) is being served already by another server instance upon receiving a new message $\langle C, m \rangle$. Note that the 'serving' corresponds to the authentication session only. This was actually contrived for resisting the many-to-many guessing attack. For the purpose, a small hash table may be maintained by the server to track the currently served or blocked clients. The blocking policy should be considered carefully but flexibly. This resolution method may reduce the window of vulnerability notably but still leaves an issue about DoS (Denial of Service). Note that there is a recent literature considering DoS attacks on password-based protocols [9]. Aside from the danger of DoS attacks, a race condition and some bottleneck to the hash table are now only concerns while they could be negligible by careful consideration. The possible prevention methods might be considered both in the design and implementation phases.

In order to examine the reality of our attack, we implement a prototype of TP-AMP and launch the many-to-many guessing attack on it. We implement both client and server using `CreateThread(·)` functions and `WinSock` in Pentium IV 1.8GHz, 512MB, MS-Windows platforms. MIRACL is utilized as a mathematical library. As for the ACCEPTABLE$(\cdot)$ function, we synchronized server threads with regard to serving a user and checking memory table. Simply a single run for our attack experiments takes 266 to 297 milliseconds. We then drive multi-threaded clients to starts many simultaneous authentication sessions. We summarize the experimental results as shown in Figure 3. Let $\delta = 5$ (times) and $\varepsilon = 3$ (seconds). In the experiments, Figure 3-(a) shows the result of M2M (many-to-many) attacks against a server without correct ACCEPTABLE$(\cdot)$ function. Until we increase the number of adversarial client threads to 100, we could observe all requests are stably served and the same number of

[4] This attack is negligible in the four-pass protocols since the server does not give sufficient information to the adversary forward and the client is usually not capable of listening to so many concurrent requests in the opposite case. Also, the best-known predecessors, EKE [5] and A-EKE [6], avoid this attack very *impressively* by not optimizing the protocol steps.

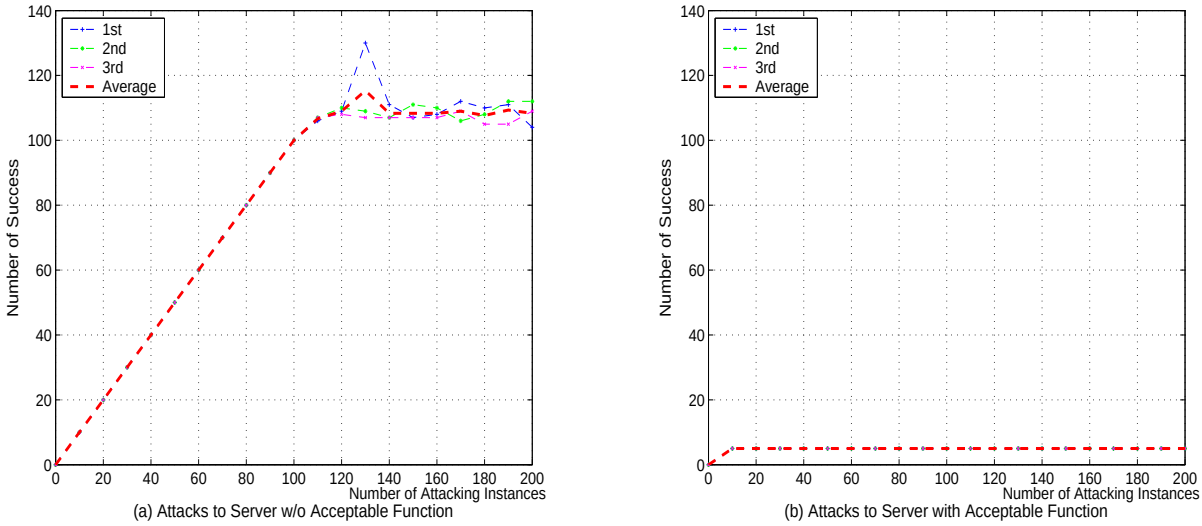


Fig. 3. M2M Attack Experiments

triples, $\langle m, \mu, k_1 \rangle$, are gathered. Up to 200 threads, we could observe that about 110 triples (130 at peak) are gathered by the adversary, on the average. Say, the M2M attack is very successful over the boundary of δ. When we increase the size of boundary δ for user convenience and consider a kind of delay including queueing and propagation delay, we could expect much better results. However, ACCEPTABLE$(\cdot)$ function could improve the security. Figure 3-(b) shows the result of M2M attacks against a server with correct ACCEPTABLE$(\cdot)$ function. We could observe that the adversary cannot gather more than 5 triples regardless of the number of simultaneous requests greater than δ. The remaining requests are exactly blocked by the server up to 200 threads. Figure 4 shows that the ACCEPTABLE$(\cdot)$ function does not decrease the performance of the protocol. Average service time per thread is much faster than the single run service time due to the optimization with regard to multi-threading. From our experiments, we could conclude that the M2M attack is realistic and the ACCEPTABLE$(\cdot)$ function is useful for preventing it.

4 Security and Efficiency Analysis

In this section, we discuss security and efficiency of TP-AMP.

4.1 Security of TP-AMP

For formal security, we adapt the improved models of [26] and [8]. Our refreshed security model is described in Appendix A. Readers are referred to it due to the page restriction of this paper. We prove that the TP-AMP protocol is secure, in the sense that an adversary attacking the system cannot determine session keys of fresh instances with greater advantage than that of an online dictionary attack, and cannot determine session keys of semi-fresh instances with greater advantage than that of an off-line dictionary attack. We define q_{se}, q_{ex}, q_{re}, q_{co} queries as those of type Send, Execute, Reveal, Corrupt, respectively, and q_{ro} queries to be made to the random oracles. Also we define some events related to the adversary making a password guess. Note that $[\alpha, \beta]$ means one of α and β is drawn. Also recall that the order

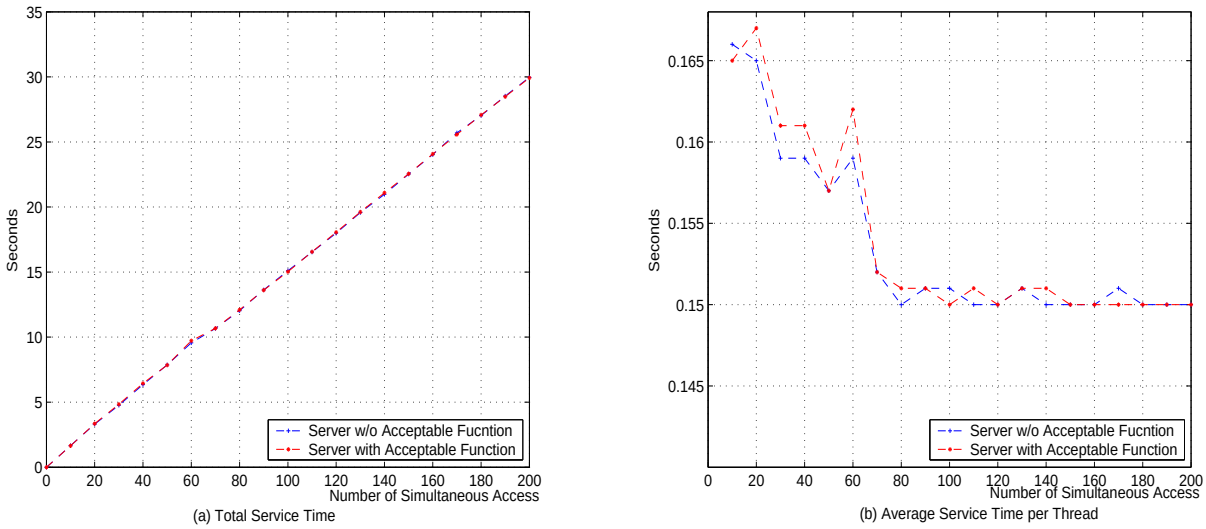


Fig. 4. Performance Experiments

of $\bar{\mathbb{G}}_q$ is $q-1$. Security arguments for the following theorems are described in Appendix B. Readers are referred to it.

Theorem 1. *Let $\mathcal{P}$ be the TP-AMP protocol with a password dictionary of size N. Fix an adversary $\mathcal{A}$ that runs in time t, and makes q_{se}, q_{ex}, q_{re}, q_{co} queries and q_{ro} queries. Then for $t' = O(t + (q_{ro} + q_{se} + q_{ex})t_{\mathsf{exp}})$ with t_{exp} denoting the computational time for exponentiation in $\mathbb{G}_q$:*

$$\mathsf{Adv}_{\mathcal{P}}^{\mathsf{ake-fs}}(\mathcal{A}) \leq \frac{q_{se}}{N} + O((q_{se}+q_{ex})q_{ro}\mathsf{Adv}_{\mathbb{G}_q}^{\mathsf{CDH}}(t', q_{ro})) + \frac{O((q_{se}+q_{ex})^2)}{q-1} + \frac{O(q_{se}+q_{ro}^2)}{2^{\kappa}}$$

and

$$\mathsf{Adv}_{\mathcal{P}}^{\mathsf{ake-fs.s}}(\mathcal{A}) = \frac{q_{ro}}{N} + \mathsf{Adv}_{\mathcal{P}}^{\mathsf{ake-fs}}(\mathcal{A}).$$

Proof sketch: Our proof will proceed by defining a sequence of games starting at the real game $\mathbf{G}_0$ and ending up at $\mathbf{G}_7$. In the beginning we simulate all protocol queries, and remove possible collisions and lucky events. We then reduce our protocol from solving CDH in a stringent way, for respective Execute and Send queries. So $\mathbf{G}_5$ models passive adversaries, while $\mathbf{G}_6$ does active adversaries. Finally, server compromise is manipulated in $\mathbf{G}_7$. □

Theorem 2. *Let $\mathcal{P}$ be the TP-AMP protocol with a password dictionary of size N. Fix an adversary $\mathcal{A}$ that runs in time t, and makes q_{se}, q_{ex}, q_{re}, q_{co} queries and q_{ro} queries. Then for $t' = O(t + (q_{ro} + q_{se} + q_{ex})t_{\mathsf{exp}})$ with t_{exp} denoting the computational time for exponentiation in $\mathbb{G}_q$:*

$$\mathsf{Adv}_{\mathcal{P}}^{\mathsf{ma}}(\mathcal{A}) \leq \frac{q_{se}}{N} + O((q_{se}+q_{ex})q_{ro}\mathsf{Adv}_{\mathbb{G}_q}^{\mathsf{CDH}}(t', q_{ro})) + \frac{O((q_{se}+q_{ex})^2)}{q-1} + \frac{O(q_{se}+q_{ro}^2)}{2^{\kappa}}$$

and

$$\mathsf{Adv}_{\mathcal{P}}^{\mathsf{c2s.s}}(\mathcal{A}) = \frac{q_{ro}}{N} + \mathsf{Adv}_{\mathcal{P}}^{\mathsf{ake-fs}}(\mathcal{A}).$$

We believe the given security argument in the random oracle model in Appendix B is sufficient to ensure that TP-AMP is a secure password authenticated key agreement protocol in the augmented model, though a full proof might be more intricate. A resistance to our real world attack can be observed by manipulating the $\mathsf{ACCEPTABLE}(\cdot)$ function. Since TP-AMP is simple in its structure, it might also be easy and obvious to examine its security heuristically but we do not manipulate any heuristic analysis in this paper.

4.2 Efficiency of TP-AMP

We may consider the number of expensive operations, for example, multiple precision multiplications (MPM) in $\mathbb{Z}_p^*$, in order to analyze the performance of TP-AMP. We approximate the number of multiplications on average, by assuming the use of a left-to-right binary exponentiation method or a simultaneous exponentiation method (denoted by sim) [27]. A slight computational difference between squaring and multiplication can be ignored for convenience.

Let $\kappa' = \kappa'' = \kappa''' = \kappa$ for secure prime p while $\kappa' = \kappa'' = \kappa''' = \ell - 2/3$ for safe prime p for simple analysis. Practically we can set $\kappa'' = 2\kappa$ with intentionally smaller exponents, for example, $x \stackrel{\mathrm{R}}{\leftarrow} \{0,1\}^{\kappa''}$ and not $\stackrel{\mathrm{R}}{\leftarrow} \mathbb{Z}_q^*$ for safe prime p though it is not general for security argument. For the client, γ may take $1.5(\ell - \kappa')$ while m may need $1.5\kappa'' + 1$ MPM. For the server, β may take $1.5(\kappa + \kappa'') + 2$ or $2\kappa'' + 1(\mathsf{sim})$ MPM. Finally the client may need $1.5\kappa'''$ for α. As a result, the client may need $1.5(\ell - \kappa' + \kappa'' + \kappa''') + 1$ while the server may require $1.5(\kappa + 2\kappa'') + 2$ MPM totally. Also if we assume an ideal cipher for γ as like EKE2 and AuthA [3, 4], the client may need $1.5(\kappa'' + \kappa''')$ MPM only. One can easily see that the TP-AMP protocol is efficient especially when we use a secure prime, p, since $\kappa' = \kappa'' = \kappa''' = \kappa$.

TP-AMP is comparable to the most closely related protocol PAK-Z (with an efficient instance Y using Schnorr's signature [30]) [25, 26] and AuthA [4, 8] in terms of efficiency. In general (with regard to computation and communications costs), TP-AMP is more efficient than those related schemes under the same assumption, for example, on a safe or secure prime with group size exponents, or an ideal cipher. However, when we use a safe prime with intentionally smaller (say, 160 bits) exponents, PAK-Y shows better in the client, while TP-AMP does still better in the server. As we mentioned already, TP-AMP can be instantiated on EKE2 [3, 4, 8] and provide efficiency on that framework. Thus, we would like to address that TP-AMP is a practical password authenticated key agreement protocol with sufficient security and generic features in the augmented model.

5 Conclusion

Though three-pass authenticated key agreement protocols may reduce one-round from four-pass protocols and are easier to apply provable security, we show that they are vulnerable to the novel many-to-many password guessing attacks if the protocol uses a password as a long-term secret. From the practical perspective we design and analyze a new three-pass protocol, TP-AMP, in the augmented model and show several interesting features. In the future study, we will conduct more intensive work on three-pass and four-pass protocols for password authenticated key agreement.

Acknowledgement

This work was supported in part by Korea Research Foundation Grant (KRF-2003-003-D00434). The author thanks anonymous referees for kind comments.

References

1. M. Bellare and P. Rogaway, "Entity authentication and key distribution," In *Crypto 1993*, LNCS 773, pp.232-249, 1993.
2. M. Bellare and P. Rogaway, "Provably secure session key distribution-the three party case," In *ACM Symposium on the Theory of Computing*, pp.232-249, 1993.
3. M. Bellare, D. Pointcheval and P. Rogaway, "Authenticated key exchange secure against dictionary attack," In *Eurocrypt 2000*, LNCS 1807, pp.139-155, 2000.
4. M. Bellare and P. Rogaway, "The AuthA protocol for password-based authenticated key exchange," Submission to the IEEE P1363.2 study group, available from `http://www.cs.ucdavis.edu/~rogaway/papers/autha.ps`
5. S. Bellovin and M. Merritt, "Encrypted key exchange : password-based protocols secure against dictionary attacks," In *IEEE Symposium on Research in Security and Privacy*, pp. 72-84, 1992.
6. S. Bellovin and M. Merritt, "Augmented encrypted key exchange: a password-based protocol secure against dictionary attacks and password-file compromise," In *ACM Conference on Computer and Communications Security*, pp. 244-250, 1993.
7. V. Boyko, P. MacKenzie and S. Patel, "Provably secure password authenticated key exchange using Diffie-Hellman," In *Eurocrypt 2000*, LNCS 1807, pp.156-171, 2000.
8. E. Bresson, O. Chevassut, and D. Pointcheval, "Security proofs for an efficient password-based key exchange," In *ACM Conference on Computer Communications Security*, 2003.
9. E. Bresson, O. Chevassut, and D. Pointcheval, "New security results on Encrypted Key Exchange," In *International Workshop on Theory and Practice in Public Key Cryptography*, LNCS 2947, pp. 145-158, 2004.
10. W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol.22, no.6, pp.644-654, November 1976.
11. W. Diffie, P. van Oorschot, and M. Wiener, "Authentication and authenticated key exchanges," Designs, Codes and Cryptography, 2, pp. 107-125, 1992.
12. O. Goldreich and Y. Lindell, "Session-Key Generation Using Human Passwords Only," In *Cypto 2001*, LNCS 2139, pp.408-432, 2001.
13. IEEE P1363.2, *Standard specifications for password-based public key cryptographic techniques*, available from `http://grouper.ieee.org/groups/1363/`, December 2002.
14. ISO/IEC WD 11770-4, *Information technology - Security techniques - Key management - Part 4: Mechanisms based on weak secrets*, ISO/IEC JTC 1/SC 27, November 2003.
15. Phoenix Technologies, Inc., "Research Papers on Strong Password Authentication," available from `http://www.integritysciences.com/links.html`, 2002.
16. D. Jablon, "Strong password-only authenticated key exchange," *ACM Computer Communications Review*, vol.26, no.5, pp.5-26, 1996.
17. D. Jablon, "Extended password key exchange protocols," In *WETICE Workshop on Enterprise Security*, pp.248-255, 1997.
18. J. Katz, R. Ostrovsky, and M. Yung, "Efficient Password-Authenticated Key Exchange Using Human-Memorable Passwords ," In *Eurocrypt 2001*, LNCS 2045, pp.475-494, 2001.
19. K. Kobara and H. Imai, "Pretty-simple password-authenticated key-exchange protocol proven to be secure in the standard model," IEICE Trans., E85-A(10), pp.2229-2237, 2002.
20. H. Krawczyk, "SIGMA: The 'SINGn-and-MAc' approach to authenticated Diffie-Hellman and its use in the IKE protocols," *Advances in Cryptology - CRYPTO 2003*, Lecture Notes in Computer Science, Vol. 2729, Springer-Verlag, pp. 400-425, 2003.
21. T. Kwon, "Authentication and key agreement via memorable password," In *ISOC Network and Distributed System Security Symposium*, February 2001.
22. T. Kwon, "Practical authenticated key agreement using passwords," Full version of this paper, available from `http://dasan.sejong.ac.kr/~tkwon/amp.html`.
23. C. Lim and P. Lee, "A key recovery attack on discrete log-based schemes using a prime order subgroup," In *CRYPTO 97*, pp.249-263, 1997.
24. M. Lomas, L. Gong, J. Saltzer, and R. Needham, "Reducing risks from poorly chosen keys," In *ACM Symposium on Operating System Principles*, pp.14-18, 1989.
25. P. MacKenzie, "More efficient password-authenticated key exchange," In *RSA Conference*, Cryptographers Track, LNCS 2020, pp.361-377, 2001.
26. P. MacKenzie, "The PAK suite: Protocols for Password-Authenticated Key Exchange," Submission to IEEE P1363.2, April 2002.

27. A. Menezes, P. van Oorschot and S. Vanstone, *Handbook of applied cryptography*, CRC Press,Inc., pp.517-518, 1997.
28. P. van Oorschot and M. Wiener, "On Diffie-Hellman key agreement with short exponents," In *Eurocrypt 96*, pp. 332-343, 1996.
29. R. Perlman and C. Kaufman, "PDM: A new strong password-based protocol," In *USENIX Security Symposium*, pp.313-321, 2001.
30. C. Schnorr, "Efficient identification and signatures for smart cards," In *Crypto 89*, pp.239-251, 1989.
31. M. Scott, *Personal communication*, July 2001.
32. T. Wu, "Secure remote password protocol," In *ISOC Network and Distributed System Security Symposium*, 1998.

A Security Model

For the formal handling of security, we recall the model of [3] that was designed for the problem of authenticated key agreement between two parties, a client and a server, sharing a weak secret. Also we consider the extended parts of [26] and [8] for the augmented model.

A.1 Participants and Keys.

Let *Clients* and *Servers* be the finite, disjoint, nonempty sets of principals which are modeled as probabilistic polynomial time algorithms with input/output tapes. A client $C \in Clients$ has a secret password π which is drawn randomly from small space Π of size N. A server $S \in Servers$ has a transformed-password τ_C which contains an entry per client, where $\tau_C \stackrel{\mathrm{T}}{\leftarrow} \pi$ for C. We call π and τ_C long-lived-weak keys (LL-keys). They are the same values in the balanced model, while not in the augmented model.

A.2 Execution of the Protocol.

A protocol, $\mathcal{P}$, is formally a probabilistic algorithm that determines how instances of the principals behave in response to inputs sent from their environment. The inputs may be given by an adversary, $\mathcal{A}$, that has complete control over the environment. Formally, the adversary is a probabilistic algorithm with a distinguished query tape. Queries written on this tape are answered by principals according to $\mathcal{P}$. An unlimited number of instances of principals are modeled for multiple execution of the protocol. Instance i of principal $U \in \{C, S\}$ is denoted by U^i and considered as an oracle. The adversary $\mathcal{A}$ is able to make various queries to any instance U^i, including random oracle queries:

- $\mathsf{Send}(U^i, M)$: This query models $\mathcal{A}$ sending message M to instance U^i. The instance computes what the protocol says to, and sends back the response to $\mathcal{A}$.
- $\mathsf{Execute}(C^i, S^j)$: This query models passive attacks, where $\mathcal{A}$ gets access to the execution of $\mathcal{P}$ between C^i and S^j by eavesdropping, so as to output the transcript of the execution.
- $\mathsf{Reveal}(U^i)$: This query models the compromise of the session key sk held by U^i.
- $\mathsf{Corrupt}(U)$: This query models the case that π or τ_C is disclosed, to deal with forward secrecy. This query may also be used to replace the value of τ_C used by server S.
- $\mathsf{Test}(U^i)$: This query models the semantic security of session key sk. It may be asked at any time during the execution of $\mathcal{P}$, but at most once. It is answered by flipping a coin b. If $b = 1$, then sk is returned. Otherwise, a random value is returned as the key.

A.3 Partnering and Pairing.

When a client or server instance accepts, it may hold a partner-id *pid*, session-id *sid*, and a session key *sk*. By the time both instances have terminated in a fixed number of flows, each instance should have accepted. Then instances C^i and S^j are said to be *partnered* if both accept, they hold (*pid*, *sid*, *sk*) and (*pid′*, *sid′*, *sk′*), respectively, with *pid*=*S*, *pid′*=*C*, *sid*=*sid′*, and *sk*=*sk′*, and no other instance accepts with session-id equal to *sid*. Also instances C^i and S^j are said to be *paired with* each other if there have been the correct Send queries before each termination regardless of the final acceptance. We define internal variables for handling this in the simulation.

A.4 Freshness.

Two notions of freshness are defined to ensure that a session key is not disclosed to the adversary and to incorporate a requirement for forward secrecy [3, 26].

- An instance U^i is nfs-fresh (fresh with no requirement for forward secrecy) unless either (1) a Reveal(U^i) query occurs, (2) a Reveal(U'^j) query occurs where U'^j is the partner of U^i, or (3) a Corrupt(U^*) query occurs where U^* denotes "somebody" [3].
- An instance U^i is fs-fresh (fresh with forward secrecy) unless either (1) a Reveal(U^i) query occurs, (2) a Reveal(U'^j) query occurs where U'^j is the partner of U^i, or (3) a Corrupt(U^*) query occurs before the Test query and a Send(U^i, M) query occurs for some string M.

A.5 Semi-freshness.

Two notions of freshness are modified as in [26], in order to consider the resistance to server compromise. The modification applies to server instances, even if some server has been compromised.

- An instance U^i is semi-nfs-fresh (semi-fresh with no requirement for forward secrecy) unless either (1) a Reveal(U^i) query occurs, (2) a Reveal(U'^j) query occurs where U'^j is the partner of U^i, (3) a Corrupt(C) query occurs, or (4) $U \in$ *Clients* and a Corrupt(S) query occurs.
- An instance U^i is semi-fs-fresh (fresh with forward secrecy) unless either (1) a Reveal(U^i) query occurs, (2) a Reveal(U'^j) query occurs where U'^j is the partner of U^i, (3) a Corrupt(C) query occurs before the Test query and a Send(U^i, M) query occurs for some string M, or (4) $U \in$ *Clients*, a Corrupt(S) query occurs before the Test query and a Send(U^i, M) query occurs for some string M.

A.6 Advantage of the Adversary.

We define the authenticated key exchange (ake) advantage of the adversary against protocol $\mathcal{P}$ as the probability that $\mathcal{A}$ correctly guesses the bit b selected in the Test query when the single Test query was made to some fresh and terminated instance U^i. Assuming $\mathcal{A}$ outputs b', the advantage is defined as follows:

$$\mathsf{Adv}^{\mathsf{ake}}_{\mathcal{P}}(\mathcal{A}) \stackrel{\text{def}}{=} 2\Pr[b = b'] - 1$$

The probability space is over all the random coins of the adversary and all the oracles. The protocol is said to be ake-*secure* if $\mathcal{A}$'s advantage is negligible in the security parameter.

As defined in [3, 26] we can distinguish $\mathsf{Adv}^{\mathsf{ake\text{-}fs}}_{\mathcal{P}}(\mathcal{A})$ and $\mathsf{Adv}^{\mathsf{ake\text{-}nfs}}_{\mathcal{P}}(\mathcal{A})$. Also the notions of authentication are defined for client-to-server authentication, server-to-client authentication, and mutual authentication. We define $\mathsf{Adv}^{\mathsf{c2s}}_{\mathcal{P}}(\mathcal{A})$ to be the probability that a server oracle terminates before any Corrupt query without having a partner oracle. We define $\mathsf{Adv}^{\mathsf{s2c}}_{\mathcal{P}}(\mathcal{A})$ to be the probability that a client oracle terminates before any Corrupt query without having a partner oracle. We define $\mathsf{Adv}^{\mathsf{ma}}_{\mathcal{P}}(\mathcal{A})$ to be the probability that some oracle terminates before any Corrupt query without having a partner oracle. For semi-freshness considering the resistance to server compromise, we can distinguish $\mathsf{Adv}^{\mathsf{ake\text{-}fs.s}}_{\mathcal{P}}(\mathcal{A})$ and $\mathsf{Adv}^{\mathsf{ake\text{-}nfs.s}}_{\mathcal{P}}(\mathcal{A})$. Similarly, we define $\mathsf{Adv}^{\mathsf{c2s.s}}_{\mathcal{P}}(\mathcal{A})$ to be the probability that a server oracle terminates before any Corrupt query to a client without having a partner oracle.

A.7 Computational Diffie-Hellman.

Let $\mathcal{A}$ be a probabilistic polynomial algorithm running in time t to output a list of group elements. We define

$$\mathsf{Adv}^{\mathsf{CDH}}_{\mathbb{G}_q}(\mathcal{A}) \stackrel{\text{def}}{=} \Pr[(x, y) \stackrel{\text{R}}{\leftarrow} \mathbb{Z}_q : \mathcal{A}(g^x, g^y) = g^{xy}].$$

Then we could have

$$\mathsf{Adv}^{\mathsf{CDH}}_{\mathbb{G}_q}(t, n) \stackrel{\text{def}}{=} \max_{\mathcal{A}}\{\mathsf{Adv}^{\mathsf{CDH}}_{\mathbb{G}_q}(\mathcal{A})\}$$

where the maximum is taken over all adversaries that run in time at most t and output a list containing at most n group elements. We often use $\mathsf{Succ}^{\mathsf{CDH}}_{\mathbb{G}_q}$ in the literature because CDH is a computational problem but we prefer $\mathsf{Adv}^{\mathsf{CDH}}_{\mathbb{G}_q}$ in order to distinguish it from the success event of the Test-query. The CDH assumption states that $\mathsf{Adv}^{\mathsf{CDH}}_{\mathbb{G}_q}(t, n)$ is negligible for t and n polynomial in κ.

B Security Proof

Theorem 1 *Let $\mathcal{P}$ be the TP-AMP protocol with a password dictionary of size N. Fix an adversary $\mathcal{A}$ that runs in time t, and makes q_{se}, q_{ex}, q_{re}, q_{co} queries and q_{ro} queries. Then for $t' = O(t + (q_{ro} + q_{se} + q_{ex})t_{\mathsf{exp}})$ with t_{exp} denoting the computational time for exponentiation in $\mathbb{G}_q$:*

$$\mathsf{Adv}^{\mathsf{ake-fs}}_{\mathcal{P}}(\mathcal{A}) \leq \frac{q_{se}}{N} + O((q_{se} + q_{ex})q_{ro}\mathsf{Adv}^{\mathsf{CDH}}_{\mathbb{G}_q}(t', q_{ro})) + \frac{O((q_{se} + q_{ex})^2)}{q - 1} + \frac{O(q_{se} + q_{ro}^2)}{2^{\kappa}}$$

and

$$\mathsf{Adv}^{\mathsf{ake-fs.s}}_{\mathcal{P}}(\mathcal{A}) = \frac{q_{ro}}{N} + \mathsf{Adv}^{\mathsf{ake-fs}}_{\mathcal{P}}(\mathcal{A}).$$

Proof: Our proof will proceed by defining a sequence of games starting at the real game $\mathbf{G}_0$ and ending up at $\mathbf{G}_7$. We define the success event in any game $\mathbf{G}_n$:

- Succ_n: This event occurs if $b = b'$, where b is the bit involved in the Test-query, and b' is the output of the adversary.

Then we proceed with a sequence of games. The first game $\mathbf{G}_0$ is equivalent to the real protocol $\mathcal{P}$. In the ending games such as $\mathbf{G}_5$, $\mathbf{G}_6$, and $\mathbf{G}_7$, $\mathcal{A}$ will be reduced to a straightforward and negligible attacker.

***Game* $\mathbf{G}_0$:** This is the real protocol in the random oracle model. The oracles including five random oracles (H_0, h_1, h_2, h_3, and h_4) and all instances (C^i and S^j) are thus available to the adversary. By definition,

$$\mathsf{Adv}^{\mathsf{ake-fs}}_{\mathsf{tp}}(\mathcal{A}) = 2\Pr[\mathsf{Succ}_0] - 1.$$

Then we furthermore assume that we choose a random bit b' if the game aborts or stops with no answer b' from the adversary, or the adversary has not finished the game with more than q_{se} queries or lasts for more than time t where q_{se} and t are predetermined upper-bounds.

***Game* $\mathbf{G}_1$:** This is the game of minute simulation for the adversary, so that we simulate all the oracles for each query in Figure 5. To make our simulation sound, we keep three lists of transcripts: $\mathcal{L}_h$ for random oracle queries to $h_i()$ for $i \in \{1, 2, 3, 4\}$ and H_0, $\mathcal{L}_{\mathcal{A}}$ for the random oracle queries directly asked by the adversary, and $\mathcal{L}_{\mathcal{P}}$ for the exchanged protocol messages. Internal variables and states are postulated to be written on the corresponding tapes. We assume that $h_1(\cdot)$ and $H_0(\cdot)$ are queried with $\langle C, \pi\rangle$ at most once in the whole games. We also postulate that the adversary queries $h_i(\cdot)$ without loss of generality for $i \in \{2, 3, 4\}$. The internal variables, $\mathrm{Open}_{U^{[i,j]}}$, $\mathrm{Close}_{U^{[i,j]}}$ and $\mathrm{Accept}_{U^{[i,j]}}$, are all set $false$ initially. Another internal variable $\mathrm{Acceptable}_{S^j}$ is set $true$ initially. If Open_{C^i} and Open_{S^j} are all true and there is a $\mathsf{Send}(C^i, \langle \mu, k_1\rangle)$ query, then C^i is paired with S^j and thus we have $\langle\langle C, m\rangle, \langle \mu, k_1\rangle, *\rangle \in \mathcal{L}_{\mathcal{P}}$. Similarly, if Open_{C^i} and Open_{S^j} are all true and there is a $\mathsf{Send}(S^j, k_2)$ query only for the case that a $\mathsf{Send}(C^i, \langle \mu, k_1\rangle)$ query was asked, then S^j is paired with C^i and thus we have $\langle\langle C, m\rangle, \langle \mu, k_1\rangle, k_2\rangle \in \mathcal{L}_{\mathcal{P}}$. $\mathrm{Close}_{U^{[i,j]}}$ and $\mathrm{Accept}_{U^{[i,j]}}$ can be used to allege partnering.

From this minute simulation, we can easily see that the game is perfectly indistinguishable from the real attack game in the random oracle model. Thus we have:

$$\Pr[\mathsf{Succ}_1] = \Pr[\mathsf{Succ}_0].$$

***Game* $\mathbf{G}_2$:** In this game, we avoid collisions amongst an m or μ value of the current honest choice and those of the previous execution, and amongst the $h_i(\cdot)$ queries asked by the adversary for $i \in \{2, 3, 4\}$, with probability bounded by the birthday paradox. We can postulate random oracle queries $H_0(C, \pi)$ and $h_1(C, \pi)$ are respectively free from mutual collisions in the information theoretic sense. Let Coll_χ be the event that an χ value generated by a Send or $\mathsf{Execute}$ query is equal to a value of the previous execution's corresponding query, or an input to the previous execution's succeeding Send query. We play the game in a way to abort if the event Coll_m or Coll_μ occurs. A new list $\mathcal{L}_{Coll}$ keeps track of m and μ by saving $\langle [C^i, S^j], [in, out], [m, \mu]\rangle$ where $[\chi, \psi]$ means one of χ and ψ is drawn. The random outputs or inputs must be checked with $\mathcal{L}_{Coll}$ for a collision in each simulation of send queries. So, this game may abort with probability bounded by $\frac{(q_{se}+q_{ex})^2}{q-1}$. Then we modify the simulation of the random oracle queries so that we abort the game if $h_i(\cdot)$ is directly asked by the adversary and $\langle i, *, r\rangle \in \mathcal{L}_{\mathcal{A}}$ for a random answer $r \in \{0, 1\}^\kappa$ and $i \in \{2, 3, 4\}$. This may make the game abort with probability bounded by $\frac{q_{ro}^2}{2^{\kappa+1}}$.

The two games $\mathbf{G}_2$ and $\mathbf{G}_1$ are perfectly indistinguishable unless the unlikely event Coll_m or Coll_μ occurs, or a collision is found by the adversary in $h_i(\cdot)$ for $i \in \{2, 3, 4\}$. So we have:

$$|\Pr[\mathsf{Succ}_2] - \Pr[\mathsf{Succ}_1]| \leq \frac{O((q_{se} + q_{ex})^2)}{q - 1} + \frac{O(q_{ro}^2)}{2^{\kappa+1}}.$$

***Game* $\mathbf{G}_3$:** In this game, we abort the protocol runs if the adversary has been lucky in guessing the values k_1 and k_2 without asking the corresponding random oracle queries. We achieve this aim by adding more steps to processing the Send queries in the simulation.

In processing the $\mathsf{Send}(C^i, \langle \mu, k_1 \rangle)$ query, the client should check if $\langle 2, C, S, m, \mu, \alpha, \gamma', k_1 \rangle \in \mathcal{L}_{\mathcal{A}}$ or $\langle \langle C, m \rangle, \langle \mu, k_1 \rangle, * \rangle \in \mathcal{L}_{\mathcal{P}}$ just before setting Accept_{C^i} true (that is, right after checking that $k_1 = k_1'$). If both tests fail, the client should terminate without accepting. In processing the $\mathsf{Send}(S^j, k_2)$ query, the server should check if $\langle 3, C, S, m, \mu, \beta, \gamma', k_2 \rangle \in \mathcal{L}_{\mathcal{A}}$ or $\langle \langle C, m \rangle, \langle \mu, k_1 \rangle, k_2 \rangle \in \mathcal{L}_{\mathcal{P}}$ just before setting Accept_{S^j} true (that is, right after checking that $k_2 = k_2'$). If both tests fail, the server should terminate without accepting. This modification ensures that k_1 and k_2 will come, in valid form, from either the simulator that traces pairing or an adversary that has asked correct random oracle queries in all accepted cases.

The two games $\mathbf{G}_3$ and $\mathbf{G}_2$ are perfectly indistinguishable unless the client rejects the valid k_1 or the server rejects the valid k_2. The rejection may happen only if the adversary has correctly guessed the values k_1 and k_2 without asking the corresponding random oracle queries. So we have:

$$|\Pr[\mathsf{Succ}_3] - \Pr[\mathsf{Succ}_2]| \leq \frac{O(q_{se})}{2^{\kappa}}.$$

***Game* $\mathbf{G}_4$:** In this game, we abort the protocol runs if the adversary has been lucky in guessing the values γ' (or γ) and u (or u^{-1}) without asking the corresponding random oracle queries. We achieve this aim by modifying the random oracle queries to h_i for $i \in \{2, 3, 4\}$.

Provided that the $h_i(C, S, m, \mu, \alpha, \gamma')$ query was asked by the adversary and $\langle 0, C, \pi, \gamma \rangle \in \mathcal{L}_h$ for $\gamma = (\gamma')^{-1}$, it must be checked whether $\langle 0, C, \pi, \gamma \rangle \in \mathcal{L}_{\mathcal{A}}$ and $\langle 1, C, \pi, u \rangle \in \mathcal{L}_{\mathcal{A}}$ just before returning r for $i \in \{2, 3, 4\}$. If the latter test fails, the game must be aborted. Say, the simulator is able to determine if the adversary had made a correct guess for u or not by observing $\mathcal{L}_{\mathcal{A}}$. This modification ensures that the adversary should have asked correct random oracle queries to H_0 and h_1 for asking correct queries to h_i for $i \in \{2, 3, 4\}$.

The two games $\mathbf{G}_4$ and $\mathbf{G}_3$ are perfectly indistinguishable unless the protocol aborts in the random oracle queries. The abortion may happen only if the adversary has correctly guessed the values γ and u without asking the corresponding random oracle queries. So we have:

$$|\Pr[\mathsf{Succ}_4] - \Pr[\mathsf{Succ}_3]| \leq \frac{O(q_{ro})}{2^{\kappa}}.$$

***Game* $\mathbf{G}_5$:** In this game, we attempt to solve CDH if the adversary has been lucky in guessing the password before a $\mathsf{Corrupt}$ query and asks $\mathsf{Execute}(C^i, S^j)$ and random oracle queries. We achieve this aim by further modifying the random oracle queries to h_i for $i \in \{2, 3, 4\}$ and using the reduction from CDH. In other words, given a random Diffie-Hellman instance $\langle X, Y \rangle$ such that $X \leftarrow g^x$ and $Y \leftarrow g^y$, we construct an algorithm Ψ that attempts to solve CDH (i.e., find Z such that $Z \leftarrow g^{xy}$) by running $\mathcal{A}$ on the simulation changed in the following way.

In answering the $\mathsf{Execute}(C^i, S^j)$ query, the respective values are set as

$$m \leftarrow Xg^{\theta}, \mu \leftarrow Y^{\vartheta}, k_1' \leftarrow k_1 \xleftarrow{\mathrm{R}} \{0,1\}^{\kappa}, k_2' \leftarrow k_2 \xleftarrow{\mathrm{R}} \{0,1\}^{\kappa} \text{ and } sk_C \leftarrow sk_S \xleftarrow{\mathrm{R}} \{0,1\}^{\kappa},$$

where $\theta, \vartheta \xleftarrow{\mathrm{R}} Z_q^*$ and $\textsc{acceptable}(C, m)$ is evaluated, rather than asking the Send queries internally. So the values $\mathrm{Open}_{U^{[i,j]}}$ and $\mathrm{Close}_{U^{[i,j]}}$ must be false and the simulator does not ask random oracle queries in this case. Subsequent random oracle queries by the adversary are backpatched for consistency.

In answering the random oracle queries $h_i(C, S, m, \mu, [\alpha, \beta], \gamma')$ for $i \in \{2, 3, 4\}$, provided that $\langle 0, C, \pi, \gamma\rangle \in \mathcal{L}_{\mathcal{A}}$ and $\langle 1, C, \pi, u\rangle \in \mathcal{L}_{\mathcal{A}}$ for a correct password π and γ such that $\gamma = (\gamma')^{-1}$, it must be checked whether at least one of $\mathrm{Open}_{U^{[i,j]}}$ and $\mathrm{Close}_{U^{[i,j]}}$ is true. If the latter test fails, we stop the game and can say the adversary succeeds.

Then $\mathcal{A}$ may finish with asking the $\mathsf{Execute}(C^i, S^j)$ and random oracle queries such as $H_0(C, \pi)$, $h_1(C, \pi)$, and $h_i(C, S, m, \mu, [\alpha, \beta], \gamma')$ for either one of $i \in \{2, 3, 4\}$. As a result, the game stops and $\mathcal{A}$ succeeds. Recall that $g^{\rho} = H_0(C, \pi)$ and $u = h_1(C, \pi)$. In Ψ, the list $\mathcal{L}_{\mathcal{A}}$ and the oracles' internal tapes are parsed, and the value

$$\sigma^{\vartheta^{-1}u} Y^{\rho-\theta-m}$$

is added to the list of values Z, where $\sigma = [\alpha, \beta]$. Note that σ can be picked up with probability $O(\frac{1}{q_{ro}})$.

The two games $\mathbf{G}_5$ and $\mathbf{G}_4$ are perfectly indistinguishable unless the adversary has correctly guessed the password and asked the $\mathsf{Execute}(C^i, S^j)$ and corresponding random oracle queries. The event happens with probability bounded by CDH for collecting the list elements. Let t' be the running time of Ψ and note that $t' = O(t + (q_{ro} + q_{se} + q_{ex})t_{\mathsf{exp}})$. So we have:

$$|\Pr[\mathsf{Succ}_5] - \Pr[\mathsf{Succ}_4]| \leq O(q_{ro}\mathsf{Adv}^{\mathsf{CDH}}_{\mathbb{G}_q}(t', q_{ro})).$$

Game $\mathbf{G}_6$**:** In this game, we attempt to solve CDH (though it can fail in some case) if the adversary has been lucky in guessing the password before a $\mathsf{Corrupt}$ query and asks Send and random oracle queries. When solving CDH fails, we derive the probability boundary directly. We again use the reduction from CDH and modify the simulation of the corresponding queries, so that the algorithm Φ attempting to solve CDH may run $\mathcal{A}$ on the simulation.

The modification is simple; given a random Diffie-Hellman instance $\langle X, Y\rangle$, in each part of processing Send queries in the simulation, the following derivations are substituted for the original ones and the corresponding random oracle queries are all removed.

$$m \leftarrow Xg^{\theta}, \mu \leftarrow Y^{\vartheta}, k_1' \leftarrow k_1 \xleftarrow{\mathrm{R}} \{0,1\}^{\kappa}, k_2' \leftarrow k_2 \xleftarrow{\mathrm{R}} \{0,1\}^{\kappa} \text{ and } sk_C \leftarrow sk_S \xleftarrow{\mathrm{R}} \{0,1\}^{\kappa},$$

where $\theta, \vartheta \xleftarrow{\mathrm{R}} Z_q^*$. So the simulator will never ask random oracle queries in this game.

In answering the random oracle queries $h_i(C, S, m, \mu, \sigma, \gamma')$ for $i \in \{2, 3, 4\}$, the internal variables must be checked in the following ways.

(1) Provided that only one of Open_{C^i} and Open_{S^j} is true, Close_{C^i} is true when Open_{C^i} is true, and $\langle 0, C, \pi, \gamma\rangle \notin \mathcal{L}_{\mathcal{A}}$ and $\langle 1, C, \pi, u\rangle \notin \mathcal{L}_{\mathcal{A}}$ for a correct password π (say, $\gamma \neq (\gamma')^{-1}$ and $\sigma \neq \mathsf{CDH}(m\gamma', \mu^{u^{-1}}) \cdot \mu^{u^{-1}m} \bmod p$ due to game $\mathbf{G}_4$), we define this event as $\mathsf{NullOpen}$. We furthermore record $\#\mathsf{NullOpen}$ (which was initially set zero) by increasing one so that we can track the number of failed guesses without detection, if the same incorrect password was not claimed previously in $\mathcal{L}_{\mathcal{A}}$. Thus we can easily see that the probability for the next $\mathsf{NullOpen}$ event is bounded by $1 - \frac{1}{N - \#\mathsf{NullOpen}}$ where $\#\mathsf{NullOpen} \leq q_{se}$. We terminate the protocol runs without accepting in this case.

(2) If $\langle 0, C, \pi, \gamma\rangle \in \mathcal{L}_{\mathcal{A}}$ and $\langle 1, C, \pi, u\rangle \in \mathcal{L}_{\mathcal{A}}$ for a correct password π and γ such that $\gamma = (\gamma')^{-1}$, we abide by the following rules for each case attempting to solve CDH.

- If both Open_{C^i} and Open_{S^j} are true, the case is very similar to that of game $\mathbf{G}_5$. So it is obvious that the probability is bounded by $O(q_{se}q_{ro}\mathsf{Adv}^{\mathsf{CDH}}_{\mathbb{G}_q}(t', q_{ro}))$, since Φ may collect the list of Z using the $\iota-$th query in a random index $\iota \in \{1, 2, \cdots, q_{se}\}$.

- If only one of Open_{C^i} and Open_{S^j} is true and Close_{C^i} is true when Open_{C^i} is true, the adversary succeeds perfectly but CDH is only solvable for $\langle X, \mu^{u^{-1}} \rangle$ or $\langle m\gamma', Y \rangle$, say not for $\langle X, Y \rangle$ in this case. This must be the mentioned failed case for solving CDH under the adversary's correct password guess. Hence, we do not consider CDH in this case, but rather we stop the game by defining the event, OnLine. When we consider #NullOpen and previous NullOpen events, the OnLine may happen with probability approximately bounded by $\frac{\#\mathsf{NullOpen}}{N-\#\mathsf{NullOpen}} \prod_{i=0}^{\#\mathsf{NullOpen}-1}(1-\frac{1}{N-i}) = \frac{\#\mathsf{NullOpen}}{N}$ since we have considered and avoided all collisions in the previous games. Due to the definition of fresh and semi-fresh oracles, there could be no reveal query. Since $sk_C \leftarrow sk_S \xleftarrow{\text{R}} \{0,1\}^\kappa$, the success probability of the adversary is $\frac{1}{2}$ unless OnLine occurs in this case. Thus, we have $\Pr[\mathsf{OnLine}] + \frac{1}{2}\Pr[\neg\mathsf{OnLine}]$ as the adversary's success probability in this case. One can easily derive the direct probability boundary such as $\frac{1}{2} + \frac{q_{se}}{2N}$ since $\#\mathsf{NullOpen} \leq q_{se}$.
- Otherwise and if Acceptable_{S^j} is true, the game is exactly the same as the previous game, $\mathbf{G}_5$. So it is obvious that the probability is bounded by $O(q_{ro}\mathsf{Adv}^{\mathsf{CDH}}_{\mathbb{G}_q}(t', q_{ro}))$. If Acceptable_{S^j} is false, the adversary fails without any advantage.

The two games $\mathbf{G}_6$ and $\mathbf{G}_5$ are indistinguishable unless the adversary has correctly guessed the password and asked at least one of $\mathsf{Send}(C^i, \mathsf{Start})$ and $\mathsf{Send}(S^i, \langle C, m \rangle)$ along with corresponding random oracle queries. So we have:

$$|\Pr[\mathsf{Succ}_6] - \Pr[\mathsf{Succ}_5]| \leq \frac{q_{se}}{2N} + O(q_{se}q_{ro}\mathsf{Adv}^{\mathsf{CDH}}_{\mathbb{G}_q}(t', q_{ro})).$$

***Game* $\mathbf{G}_7$:** In this game, we consider Corrupt queries. Thus, fresh and semi-fresh oracles will only imply fs-fresh and semi-fs-fresh oracles, respectively, in this game. When a Corrupt query is made to a fresh or semi-fresh client, π will be answered, while to a fresh or semi-fresh server, γ' and ν will be answered. In each case, we will modify the games to the corresponding "corrupted" games. We define a *corrupted game* as a game where Execute queries and corresponding random oracle queries are only allowed to the adversary for our measurement. This game could model the notion of forward secrecy. We also define a *semi-corrupted game* as a game where $\mathsf{Send}(S^j, \langle C, m \rangle)$ queries and corresponding random oracle queries are only allowed to the adversary for our measurement. This game could model the resistance against server compromise. All disallowed or unexpected queries are answered by following the rules defined in the previous games as much as possible, while the modified game can be considered as a kind of mini game without those queries. These will make our modified game indistinguishable from the previous games unless specific events occur in the corrupted games.

(1) By the definition of fresh and semi-fresh oracles, a Corrupt query must be made by an adversary to the client oracle definitely after the Test query. Thus, we assume such a $\mathsf{Corrupt}(C)$ query was made by an adversary after the Test query but in the old games. Then, we could modify the current game to another corrupted game in a way to reply with old transcripts having the same $\text{Open}_{U^{[i,j]}}$ values (say, true or false) through the random index $\iota \in \{1, 2, \cdots, q_{se} + q_{ex}\}$ when an Execute query is asked. Due to game $\mathbf{G}_5$, $\mathcal{A}$ may finish with asking the $\mathsf{Execute}(C^i, S^j)$ and random oracle queries such as $H_0(C, \pi)$, $h_1(C, \pi)$, and $h_i(C, S, m, \mu, [\alpha, \beta], \gamma')$ for either one of $i \in \{2, 3, 4\}$. As a result, the game stops and $\mathcal{A}$ succeeds. One can easily see that the success probability is bounded by $O((q_{se} + q_{ex})q_{ro}\mathsf{Adv}^{\mathsf{CDH}}_{\mathbb{G}_q}(t', q_{ro}))$, since Ψ as a result may collect the list of Z using the random index $\iota \in \{1, 2, \cdots, q_{se} + q_{ex}\}$.

(2) If a Corrupt query is made by the adversary before the Test query in this game, it must be the Corrupt query with regard to semi-fresh server. Say, it must be the Corrupt(S) query regardless of timeliness to the Test query. In this case, we modify this game to the semi-corrupted game in the following ways.
- In answering the random oracle queries $h_i(C, S, m, \mu, [\alpha, \beta], \gamma')$ for $i \in \{2, 3, 4\}$, provided that a Send$(S^j, \langle C, m\rangle)$ query is made by the adversary, $\langle 1, C, \pi, u\rangle \notin \mathcal{L}_{\mathcal{A}}$ for a correct password π, and $\langle i, C, S, m, \mu, \varrho, \gamma'\rangle \in \mathcal{L}_{\mathcal{A}}$ for $\varrho = (m\gamma' g^m)^y$, we stop the game and the adversary automatically succeeds. From the information theoretic perspective, this is a lucky event bounded by $\frac{O(q_{se})}{q-1}$. This is because the discrete logarithm of γ is eventually necessary for choosing m in a pre-defined subgroup. If we set $\beta = (m\gamma' g^{h(m,\mu)})^y$ and $w = u^{-1}(x + h(m, \mu)) \bmod q$ as we discussed in Section 2.3, we could observe the boundary is modified to $\frac{O(q_{ro})}{2^\kappa}$.
- If random oracle queries to H_0 and h_1 are made with the correct password, the game stops and the adversary automatically succeeds. Since γ' and ν are already given to the adversary, (s)he may ask corresponding random oracle queries as much as (s)he can. Due to game $\mathbf{G}_6$, this event may happen with probability bounded by $\frac{q_{ro}}{2N}$.

The two games $\mathbf{G}_7$ and $\mathbf{G}_6$ are indistinguishable unless the adversary has asked a Corrupt query and Execute or Send queries along with corresponding random oracle queries. So we have for fresh oracles:

$$|\Pr[\mathsf{Succ}_7] - \Pr[\mathsf{Succ}_6]| \leq O((q_{se} + q_{ex})q_{ro}\mathsf{Adv}^{\mathsf{CDH}}_{\mathbb{G}_q}(t', q_{ro})).$$

Also we have for semi-fresh oracles:

$$|\Pr[\mathsf{Succ}_7] - \Pr[\mathsf{Succ}_6]| \leq \frac{q_{ro}}{2N} + \frac{O(q_{se})}{q-1} + \frac{O(q_{ro})}{2^\kappa} + O((q_{se} + q_{ex})q_{ro}\mathsf{Adv}^{\mathsf{CDH}}_{\mathbb{G}_q}(t', q_{ro})).$$

Summary: From the sequence of games above, we could aggregate the probability gaps among the games. By definition we could have $\Pr[\mathsf{Succ}_n] = \frac{1}{2} + \frac{\mathsf{Adv}^{\mathsf{ake}}_{\mathcal{P}}(\mathcal{A};n)}{2}$. Thus we have $\mathsf{Adv}^{\mathsf{ake}}_{\mathcal{P}}(\mathcal{A}) \leq 2\sum_{i=0}^{6} |\Pr[\mathsf{Succ}_{i+1}] - \Pr[\mathsf{Succ}_i]|$. As a result, we prove

$$\mathsf{Adv}^{\mathsf{ake-fs}}_{\mathcal{P}}(\mathcal{A}) \leq \frac{q_{se}}{N} + O((q_{se} + q_{ex})q_{ro}\mathsf{Adv}^{\mathsf{CDH}}_{\mathbb{G}_q}(t', q_{ro})) + \frac{O((q_{se} + q_{ex})^2)}{q-1} + \frac{O(q_{se} + q_{ro}^2)}{2^\kappa}$$

and

$$\mathsf{Adv}^{\mathsf{ake-fs.s}}_{\mathcal{P}}(\mathcal{A}) = \frac{q_{ro}}{N} + \mathsf{Adv}^{\mathsf{ake-fs}}_{\mathcal{P}}(\mathcal{A}). \qquad \square$$

Theorem 2 *Let $\mathcal{P}$ be the TP-AMP protocol with a password dictionary of size N. Fix an adversary $\mathcal{A}$ that runs in time t, and makes q_{se}, q_{ex}, q_{re}, q_{co} queries and q_{ro} queries. Then for $t' = O(t + (q_{ro} + q_{se} + q_{ex})t_{\mathsf{exp}})$ with t_{exp} denoting the computational time for exponentiation in $\mathbb{G}_q$:*

$$\mathsf{Adv}^{\mathsf{ma}}_{\mathcal{P}}(\mathcal{A}) \leq \frac{q_{se}}{N} + O((q_{se} + q_{ex})q_{ro}\mathsf{Adv}^{\mathsf{CDH}}_{\mathbb{G}_q}(t', q_{ro})) + \frac{O((q_{se} + q_{ex})^2)}{q-1} + \frac{O(q_{se} + q_{ro}^2)}{2^\kappa}$$

and

$$\mathsf{Adv}^{\mathsf{c2s.s}}_{\mathcal{P}}(\mathcal{A}) = \frac{q_{ro}}{N} + \mathsf{Adv}^{\mathsf{ake-fs}}_{\mathcal{P}}(\mathcal{A}).$$

Proof: The same argument can be achieved as Theorem 1. □

Queries	$h_i(q)$ for $i \in \{1,2,3,4\}$	$H_0(q)$
Simulation	If $\langle i,q,r\rangle \notin \mathcal{L}_h$: $r \xleftarrow{R} \{0,1\}^\kappa$ Add $\langle i,q,r\rangle$ to $\mathcal{L}_h$ If the query was asked by $\mathcal{A}$: Add $\langle i,q,r\rangle$ to $\mathcal{L}_\mathcal{A}$ Return r	If $\langle 0,q,R\rangle \notin \mathcal{L}_h$: $\rho \xleftarrow{R} \mathbb{Z}_q^*$ $R \leftarrow g^\rho \bmod p$ Add $\langle 0,q,R\rangle$ to $\mathcal{L}_h$ If the query was asked by $\mathcal{A}$: Add $\langle 0,q,R\rangle$ to $\mathcal{L}_\mathcal{A}$ Return R

$\mathsf{Send}(C^i, \mathsf{Start})$	$\mathsf{Send}(C^i, \langle \mu, k_1\rangle)$	$\mathsf{Send}(S^j, \langle C, m\rangle)$
$x \xleftarrow{R} \mathbb{Z}_q^*$ $\gamma \leftarrow H_0(C,\pi)$ $m \leftarrow g^x\gamma \bmod p$ Return $\langle C, m\rangle$ Update $\mathcal{L}_\mathcal{P}$ $\mathrm{Open}_{C^i} \leftarrow True$	$\gamma \leftarrow H_0(C,\pi)$ $\gamma' \leftarrow \gamma^{-1} \bmod p$ $u \leftarrow h_1(C,\pi)$ $\alpha \leftarrow \mu^{u^{-1}(x+m)} \bmod p$ $k_1' \leftarrow h_2(C,S,m,\mu,\alpha,\gamma')$ If $k_1 = k_1'$: $\mathrm{Accept}_{C^i} \leftarrow True$ $k_2 \leftarrow h_3(C,S,m,\mu,\alpha,\gamma')$ $sk_C \leftarrow h_4(C,S,m,\mu,\alpha,\gamma')$ Return k_2 Update $\mathcal{L}_\mathcal{P}$ Terminate ($\mathrm{Close}_{C^i} \leftarrow True$)	If $\neg\mathsf{ACCEPTABLE}(C,m)$: Abort ($\mathrm{Acceptable}_{S^j} \leftarrow False$) Else: $y \xleftarrow{R} \mathbb{Z}_q^*$ $u \leftarrow h_1(C,\pi)$ $\mu \leftarrow (g^u)^y \bmod p$ $\gamma \leftarrow H_0(C,\pi)$ $\gamma' \leftarrow \gamma^{-1} \bmod p$ $\beta \leftarrow (m\gamma^{-1}g^m)^y \bmod p$ $k_1 \leftarrow h_2(C,S,m,\mu,\beta,\gamma')$ Return $\langle \mu, k_1\rangle$ Update $\mathcal{L}_\mathcal{P}$ $\mathrm{Open}_{S^j} \leftarrow True$

$\mathsf{Send}(S^j, k_2)$	$\mathsf{Execute}(C^i, S^j)$
$\gamma \leftarrow H_0(C,\pi)$ $\gamma' \leftarrow \gamma^{-1} \bmod p$ $k_2' \leftarrow h_3(C,S,m,\mu,\beta,\gamma')$ If $k_2 = k_2'$: $\mathrm{Accept}_{S^j} \leftarrow True$ $sk_S \leftarrow h_4(C,S,m,\mu,\beta,\gamma')$ Terminate ($\mathrm{Close}_{S^j} \leftarrow True$)	$\langle C, m\rangle \leftarrow \mathsf{Send}(C^i, \mathsf{Start})$ $\langle \mu, k_1\rangle \leftarrow \mathsf{Send}(S^j, \langle C, m\rangle)$ $k_2 \leftarrow \mathsf{Send}(C^i, \langle \mu, k_1\rangle)$ $\mathsf{Send}(S^j, k_2)$ Return $\langle\langle C, m\rangle, \langle \mu, k_1\rangle, k_2\rangle$

$\mathsf{Reveal}(U^i)$	$\mathsf{Test}(U^i)$	$\mathsf{Corrupt}(U)$
If Accept_{U^i}: Return sk_U	$sk \leftarrow \mathsf{Reveal}(U^i)$ $b \xleftarrow{R} \{0,1\}$ If $b = 0$: $sk \xleftarrow{R} \{0,1\}^\kappa$ Return sk	If $U = C$: Return π Else if $U = S$: $\gamma \leftarrow H_0(C,\pi)$ $u \leftarrow h_1(C,\pi)$ $\pi_C \leftarrow \langle \gamma^{-1} \bmod p, g^u \bmod p\rangle$ Return π_C

Fig. 5. Simulation of Oracles