Pretty-Simple Password-Authenticated Key-Exchange Under Standard Assumptions

Kazukuni Kobara and Hideki Imai

Institute of Industrial Science, The University of Tokyo
4-6-1, Komaba, Meguro-ku, Tokyo, 153-8505 Japan
Tel: +81-3-5452-6232
FAX: +81-3-5452-6631
E-mail: {kobara,imai}@iis.u-tokyo.ac.jp

Abstract. In this paper, we propose a pretty-simple password-authenticated key-exchange protocol, which is proven to be secure in the standard model under the following three assumptions. (1) DDH (Decision Diffie-Hellman) problem is hard. (2) The entropy of the password is large enough to avoid on-line exhaustive search (but not necessarily off-line exhaustive search). (3) MAC is selectively unforgeable against partially chosen message attacks, (which is weaker than being existentially unforgeable against chosen message attacks).

1 Introduction

We consider the following password-authenticated key-exchange protocol, by which two entities can share a fresh authenticated session-key (being secure against off-line attacks) by using a pre-shared human-memorable password (or pass phrases), which may be insecure against off-line attacks but secure against on-line attacks.

The on-line attack is a serial exhaustive search for a secret performed on-line using a server that verifies the secret (see Section 2), and the off-line attack is that performed off-line in parallel using recorded transcripts of a protocol. While the on-line attacks can be prevented by letting the server take appropriate intervals between invalid trials, the off-line attacks cannot be prevented by such measures since the attack is performed off-line and independently of the server. Thus the off-line attacks are critical to most of the protocols using human-memorable passwords not having enough entropy to avoid off-line exhaustive search.

While PKI (Public-Key Infrastructures) can realize an authenticated key-exchange or key-transport (being secure against off-line attacks) like SSH (Secure SHell), SSL/TLS (Secure Socket Layer/Transport Layer Security), Station-to-Station protocol [6] and the protocols in [8] do, we have to recall that the receivers of public-keys must verify them using the fingerprints (digests) of them or the verification keys of digital signatures attached with them. This means the entities must carry about something, which is hard to remember. On the other hand, PAKE (Password-Authenticated Key-Exchange) protocols do not require

its entities to carry something hard to remember (except a password) to verify something.

The studies on the PAKE with formal security proof have appeared in [7, 13, 12, 9, 1, 5, 14, 11]. Unfortunately, they are either by far inefficient or the proofs are given only in the random oracle model. In the random oracle model, the mapping of the underlying hash and encryption functions is assumed not to be fixed in advance, and then gradually determined by the random oracle at random every after the evaluation of them. And then simulators (for proving the security reduction) are assumed to know all the evaluated input-output pairs of the functions by simulating the random oracles [2]. While the proof in the random oracle model may give one reason to conjecture that the practical version (which uses conventional fixed functions instead of random oracles) might also be secure, it does not give any formal validation of the security of the practical version.

On the other hand, [7, 9] give their security proofs in the standard model where the mapping of the underlying hash and encryption functions is fixed in advance, and then simulators for showing the security reduction do not need to know the evaluated input-output pairs of the functions. Unfortunately, the protocol proposed in [7] is too inefficient to use in practice since it employs techniques from generic multi-party computations, such as non-malleable commitments, secure polynomial evaluations and zero-knowledge proofs. While [9] is more efficient than [7], it still requires large communication costs and computation costs.

In this paper, we propose a more efficient protocol that is also provably secure in the standard model. Comparative results with the previous schemes [7, 9] are summarized in Table 1. As shown in the table, our protocol is efficient in both the communication costs and the computation costs. It requires only about 2.34 modular exponentiations of each entity whereas more than 6.5 modular exponentiations are required in the previous schemes. If pre-computation is used, ours requires only 1 and 2 modular exponentiations of the server and of the client respectively, whereas more than 3.2 and 4.2 modular exponentiations are required of them respectively in the previous schemes. (The difference between the server and the client in the pre-computation phase is whether passwords are stored in advance or given every time.)

Ours has an advantage in communication costs too. It requires only 4 data unit exchange whereas more than 11 data unit exchange is required in the previous schemes where one data unit denotes either a member of the underlying field or one hashed value, such as a MAC. In addition, our protocol ends in only one round in parallel since both y_1 and y_2 in our protocol can be calculated independently and then sent independently. The implementation overhead of our protocol is very small since the clients and the servers use almost the same algorithm, and it can be obtained with small modification of the widely used Diffie-Hellman key-exchange protocol.

Our protocol has a formal validation of security in the standard model. The intuitive explanation of the result is that even if adversaries can abuse entities as oracles, the possibility for them obtaining some significant information on

Table 1. Comparison of PAKEs proven to be secure in the standard model

Schemes	Computation Costs*1		Communication Costs*2		Core Hard Problems*3
	Client C	Server S	C to S	S to C	
Efficient Construction of [7]*4	*5 $\geq (m+1)n, (\geq mn)$	*5 $\geq 2n, (\geq n)$	*5 $\geq (m+1)n$	*5 $\geq 2m+n$	ITP, PR and DDH *6
KOY [9]	$\geq 7.75, (\geq 4.29)$	$\geq 6.58, (\geq 3.29)$	≥ 6	5	DDH
Our Proposal	$2.34, (2)$	$2.34, (1)$	2	2	DDH

*1: The number of modular exponentiations where the costs for one simultaneous calculation of two bases and five bases are converted into 1.17 and 1.29, respectively [15]. The figures in the parentheses are the remaining costs after pre-computation.
*2: The number of data units to be sent where one data unit denotes either a member of the underlying field or one hashed value, such as a MAC.
*3: In addition to the core hard problems, all the schemes commonly require: (1) Passwords chosen securely against on-line attacks, (2) Unforgeable MACs or signatures against chosen-message attacks.
*4: Efficient construction using the polynomial evaluation in [16] and the efficient oblivious transfer in [18].
*5: Only the costs in the pre-key exchange phase are shown. Both n and m depends on the security parameter. Currently, at least $\binom{m}{n_1} > 2^{80}$ must hold for $n \geq n_1$ to make the underlying polynomial reconstruction problem hard (which is required in the efficient polynomial evaluation)[4].
*6: ITP and PR denote Inversion of Trapdoor Permutation and Polynomial Reconstruction, respectively. Inefficient construction of [7] assumes only trapdoor permutations as its core hard problems.

the session-key of the challenge session can be negligibly small if DDH (Decision Diffie-Hellman) problem is hard, passwords are unguessable with on-line exhaustive search and MACs are selectively unforgeable against partially chosen message attacks, (which is weaker than being existentially unforgeable against chosen message attacks).

This paper consists as follows: in Section 2, we explain both on-line and off-line attacks that are crucial to the password-based protocols. Then, in Section 3, we propose a pretty-simple protocol which has an immunity against off-line attacks. And finally, in Section 4, we show the formal validation of security of our protocol in the standard model.

2 On-line and Off-line Attacks

Since on-line attacks and off-line attacks are crucial to the password-based protocols, we explain them in this section using some examples.

At first, we consider the following password-based challenge-response protocol where a server gives a random challenge r to a client, and then the client returns the server $res := E_{pass}(r)$, the encryption of r using a pre-shared (hashed) password $pass$ as its symmetric key. An adversary, in the on-line attack, runs a

protocol with the server impersonating the client, and then tries guessed passwords $pass'$ on-line returning $res' := E_{pass'}(r)$ to the server. If it is accepted, $pass'$ is the target password with high probability.

While almost all of the password-based protocols accept this kind of attack, it can be prevented by letting the server take appropriate intervals between invalid trials. On the other hand, off-line attacks, described bellow, are more powerful since they cannot be prevented by the above measures. Adversaries, in the off-line attack, firstly obtain valid pairs of r and res by eavesdropping honest executions of the protocol, and then finds $pass'$ satisfying $res = E_{pass'}(r)$ off-line in parallel. Since the attack is performed off-line in parallel and the entropy of a password is usually not large enough, they can find it in a practical time with high probability.

The off-line attack is also applicable to DH-EKE (Diffie-Hellman Encrypted Key-Exchange) [3] if the underlying group size $\log_2 |\mathcal{G}| = \log_2 q$ is smaller than the encryption size[1]. Note that the above condition is usually true when a prime order subgroup and a conventional stream cipher or a block cipher, such as AES, are used. DH-EKE is a protocol, in which two entities exchange $y_1 := E_{pass}(g^{r_1})$ and $y_2 := E_{pass}(g^{r_2})$ respectively, and then share $D_{pass}(y_1)^{r_2} = D_{pass}(y_2)^{r_1} = g^{r_1 \cdot r_2}$ as a fresh secret where g is a generator of a finite cyclic group $\mathcal{G} =< g >$, $E_{pass}()$ and $D_{pass}()$ are encryption and decryption functions using a (hashed) password $pass$ as its symmetric key. The off-line attack on DH-EKE is performed as follows: adversaries obtain some y_1 and y_2 eavesdropping the protocol, and then see off-line whether $D_{pass'}(y_1)$ and $D_{pass'}(y_2)$ (for obtained y_1 and y_2) represent right members in $\mathcal{G}$ for guessed passwords $pass'$. If at least one of them is not a right member, the guessed password is wrong. By continuing the above process, they can find the correct password.

Our protocol, proposed below, has the immunity against these off-line attacks.

3 Our Proposal: Pretty-Simple PAKE

Our protocol is defined over a finite cyclic group $\mathcal{G} =< g >$ where $|\mathcal{G}| = q$ and q is a large prime (or a positive integer divisible by a large prime). While $\mathcal{G}$ can be a group over an elliptic curve, we assume, in this paper, $\mathcal{G}$ is a prime order subgroup over a finite field F_p. That is, $\mathcal{G} = \{g^i \bmod p : 0 \leq i < q\}$ where p is a large prime number, q is a large prime divisor of $p-1$ and g is an integer such that $1 < g < p-1$, $g^q = 1$ and $g^i \neq 1$ for $0 < i < q$. A generator of $\mathcal{G}$ is any element of $\mathcal{G}$ except 1.

Both g and h are two generators of $\mathcal{G}$, chosen so that its DLP (Discrete Logarithm Problem), i.e. calculating

$$a = \log_g h, \tag{1}$$

[1] While DH-EKE is proven to be secure in the random oracle model in [1], the proof is given under the assumption that the underlying group size is at least the same as the encryption size. Thus the proof cannot be applied when the group size is smaller than the encryption size.

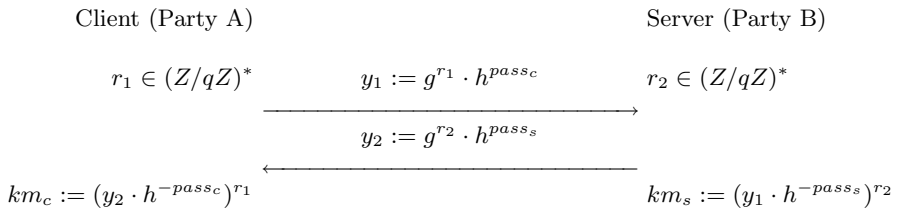

Fig. 1. Secrecy-amplification phase of our protocol

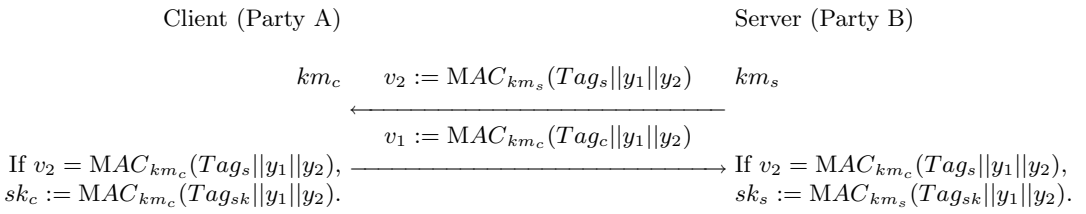

Fig. 2. Verification phase and session-key generation phase of our protocol

should be hard[2] for each entity. Both g and h may be given as system parameters or chosen with the negotiation between entities. For example, g is a random generator of $\mathcal{G}$ and $h := Hash(g)^{(p-1)/q} \bmod p$, or one entity $\mathcal{A}$ chooses $g := g_b^{s_1}$ for a random $s_1 \in (Z/qZ)^*$ and a public generator g_b, and then sends the commitment $Hash(g)$ to the other entity $\mathcal{B}$, $\mathcal{B}$ replies $h := g_b^{s_2}$ for a random $s_2 \in (Z/qZ)^*$, and finally $\mathcal{A}$ reveals g to $\mathcal{B}$.

The protocol consists of the following three phases: a secrecy-amplification phase, a verification phase and a session-key generation phase. In the secrecy-amplification phase, the secrecy of the pre-shared weak secret, i.e. a human memorable password that may be vulnerable against off-line attacks, is amplified to a strong secret (we call it a keying material) that is secure even against off-line attacks. In the verification phase, entities confirm whether they can share the same keying material or not using a challenge-response protocol with the keying material as its key. In the session-key generation phase, a session-key is generated using the keying material.

3.1 Secrecy-Amplification Phase

The secrecy-amplification phase is illustrated in Fig. 1. The client chooses a random number $r_1 \in (Z/qZ)^*$ and then calculates $y_1 := g^{r_1} \cdot h^{pass_c}$ using its (hashed) password $pass_c$, which is shared with the server. It sends y_1 to the server. The server also calculates $y_2 := g^{r_2} \cdot h^{pass_s}$ using its (hashed) password $pass_s$ (shared with the client) and a random number $r_2 \in (Z/qZ)^*$, and then sends it to the client The client's keying material is $km_c = (y_2 \cdot h^{-pass_c})^{r_1}$ and the server's one is $km_s = (y_1 \cdot h^{-pass_s})^{r_2}$.

[2] It is reasonable to assume that DLP is hard since our protocol is based on the difficulty of DDH (Decision Diffie-Hellman) problem, and DLP is harder than DDH.

Only when they use the same password, they can share the same keying material. Otherwise guessing the other's keying material is hard due to the DLP between g and h (see also Section 4.1). Adversaries cannot determine the correct password of the other entity with off-line attacks since they cannot know the keying material of it, which is required to narrow down the password.

This phase ends in only one pass in parallel since both y_1 and y_2 can be calculated and sent independently (where g^{r_1} and y_2 are pre-computable). This speeds up the protocol. The implementation cost of this phase is very low since it can be obtained with a very small modification of widely used Diffie-Hellman key exchange protocols.

3.2 Verification Phase

This phase is illustrated in Fig. 2. In this phase, entities verify whether they share the same keying material or not with a challenge-response protocol using the keying material calculated in the secrecy-amplification phase.

The client and the server calculate $v_1 := \mathrm{M}AC_{km_c}(Tag_c||y_1||y_2)$ and $v_2 := \mathrm{M}AC_{km_s}(Tag_s||y_1||y_2)$ respectively using a MAC generation function $\mathrm{M}AC_k()$ and the keying materials as its key k. Both Tag_s and Tag_c are pre-determined distinct values, e.g. $Tag_s = (ID_c||ID_s||00)$ and $Tag_c = (ID_c||ID_s||01)$ where ID_c and ID_s are IDs of the client and the server. The client and the server exchange v_1 and v_2 each other, and then they verify $v_1 = \mathrm{M}AC_{km_c}(Tag_s||y_1||y_2)$ and $v_2 = \mathrm{M}AC_{km_s}(Tag_c||y_1||y_2)$ respectively. If at least one of them does not hold, the corresponding entities wipe off all the temporally data including the keying materials, and then close the session. Otherwise they proceed to the session-key generation phase.

Adversaries can try off-line exhaustive search for the keying material using $(Tag_c||y_1||y_2)$ and v_1 or $(Tag_s||y_1||y_2)$ and v_2. The success probability achieved within a polynomial time t can be negligible if a strong secret can be shared in the secrecy-amplification phase and an appropriate MAC generation function, whose keys are unguessable, is used.

3.3 Session-Key Generation Phase

If the above verification phase succeeds in, the entities generate their session keys using the verified keying materials as follows:

$$sk_s := \mathrm{M}AC_{km_s}(Tag_{sk}||y_1||y_2) \tag{2}$$

$$sk_c := \mathrm{M}AC_{km_c}(Tag_{sk}||y_1||y_2) \tag{3}$$

where Tag_{sk} is a pre-determined distinct value from both Tag_{v_2} and Tag_{v_1}, e.g. $Tag_{sk} = (ID_c||ID_s||11)$. The generated session keys are then used in the subsequent application.

The requirement for the MAC generation function in this phase and the previous phase is $\epsilon_{mac}(k_2, t, i)$ given in Definition 4 can be negligibly small for

practical security parameter k_2 and i (that is a polynomial of k_2) since if adversaries cannot forge a MAC corresponding to $(Tag_{sk}||y_1||y_2)$ and km_s or km_c with a significant probability, they cannot obtain any significant information of the session-key.

This requirement can be satisfied by using a universal one-way hash function [17] or by using a practical MAC generation function, such as HMAC-SHA-1 [10] (and even KeyedMD5) so far since no effective algorithms are known so far to make $\epsilon_{mac'}(k_2, t, i)$ non-negligible where $\epsilon_{mac'}(k_2, t, i)$ is given in Definition 5 and it is larger than or equal to $\epsilon_{mac}(k_2, t, i)$.

4 Security of Our Protocol

4.1 Replacement of h with g

Before we show the formal security proof of our protocol, we describe why two distinct generators, h and g, should be used (instead of one generator). It is because the following adversary $\mathcal{A}_I$ can narrow down the candidates for the keying material to at most N, the number of the possible passwords, with off-line attacks.

$\mathcal{A}_I$ runs the protocol with the target entity impersonating its partner. For simplicity, we assume $\mathcal{A}_I$ impersonates a client. $\mathcal{A}_I$ generates y_1 using randomly chosen r_1 and $pass_c$, and then sends it to the target. The keying material of the target is $km_s := (y_1 \cdot g^{-pass_s})^{r_2}$, and $\mathcal{A}_I$ can narrow down its candidates to at most N since

$$km_s = (y_2 \cdot g^{-pass_s})^{r_1 + pass_c - pass_s} \tag{4}$$

and $\mathcal{A}_I$ knows $pass_c$, r_1 and the candidates for $pass_s$, which is at most N.

If N is in the range of off-line exhaustive search, $\mathcal{A}_I$ can determine the correct one by seeing whether $v_2 = \mathrm{MAC}_{km_s}(Tag_s||y_1||y_2)$ holds or not with off-line search.

On the other hand, in our protocol, adversaries have to find $a = \log_g h$ to narrow down the candidates for km_s since the following holds

$$km_s = (y_2 \cdot h^{-pass_s})^{r_1 + a(pass_c - pass_s)}. \tag{5}$$

4.2 Security Model and Formal Validation of Security

In order to consider a more advantageous situation for adversaries, we assume they have access to the following oracles, which were originally introduced by Bellare et al in [1], but a little bit modified for our protocol.

Execute oracle: It accepts two IDs of entities sharing the same password. Then it carries out a honest execution of the protocol between them, and outputs the corresponding transcript. This oracle ensures that adversaries are able to observe all the transcripts between any entities including the target ones.

Send oracle: It accepts an entity ID and a message that is a part of a transcript. It acts as the entity, and then outputs a completed transcript corresponding to them. This oracle ensures that adversaries are able to run a protocol with any entity impersonating its partner and obtain the corresponding transcripts.

Reveal oracle: It accepts both an entity ID and a session ID, and then reveals the corresponding session-key. (This oracle does not reveal the session-key of the challenge transcript.) Note that a session-key might be leaked out since it is used outside of the protocol in various applications that might deal it insecurely (e.g. by using it as a key of very weak encryption algorithms). Reveal oracle simulates such a situation.

Corrupt oracle: This oracle is used to see whether the protocol satisfies the forward secrecy, i.e. whether the disclosure of a long-lived secret (a password in our protocol) does not compromise the secrecy of the session-keys from earlier runs (even though that compromises the authenticity and thus the secrecy of new runs). It accepts two entity IDs and then reveals the corresponding password shared between them. This oracle can be used after the transcripts related with the target password are generated.

Test$_{sk}$ oracle: This oracle is used to see whether adversaries can obtain some information on the challenge session-key by giving a hint on it to them. It accepts an entity ID in the challenge session, and then flips a coin $b \in \{0, 1\}$. If $b = 0$, it returns the corresponding session-key. Otherwise it returns a random one except the correct session-key. This oracle can be used only once per challenge.

Test$_{km}$ oracle: Since a session-key is generated from a keying material, we prepare this oracle to see whether a strong secret can be generated in the secrecy amplification phase. This oracle accepts both an entity ID and an session ID, and then flips a coin $b \in \{0, 1\}$. If $b = 0$, it returns the corresponding keying material. Otherwise, it returns a random one except the correct keying material. Note that adversaries are not allowed to distinguish the obtained information from this oracle using $(Tag_c||y_1||y_2)$ and v_1 or $(Tag_s||y_1||y_2)$ and v_2 since it is given to see whether a strong secret can be generated in the secrecy amplification phase.

Using the above oracles, adversaries suppose to try to distinguish a session-key given by Test$_{sk}$ oracle.

At first, we define the followings:

Definition 1 (Advantage) *Let $Pr(Win)$ denote the probability that an algorithm $\mathcal{A}$ can distinguish whether a given key is the correct session-key or not. Then $Adv_{\mathcal{A}}^{ind_{sk}}$, the advantage of $\mathcal{A}$ distinguishing the session-key, is given by*

$$Adv_{\mathcal{A}}^{ind_{sk}} = 2Pr(Win) - 1. \tag{6}$$

Definition 2 (DDH Problem) *Given $g_b \in \mathcal{G}$ and $d = (d_1, d_2, d_3) = (g_b^{x_1}, g_b^{x_2}, g_b^{x_3})$ where x_3 is either $x_1 \cdot x_2$ or not with probability $1/2$, then decide whether $g_b^{x_3} = g_b^{x_1 \cdot x_2}$ or not.*

Definition 3 (Probability of Solving DDH Problem) *Let $\epsilon_{ddh}(k_1, t)$ denote the probability that the DDH problem of size $k_1 = \log_2 q$ is solved in a polynomial time t with the best known algorithm.*

The requirement for the MAC generation function in our protocol is $\epsilon_{mac}(k_2, t, i)$, given in the following Definition 4, can be negligibly small for practical security parameter k_2 and i (that is a polynomial of k_2). $\epsilon_{mac}(k_2, t, i)$ is upper bounded by $\epsilon_{mac'}(k_2, t, i)$, which is given in Definition 4 that is a more general definition.

Definition 4 (Selective UnForgeability of a MAC Against Partially Chosen Message Attack) *Let $\epsilon_{mac}(k_2, t, i)$ denote the probability that a k_2 bit length MAC of a given message can be forged in a polynomial time t with the best known algorithm that are allowed to ask at most i (which is a polynomial of k_2) queries to the following MAC generation oracle (which is available in our protocol by abusing entities or using Send, Execute and Reveal oracles). The MAC generation oracle here accepts a message m, entity $\in \{server, client\}$, target $\in \{v, sk\}$ and a bijective function $f()$ and then returns, for randomly chosen r_1 and r_2, $\mathrm{MAC}_{f(km)}(Tag_s||m||g^{r_2})$ if entity $=$ server and target $= v$, $\mathrm{MAC}_{f(km)}(Tag_c||g^{r_1}||m)$ if entity $=$ client and target $= v$, $\mathrm{MAC}_{f(km)}(Tag_{sk}||m||g^{r_2})$ if entity $=$ server and target $= sk$ or $\mathrm{MAC}_{f(km)}(Tag_{sk}||g^{r_1}||m)$ if entity $=$ client and taget $= sk$, respectively. A MAC is said to be SUF-PCMA (Selectively UnForgeable against Partially Chosen Message Attacks) if $\epsilon_{mac}(k_2, t, i)$ is negligibly small.*

Definition 5 (Existential UnForgeability of a MAC Against Chosen Message Attack) *Let $\epsilon_{mac'}(k_2, t, i)$ denote the probability that a new MAC-message pair for a k_2 bit length MAC can be generated in a polynomial time t with the best known algorithm that are allowed to ask at most i (which is a polynomial of k_2) queries to a MAC generation oracle, which accepts a message m and a bijective function $f()$ and then returns $\mathrm{MAC}_{f(km)}(m)$. A MAC is said to be EUF-CMA (Existential UnForgeable against Chosen Message Attacks) if $\epsilon_{mac'}(k_2, t, i)$ is negligibly small.*

Under the following assumption, Theorem 1 is true[3]. The intuitive interpretation of Theorem 1 is that if both N and $|\mathcal{G}|$ are large enough and both $\epsilon_{mac}(k_2, t, q_{se} + 2q_{ex} + q_{re} + 2)$ and $\epsilon_{ddh}(k_1, t)$ can be negligibly small for appropriate security parameters k_1 and k_2, the advantage for the active adversaries can be bounded by a negligibly small value.

Assumption 1 (Password) *Users' passwords are chosen uniformly at random from a set of cardinality N.*

Theorem 1 (Indistinguishability of sk) *Suppose the following adversary $\mathcal{A}$, which accepts a challenge transcript (that may be obtained by eavesdropping a*

[3] Theorem 1 can be extended easily to the case where passwords are chosen non-uniformly since the uniformity assumption of the passwords is just for simplicity.

protocol, impersonating a partner or intruding in the middle of the target entities), and then asks q_{ex}, q_{se} and q_{re} queries to the Execute, Send, Reveal oracles respectively, and finally is given sk_x by $Test_{sk}$ oracle where sk_x is either the target session-key or not with the probability $1/2$. *Then* $Adv_{\mathcal{A}}^{ind_{sk}}$, *the advantage of it to distinguish whether sk_x is the target session key or not in a polynomial time t is upper bounded by*

$$\begin{aligned} Adv_{\mathcal{A}}^{ind_{sk}} \leq\ & \varepsilon_{mac}(k_2, t, q_{se} + 2q_{ex} + q_{re} + 2) \\ & + 2(q_{se} + q_{ex} + 1) \cdot \varepsilon_{ddh}(k_1, t) \\ & + \frac{2q_{se} + 1}{N} + \frac{2(q_{se} + q_{ex})}{|\mathcal{G}|} \end{aligned} \tag{7}$$

where both k_1 and k_2 are the security parameters.

Proof.

Recall that Win is an event that $\mathcal{A}$ distinguishes sk_x correctly. Win happens either after an event KmUnknown occurs or after its compliment event $\overline{\text{KmUnknown}}$ occurs where $\overline{\text{KmUnknown}}$ is an event that $\mathcal{A}$ obtains some significant information on the keying material km in the secrecy amplification phase, and KmUnknown is an event that $\mathcal{A}$ does not obtains any significant information on the keying material km in the secrecy amplification phase. Thus $Pr(\text{Win})$ is upper bounded by

$$\begin{aligned} & Pr(\text{Win}) \\ & = Pr(\text{Win}|\text{KmUnknown})Pr(\text{KmUnknown}) \\ & \quad + Pr(\text{Win}|\overline{\text{KmUnknown}})Pr(\overline{\text{KmUnknown}}) \\ & \leq Pr(\text{Win}|\text{KmUnknown}) + Pr(\overline{\text{KmUnknown}}). \end{aligned} \tag{8}$$

We evaluate $Pr(\text{Win}|\text{KmUnknown})$ first. Even if km is unknown, the following two adversaries $\mathcal{A}_{replay}$ and $\mathcal{A}_{mac}$, can distinguish sk_x. $\mathcal{A}_{mac}$ tries to forge a MAC of $(Tag_{sk}||y_1||y_2)$, and then distinguish sk_x. $\mathcal{A}_{replay}$ tries to obtain at least one transcript coinciding with the challenge transcript using Send or Execure oracles, and then obtains the corresponding session-key, which is the same as the challenge session-key, using Reveal oracle.

Let $Pr(\text{Win}_{\mathcal{A}_{replay}})$ and $Pr(\text{Win}_{\mathcal{A}_{mac}})$ denote the probabilities of $\mathcal{A}_{replay}$ and $\mathcal{A}_{mac}$ being able to distinguish sk_x, respectively. $Pr(\text{Win}_{\mathcal{A}_{replay}})$ is upper bounded by

$$Pr(\text{Win}_{\mathcal{A}_{replay}}) \leq \frac{(q_{se} + q_{ex})}{|\mathcal{G}|} \tag{9}$$

since $\mathcal{A}_{replay}$ cannot control at least either r_1 or r_2 and can obtain at most $(q_{se}+q_{ex})$ transcripts. The upper bound of $Pr(\text{Win}_{\mathcal{A}_{mac}})$ is given by the following lemma.

Lemma 1 *Suppose the probability that an adversary $\mathcal{A}_{mac}$ can forge a k_2 bit length MAC of a given message in a polynomial time t using i message-MAC*

pairs without knowing its key is $\epsilon_{mac}(k_2, t, i)$. Then $Pr(Win_{\mathcal{A}_{mac}})$, the probability of $\mathcal{A}_{mac}$ distinguishing a given session-key without knowing its keying material is upper bounded by $1/2 + \epsilon_{mac}(k_2, t, i)/2$.

Proof.

The situation where $\mathcal{A}_{mac}$ tries to distinguish a session-key can be divided into the following four cases according to whether a MAC forged by $\mathcal{A}_{mac}$ (of the given message $(Tag_{sk}||y_1||y_2)$) is valid or not, and whether a key given by Test_{sk} oracle is correct or not, i.e. $b = 0$ or $b = 1$.

Let MACValid denote an event that the forged MAC is valid. The best strategy for $\mathcal{A}_{mac}$ to maximize the winning probability to distinguish the given key from Test_{sk} oracle is to return $b = 0$ (with the probability 1) if the generated MAC coincides with the given key, and $b = 1$ (with the probability 1) otherwise since $\mathcal{A}_{mac}$ can only know whether the generated MAC and the given key coincide or not, and then the probabilities they coincide and they do not are given by

$$\begin{aligned} &Pr(b = 0, \text{MACValid}) \\ &+Pr(b = 1, \overline{\text{MACValid}}) \cdot \frac{1}{2^{k_2} - 1} \end{aligned} \tag{10}$$

and

$$\begin{aligned} &Pr(b = 0, \overline{\text{MACValid}}) \\ &+Pr(b = 1, \text{MACValid}) \\ &+Pr(b = 1, \overline{\text{ForgeMAC}}) \cdot \frac{2^{k_2} - 2}{2^{k_2} - 1} \end{aligned} \tag{11}$$

respectively where $Pr(b = 0, \text{MACValid}) > Pr(b = 1, \overline{\text{MACValid}}) \cdot \frac{1}{2^{k_2}-1}$ and $Pr(b = 0, \overline{\text{MACValid}}) > Pr(b = 1, \text{MACValid}) + Pr(b = 1, \overline{\text{ForgeMAC}}) \cdot \frac{2^{k_2}-2}{2^{k_2}-1}$ hold as long as $\epsilon_{mac}(k_2, t, i) > \frac{1}{2^{k_2}}$.

This give the following probability

$$Pr(\text{Win}_{\mathcal{A}_{mac}} \mid b = 0, \text{MACValid}) = 1, \tag{12}$$

$$Pr(\text{Win}_{\mathcal{A}_{mac}} \mid b = 1, \text{MACValid}) = 1, \tag{13}$$

$$Pr(\text{Win}_{\mathcal{A}_{mac}} \mid b = 0, \overline{\text{MACValid}}) = 0, \tag{14}$$

$$\begin{aligned} Pr(\text{Win}_{\mathcal{A}_{mac}} \mid b = 1, \overline{\text{MACValid}}) \\ = \frac{2^{Len(sk)} - 2}{2^{Len(sk)} - 1}, \end{aligned} \tag{15}$$

And thus $Pr(\text{Win}_{\mathcal{A}_{mac}})$ is upper bounded by

$$\begin{aligned} &Pr(\text{Win}_{\mathcal{A}_{mac}}) \\ &= Pr(\text{Win}_{\mathcal{A}_{mac}} | b = 0, \text{MACValid}) \\ &\quad \cdot Pr(b = 0) \cdot Pr(\text{MACValid}) \end{aligned}$$

$$
\begin{aligned}
&+Pr(\mathrm{Win}_{\mathcal{A}_{mac}}|b=1,\mathrm{MACValid})\\
&\cdot Pr(b=1)\cdot Pr(\mathrm{MACValid})\\
&+Pr(\mathrm{Win}_{\mathcal{A}_{mac}}|b=0,\overline{\mathrm{MACValid}})\\
&\cdot Pr(b=0)\cdot Pr(\overline{\mathrm{MACValid}})\\
&+Pr(\mathrm{Win}_{\mathcal{A}_{mac}}|b=1,\overline{\mathrm{MACValid}})\\
&\cdot Pr(b=1)\cdot Pr(\overline{\mathrm{MACValid}})\\
=\ & Pr(\mathrm{MACValid})\\
&+Pr(\mathrm{Win}_{\mathcal{A}_{mac}}|b=1,\overline{\mathrm{MACValid}})\\
&\cdot Pr(b=1)\cdot Pr(\overline{\mathrm{MACValid}})\\
\leq\ & \epsilon_{mac}(k_2,t,i)+\frac{2^{k_2}-2}{2^{k_2}-1}\cdot\frac{1}{2}\cdot\{1-\epsilon_{mac}(k_2,t,i)\}\\
\leq\ & \frac{1}{2}+\frac{\epsilon_{mac}(k_2,t,i)}{2} \qquad (16)
\end{aligned}
$$

□

$\mathcal{A}_{mac}$ can obtain at most $q_{se}+2q_{ex}+q_{re}$ message-MAC pairs using Send, Execute, Reveal oracles, and at most 2 message-MAC pairs from a challenge transcript. Thus

$$i = q_{se}+2q_{ex}+q_{re}+2. \qquad (17)$$

By substituting (17) for (16) and summing up (9) and (16), we can obtain

$$
\begin{aligned}
&Pr(\mathrm{Win}|\mathrm{KmUnknown})\\
&\leq \frac{(q_{se}+q_{ex})}{|\mathcal{G}|}+\frac{1}{2}\\
&\quad +\frac{\epsilon_{mac}(k_2,t,q_{se}+2q_{ex}+q_{re}+2)}{2}. \qquad (18)
\end{aligned}
$$

Next we evaluate $Pr(\overline{\mathrm{KmUnknown}})$, the possibility of $\mathcal{A}$ being able to obtain some information on the keying material km in the secrecy amplification phase. In the secrecy amplification phase, $\mathcal{A}$ can obtain g, h, y_1, y_2 (and pre-images of either y_1 or y_2 by impersonating the corresponding entity). The obtained data can be classified into the following four cases according to whether or not the passwords of the two entities coincide with each other, and whether or not the adversary knows the pre-image of either y_1 or y_2.

Case 1: Passwords of the target entity and its parter are different, i.e. $pass_c \neq pass_s$, and the adversary knows the pre-image of neither y_1 nor y_2.

Case 2: Passwords of the target entity and its parter are the same, i.e. $pass_c = pass_s$, and the adversary knows the pre-image of neither y_1 nor y_2.

Case 3: Passwords of the target entity and its parter are different, i.e. $pass_c \neq pass_s$, and the adversary knows the pre-image of either y_1 or y_2.

Case 4: Passwords of the target entity and its parter are the same, i.e. $pass_c = pass_s$, and the adversary knows the pre-image of either y_1 or y_2.

While Case 4 is the most advantageous for $\mathcal{A}$, it happens only when $\mathcal{A}$ inputs the correct password impersonating the parter of the target entity on-line. This probability is bounded by $(q_{se}+1)/N$ since $\mathcal{A}$ can try at most $q_{se}+1$ passwords on-line where q_{se} passwords are tried using Send oracle and 1 using the challenge session. The other cases happen with more high probabilities. For example, Case 1 and 2 happen when an adversary eavesdrops a session, or sends modified values of ever used y_1 or y_2, i.e. sends $y_1 \cdot g^{j_1} \cdot h^{j_2} \bmod p$ or $y_2 \cdot g^{j_1} \cdot h^{j_2} \bmod p$ for $j_1, j_2 \in Z/qZ$ to the target entity. Case 3 happens when an adversary generates y_1 (or y_2) from its pre-images and sends it to the target entity.

While Case 1 to 3 happen with high probability, distinguishing the keying material in these cases is as hard as or harder than solving DDH problem. Lemma 2 shows that distinguishing it in Case 2 is as hard as or harder than solving DDH problem.

Lemma 2 *Suppose there exists an algorithm $\mathcal{A}_1$, which accepts a challenge transcript g, h, y_1 and y_2 between the entities sharing the same password, and is given a hint km_x from $Test_{km}$ oracle where km_x is either equal to the keying material of the target entity, i.e. km_c or km_s, or not with the probability of $1/2$, and finally distinguishes whether km_x is the correct keying material or not in at most τ steps and with the advantage of ϵ. Then one can construct an algorithm $\mathcal{B}_1$ which runs in τ' steps and solves a given DDH problem with the advantage of ϵ' where*

$$\epsilon' = \epsilon, \tag{19}$$

$$\tau' = \tau + Poly(k_1) \tag{20}$$

and $Poly(k_1)$ is a polynomial of a security parameter $k_1 = \log_2 q$.

Proof.

$\mathcal{B}_1$ can be constructed as follows. At first $\mathcal{B}_1$ receives a DDH set g_b and $d = (d_1, d_2, d_3) = (g_b^{x_1}, g_b^{x_2}, g_b^{x_3})$. $\mathcal{B}_1$ chooses a random password $pass_s = pass_c$ and a random generator $h \in \mathcal{G}$, and then gives $g := g_b$, h, $y_1 := d_1 \cdot h^{pass_c}$, $y_2 := d_2 \cdot h^{pass_s}$ and $km_x := d_3$ to $\mathcal{A}_1$. If the answer of $\mathcal{A}_1$ is $km_x = km_c$ (which also means $km_x = km_s$), $\mathcal{B}_1$ returns $d_3 = g_b^{x_1 \cdot x_2}$. Otherwise it returns $d_3 \neq g_b^{x_1 \cdot x_2}$.

$\mathcal{B}_1$ can solve the DDH problem with the same advantage as ϵ since $d_3 = g_b^{x_1 \cdot x_2}$ holds with probability 1 if $km_x = km_s = km_c$. The number of steps required for $\mathcal{B}_1$ is mainly consumed in the calculation of h^{pass_c} and h^{pass_s} which ends in polynomial steps of $k_1 = \log_2 q$. Thus $\tau' = \tau + Poly(k_1)$.

□

Lemma 3 shows that distinguishing the keying material of the server (impersonating a client) in Case 3 is as hard as or harder than solving DDH problem. This also means distinguishing the client's keying material impersonating a server is as hard as or harder than solving DDH problem. (The corresponding proof can be obtained by replacing r_1 and $pass_c$ in the following proof with r_2 and $pass_s$ respectively, due to the symmetry of our protocol.)

Lemma 3 *Suppose there exists an algorithm $\mathcal{A}_2$, which accepts g, h, y_2, y_1, r_1, $pass_c$ and km_x where g, h, y_2 and y_1 are a challenge transcript between entities that does not share the same password, r_1 and $pass_c$ are the pre-image of y_1, and km_x is a hint given by $Test_{km}$ oracle, which is either km_s or not with the probability of $1/2$, and finally distinguishes whether $km_x = km_s$ or not in at most τ steps and with the advantage of ϵ. Then one can construct an algorithm $\mathcal{B}_2$ which runs in τ' steps and solves a given DDH problem with the advantage of ϵ' where*

$$\epsilon' = \epsilon, \tag{21}$$
$$\tau' = \tau + Poly(k_1) \tag{22}$$

and $Poly(k_1)$ is a polynomial of a security parameter $k_1 = \log_2 q$.

$\mathcal{B}_2$ can be constructed as follows. At first $\mathcal{B}_2$ receives a DDH set, g_b and $d = (d_1, d_2, d_3) = (g_b^{x_1}, g_b^{x_2}, g_b^{x_3})$. It chooses a random number $r_1 \in (Z/qZ)^*$, two distinct passwords $pass_c$ and $pass_s$, and then gives $\mathcal{A}_2$ $g := g_b$, $h := d_2$, $y_2 := d_1 h^{pass_s}$, $y_1 := g^{r_1} h^{pass_c}$, r_1, $pass_c$ and $km_x = d_1^{r_1} \cdot d_3^{(pass_c - pass_s)}$. If the answer of $\mathcal{A}_2$ is $km_x = km_s$, $\mathcal{B}_2$ returns $d_3 = g_b^{x_1 \cdot x_2}$. Otherwise it returns $d_3 \neq g_b^{x_1 \cdot x_2}$.

$\mathcal{B}_2$ can solve the DDH problem with the same advantage as ϵ since

$$km_x = d_1^{r_1} \cdot d_3^{(pass_c - pass_s)}, \tag{23}$$
$$\begin{aligned} km_s &= g^{x_1 \cdot r_1} \cdot h^{(pass_c - pass_s)x_1} \\ &= d_1^{r_1} \cdot g^{x_2 (pass_c - pass_s) x_1}, \end{aligned} \tag{24}$$

and $d_3 = g_b^{x_1 x_2}$ holds if $km_s = km_x$. The number of steps required for $\mathcal{B}_2$ is mainly consumed in the calculation of y_1, y_2 and km_x which ends in polynomial steps of $k_1 = \log_2 q$. Thus $\tau' = \tau + Poly(k_1)$.

□

Distinguishing the target keying material in Case 1 is as hard as or harder than doing that in Case 3 since the pre-images of y_1 and y_2 are not given to the adversaries in Case 1. The corresponding proof can be obtained simply by removing y_1 and $pass_c$ from the inputs of $\mathcal{B}_2$ in the proof of Lemma 3.

From the above discussion and Definition 3, the probability that one can obtain some information on the keying material from one transcript in Case 1 to 3 is upper bounded by $\epsilon_{ddh}(k_1, t)$. In total, $\mathcal{A}$ can obtain at most $q_{se} + q_{ex} + 1$ transcripts where $q_{se} + q_{ex}$ can be obtained using Send and Execute oracles, and 1 from a challenge transcript. Thus the probability of $\mathcal{A}$ being able to obtain some information on the challenge keying material in Case 1 to 3 is upper bounded by $(q_{se} + q_{ex} + 1) \cdot \epsilon_{ddh}(k_1, t)$. And then the probability of $\mathcal{A}$ being able to obtain it in the secrecy amplification phase is upper bounded by

$$\begin{aligned} &Pr(\overline{KmUnknown}) \\ &\leq \frac{q_{se} + 1}{N} + (q_{se} + q_{ex} + 1) \cdot \epsilon_{ddh}(k_1, t). \end{aligned} \tag{25}$$

By substituting (18) and (25) for (8), the upper bound of $Pr(\text{Win})$ is given by

$$\begin{aligned} Pr(\text{Win}) \leq & \frac{(q_{se}+q_{ex})}{|\mathcal{G}|} + \frac{1}{2} + \frac{q_{se}+1}{N} \\ & + \frac{\epsilon_{mac}(k_2, t, q_{se}+2q_{ex}+q_{re}+2)}{2} \\ & + (q_{se}+q_{ex}+1) \cdot \epsilon_{ddh}(k_1, t). \end{aligned} \tag{26}$$

(7) can be obtained by substituting (26) for (6).

□

5 Extension to Server Compromise

The system is said to be secure against server compromise if the off-line exhaustive search for the password is the best attack when an adversary obtains a signature of the password of a user. Note that the signature of the password means all the necessary information for the server to verify the user, and it includes enough information to perform the off-line exhaustive search for the password.

If one wants to enhance our protocol to the server compromise, the following extension is available. The server stores $V_s := h^{pass_s}$ as the signature of the password for the user. In the case of authentication, the server generates a random number $r_3 \in (Z/qZ)^*$ in addition to r_2 and sends $y_3 := g^{r_3}$ with y_2. Both the client and the server calculate $km_c := \{(y_2 \cdot h^{-pass_c})^{r_1} || y_3^{pass_c}\}$ and $km_s := \{(y_1 \cdot h^{-pass_s})^{r_2} || V_s^{r_3}\}$, respectively, and then include y_3 in each MAC like $\text{MAC}_{km}(Tag||y_1||y_2||y_3)$.

6 Conclusion

We proposed a pretty-simple password-authenticated key-exchange protocol that is proven to be secure in the standard model (instead of the random oracle model) under the following three assumptions. (1) DDH (Decision Diffie-Hellman) problem is hard. (2) The entropy of the password is large enough to avoid on-line exhaustive search (but not necessarily off-line exhaustive searches). (3) MAC is selectively unforgeable against partially chosen message attacks, (which is weaker than existentially unforgeable against chosen message attacks).

Our protocol is almost as efficient as Diffie-Hellman key-exchange, and can be implemented easily with a small modification of it.

References

1. M. Bellare, D. Pointcheval, and P. Rogaway. "Authenticated key exchange secure against dictionary attack". In *Proc. of EUROCRYPT 2000: LNCS 1807*, pages 139–155, 2000.

2. M. Bellare and P. Rogaway. "Random oracles are practical: A paradigm for designing efficient protocols". In *Proc. of the First ACM CCCS*, pages 62–73, 1993.
3. S. Bellovin and M. Merritt. "Encrypted key exchange: Password-based protocols secure against dictionary attacks". In *Proc. of IEEE Symposium on Security and Privacy*, pages 72–84, 1992.
4. D. Bleichenbacher and P. Nguyen. "Noisy polynomial interpolation and noisy chinese remaindering". In *Proc. of EUROCRYPT 2000: LNCS 1807*, pages 53–69, 2000.
5. V. Boyko, P. MacKenzie, and S. Patel. "Provably secure password authenticated key exchange using Diffie-Hellman". In *Proc. of EUROCRYPT 2000: LNCS 1807*, pages 156–171, 2000.
6. W. Diffie, P.C. van Oorschot, and M. Wiener. "Authentication and authenticated key exchanges". *Designs Codes and Cryptography*, 2, 1992.
7. O. Goldreich and Y. Lindell. "Session-key generation using human passwords only". In *Proc. of CRYPTO 2001*, pages 408–432, 2001.
8. S. Halevi and H. Krawczyk. "Public-key cryptography and password protocols". In *Proc. of ACM Conference on Computer and Commuinication s Security*, 1998.
9. J. Katz, R. Ostrovsky, and M. Yung. "Efficient password-authenticated key exchange using human-memorable passwords". In *Proc. of EUROCRYPT 2001: LNCS 2045*, pages 475–494, 2001.
10. H. Krawczyk, M. Bellare, and R. Canetti. "HMAC: Keyed-hashing for message authentication". *RFC 2104*, 1997.
11. T. Kwon. "Authentication and key agreement via memorable password". In *Proc. of NDSS 2001 Symposium Conference*, 2001.
12. P. MacKenzie. "More efficient password-authenticated key exchange". In *Proc. of Topics in Cryptology – CT-RSA 2001 : LNCS 2020*, pages 361–377, 2001.
13. P. MacKenzie. "On the security of the SPEKE password-authenticated key exchange protocol". In *IACR ePrint archive, http://eprint.iacr.org/2001/057/*, 2001.
14. P. MacKenzie, S. Patel, and R. Swaminathan. "Password-authenticated key exchange based on RSA". In *Proc. of ASIACRYPT 2000*, pages 599–613. Springer–Verlag, 2000.
15. A. J. Menezes, P. C. Oorschot, and S. A. Vanstone. "Simultaneous multiple exponentiation". In *"Handbook of Applied Cryptography"*, pages 617–619. CRC Press, 1997.
16. M. Naor and B. Pinkas. "Oblivious transfer and polynomial evaluation". In *Proc. 30th ACM Symp. on Theory of Computing*, pages 245–254, 1999.
17. M. Naor and M. Yung. "Universal one-way hash functions and their cryptographic applications". In *Proc. of STOC '98*, pages 33–43, 1998.
18. Wen-Guey Tzeng. "Efficient 1-out-n oblivious transfer schemes". In *Proc. of PKC 2002, LNCS 2274*, pages 159–171. Springer–Verlag, 2002.