New Security Results on Encrypted Key Exchange

Emmanuel Bresson[1], Olivier Chevassut[2], and David Pointcheval[3]

[1] Dépt Cryptologie, CELAR, 35174 Bruz Cedex, France
Emmanuel.Bresson@polytechnique.org.
[2] Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA,
OChevassut@lbl.gov – http://www.itg.lbl.gov/∼chevassu.
[3] Dépt d'informatique, École normale supérieure, 75230 Paris Cedex 05, France
David.Pointcheval@ens.fr – http://www.di.ens.fr/users/pointche.

Abstract. Schemes for *encrypted key exchange* are designed to provide two entities communicating over a public network, and sharing a (short) password only, with a session key to be used to achieve data integrity and/or message confidentiality. An example of a very efficient and "elegant" scheme for *encrypted key exchange* considered for standardization by the IEEE P1363 Standard working group is AuthA. This scheme was conjectured secure when the symmetric-encryption primitive is instantiated via either a cipher that closely behaves like an "ideal cipher", or a mask generation function that is the product of the message with a hash of the password. While the security of this scheme in the former case has been recently proven, the latter case was still an open problem. For the first time we prove in this paper that this scheme is secure under the assumptions that the hash function closely behaves like a random oracle and that the computational Diffie-Hellman problem is difficult. Furthermore, since Denial-of-Service (DoS) attacks have become a common threat we enhance AuthA with a mechanism to protect against them.

1 Introduction

The need for authentication is obvious when two entities communicate on the Internet. However, proving knowledge of a secret over a public link without leaking any information about this secret is a complex process. One extreme example is when a short string is used by a human as a means to get access to a remote service. This password is used by the human to authenticate itself to the remote service in order to establish a session key to be used to implement an authenticated communication channel within which messages set over the wire are cryptographically protected. Humans directly benefit from this approach since they only need to remember a low-quality string chosen from a relatively small dictionary (i.e. 4 decimal digits).

The seminal work in this area is the *Encrypted Key Exchange* (EKE) protocol proposed by Bellovin and Merritt in [5, 6]. EKE is a classical Diffie-Hellman key exchange wherein the two flows are encrypted using the password as a common symmetric key. This encryption primitive can be instantiated via either a password-keyed symmetric cipher or a mask generation function computed as the product of the message with a hash of the password. This efficient structure later evolved into a protocol named AuthA considered for standardization by the IEEE P1363 Standard working group on public-key cryptography [3]. AuthA was conjectured secure against dictionary attacks by its designers, but actually proving it was left as an open problem.

Cryptographers have began to analyze the AuthA protocol in an ideal model of computation wherein a hash function is modeled via a random function and a block cipher is modeled via random permutations [2, 5, 8]. These analyses have provided useful arguments in favor of AuthA, but do not guarantee that AuthA is secure in the real world. These analyses only show that AuthA is secure against generic attacks that do not exploit a particular implementation of the block cipher, but in practice current block ciphers are far from being random permutations. A security proof in the random-oracle model only, while still using ideal objects, would provide a stronger and more convincing argument in favor of AuthA.

One should indeed note that the ideal-cipher model seems to be a stronger model than the random-oracle one. Even if one knows constructions to build random permutations from random functions [13], they cannot be used to build ideal ciphers from random oracles. The difference here comes from the fact that the inner functions (random oracles) are available to the adversary. It could compute plaintext-ciphertext relations starting from the middle of the Feistel network, while in the *programmable* ideal-cipher model, one needs to control all these relations.

Moreover, a AuthA scheme resistant to Denial-of-Service (DoS) attacks would be more suited to the computing environment we face every day since nowadays through the Internet hackers make servers incapable of accepting new connections. These so-called *Distributed DoS attacks* exhaust the memory and computational power of the servers.

Contributions. This paper examines the security of the AuthA password-authenticated key exchange protocol in the random-oracle model under the computational Diffie-Hellman assumption; no ideal-cipher assumption is needed. We work out our proofs by first defining the execution of AuthA in the communication model of Bellare *et al.* [2] and then adapting the proof techniques recently published by Bresson *et al.* [8]. We exhibit very compact and "elegant" proofs to show that the *One-Mask* (OMDHKE– one flow is encrypted only) and the *Two-Mask* (MDHKE– both flows are encrypted) formal variants of AuthA and EKE are secure in the random-oracle model when the encryption primitive is a mask generation function. Because of lack of space, the latter variant is postponed to the full version of this paper [9].

We define the execution of AuthA in the Bellare *et al.*'s model wherein the protocol entities are modeled through oracles, and the various types of attacks are modeled by queries to these oracles. This model enables a treatment of dictionary attacks by allowing the adversary to obtain honest executions of the

AuthA protocol. The security of AuthA against dictionary attacks depends on how many interactions the adversary carries out against the protocol entities rather than on the adversary's computational power.

We furthermore enhance the schemes with a mechanism that offers protection against Denial-of-Service (DoS) attacks. This mechanism postpones the computation of any exponentiations on the server side, as well as the storage of any states, after that the initiator of the connection has been identified as being a legitimate client. Roughly speaking, the server sends to the client a "puzzle" [12] to solve which will require from the client to perform multiple cryptographic computations while the server can easily and efficiently check that the solution is correct.

Related Work. The IEEE P1363.2 Standard working group on password-based authenticated key-exchange methods [11] has been focusing on key exchange protocols wherein clients use short passwords in place of certificates to identify themselves to servers. This standardization effort has its roots in the works of Bellare *et al.* [2] and Boyko *et al.* [7], wherein formal models and security goals for password-based key agreement were first formulated. Bellare *et al.* analyzed the EKE (where EKE stands for *Encrypted Key Exchange*) protocol [5], a classical Diffie-Hellman key exchange wherein the two flows are encrypted using the password as a common symmetric key. Several proofs have already been proposed, in various models, but all very intricate. The present paper provides a very short and "elegant" proof of AuthA or OMDHKE (but also of EKE or MDHKE in the full version), that is less prone to errors.

Several works have already focused on designing mechanisms to protect against DoS attacks. Aiello *et al.* [1] treat the amount of Perfect Forward-Secrecy (PFS) as an engineering parameter that can be traded off against resistance to DoS attacks. DoS-resistance is achieved by saving the "state" of the current session in the protocol itself (i.e., in the flows) rather than on the server side. More precisely, the "state" of the protocol is hashed and put into a *cookie*, while the server needs only to memorize the hash value. Only once this is done, the server saves the full state and the connection is established. This technique prevents the attacker from exhausting the server's memory but do not prevent it from exhausting the server's computational power. One approach to counter the latter threat is to make the client compute some form of proof of computational effort, using a "puzzle" [12], also more recently used by Dwork *et al.* [10] to discourage spam. The present paper builds on that latter concept.

2 The OMDHKE Protocol: One-Mask Diffie-Hellman Key Exchange

The arithmetic is in a finite cyclic group $\mathbb{G} = \langle g \rangle$ of order a ℓ-bit prime number q, where the operation is denoted multiplicatively. We also denote by $\mathbb{G}^\star$ the subset $\mathbb{G}\backslash\{1\}$ of the generators of $\mathbb{G}$. Hash functions from $\{0,1\}^\star$ to $\{0,1\}^{\ell_i}$ are denoted $\mathcal{H}_i$, for $i = 0, 1$. While $\mathcal{G}$ denotes a full-domain hash function from $\{0,1\}^\star$ into $\mathbb{G}$.

Client A		Server S
	Initialization	
	$pw \in \mathsf{Password}$, $\mathsf{PW} = \mathcal{G}(pw) \in \mathbb{G}$	
$\mathsf{accept} \leftarrow \mathsf{terminate} \leftarrow \mathsf{false}$		$\mathsf{accept} \leftarrow \mathsf{terminate} \leftarrow \mathsf{false}$
$x \stackrel{R}{\leftarrow} \mathbb{Z}_q$, $X \leftarrow g^x$		
$X^\star \leftarrow X \times \mathsf{PW}$	$\xrightarrow{A,\, X^\star}$	$X \leftarrow X^\star/\mathsf{PW}$
		$y \stackrel{R}{\leftarrow} \mathbb{Z}_q$, $Y \leftarrow g^y$
		$K_S \leftarrow X^y$
$K_A \leftarrow Y^x$	$\xleftarrow{S,\, Y,\, \mathsf{Auth}_S}$	$\mathsf{Auth}_S \leftarrow \mathcal{H}_1(A \Vert S \Vert X^\star \Vert Y \Vert \mathsf{PW} \Vert K_S)$
$\mathsf{Auth}_S \stackrel{?}{=} \mathcal{H}_1(A \Vert S \Vert X^\star \Vert Y \Vert \mathsf{PW} \Vert K_A)$		
if true, $\mathsf{accept} \leftarrow \mathsf{true}$		$\mathsf{accept} \leftarrow \mathsf{true}$
$sk_A \leftarrow \mathcal{H}_0(A \Vert S \Vert X^\star \Vert Y \Vert \mathsf{PW} \Vert K_A)$		$sk_S \leftarrow \mathcal{H}_0(A \Vert S \Vert X^\star \Vert Y \Vert \mathsf{PW} \Vert K_S)$
$\mathsf{terminate} \leftarrow \mathsf{true}$		$\mathsf{terminate} \leftarrow \mathsf{true}$

Fig. 1. An execution of the protocol OMDHKE, run between a client and a server.

As illustrated on Figure 1 (with an honest execution of the OMDHKE protocol), the protocol runs between two parties A and S, and the session-key space **SK** associated to this protocol is $\{0,1\}^{\ell_0}$ equipped with a uniform distribution.

The parties initially share a low-quality string pw, the password, drawn from the dictionary Password according to the distribution $\mathcal{D}_{pw}$. In the following, we use the notation $\mathcal{D}_{pw}(q)$ for the probability to be in the most probable set of q passwords:

$$\mathcal{D}_{pw}(q) = \max_{P \subseteq \mathsf{Password}} \left\{ \Pr_{pw \in \mathcal{D}_{pw}} [pw \in P \mid \#P \leq q] \right\}.$$

Note that if we denote by $\mathcal{U}_N$ the uniform distribution among N passwords, $\mathcal{U}_N(q) = q/N$.

The protocol then runs as follows. The client chooses at random a private random exponent x and computes the corresponding Diffie-Hellman public value g^x, but does not send this last value in the clear. The client encrypts the Diffie-Hellman public value using a mask generation function as the product of a Diffie-Hellman value with a full-domain hash of the password. Upon receiving this encrypted value, the server unmasks it and computes the Diffie-Hellman secret value g^{xy} which is used by the server to compute its authenticator Auth_S and the session key. The server sends its Diffie-Hellman public value g^y in the clear, Auth_S, and terminates the execution of the protocol. Upon receiving these values, the client computes the secret Diffie-Hellman value and checks that the authenticator Auth_S is a valid one. If the authenticator is valid, the client computes the session key, and terminates the execution of the protocol.

3 The Formal Model

The security model is the same as the one defined by Bellare *et al.* [2]. We briefly review it.

The Security Model. We denote by A and S two parties that can participate in the key exchange protocol P. Each of them may have several *instances* called oracles involved in distinct, possibly concurrent, executions of P. We denote A (resp. S) instances by A^i (resp. S^j), or by U when we consider any user instance. The two parties share a low-entropy secret pw which is drawn from a small dictionary $\mathsf{Password}$, according to the distribution $\mathcal{D}_{pw}$.

The key exchange algorithm P is an interactive protocol between A^i and S^j that provides the instances of A and S with a session key sk. During the execution of this protocol, the adversary has the entire control of the network, and tries to break the privacy of the key, or the authentication of the players. To this aim, several queries are available to it. Let us briefly recall the capability that each query captures:

– $\mathsf{Execute}(A^i, S^j)$: This query models passive attacks, where the adversary gets access to honest executions of P between the instances A^i and S^j by eavesdropping.
– $\mathsf{Reveal}(U)$: This query models the misuse of the session key by instance U (*known-key attacks*). The query is only available to $\mathcal{A}$ if the attacked instance actually "holds" a session key and it releases the latter to $\mathcal{A}$.
– $\mathsf{Send}(U, m)$: This query enables to consider active attacks by having $\mathcal{A}$ sending a message to instance U. The adversary $\mathcal{A}$ gets back the response U generates in processing the message m according to the protocol P. A query $\mathsf{Send}(A^i, \mathtt{Start})$ initializes the key exchange algorithm, and thus the adversary receives the initial flow the player A should send out to the player S.

In the active scenario, the Execute-query may at first seem useless since using the Send-query the adversary has the ability to carry out honest executions of P among parties. Yet, even in this scenario, the Execute-query is essential for properly dealing with dictionary attacks. The number q_s of Send-queries directly asked by the adversary does not take into account the number of Execute-queries. Therefore, q_s represents the number of flows the adversary has built by itself, and therefore the number of passwords it would have tried.

Security Notions. As already noticed, the aim of the adversary is to break the privacy of the session key (a.k.a., semantic security) or the authentication of the players (having a player accepting while no instance facing him). The security notions take place in the context of executing P in the presence of the adversary $\mathcal{A}$. One first draws a password pw from Password according to the distribution $\mathcal{D}_{pw}$, provides coin tosses to $\mathcal{A}$, all oracles, and then runs the adversary by letting it ask any number of queries as described above, in any order.

AKE Security. The privacy (semantic security) of the session key is modeled by the game $\mathbf{Game}^{\mathsf{ake}}(\mathcal{A}, P)$, in which one more query is available to the adversary: $\mathsf{Test}(U)$. The Test-query can be asked at most once by the adversary $\mathcal{A}$ and is only available to $\mathcal{A}$ if the attacked instance U is **Fresh** (which roughly means that the session key is not "obviously" known to the adversary.) This query is answered as follows: one flips a (private) coin b and forwards sk (the value $\mathsf{Reveal}(U)$ would output) if $b = 1$, or a random value if $b = 0$. When playing this game, the goal of the adversary is to guess the bit b involved in the Test-query, by outputting this guess b'. We denote the **AKE advantage** as the probability that $\mathcal{A}$ correctly guesses the value of b. More precisely we define $\mathsf{Adv}_P^{\mathsf{ake}}(\mathcal{A}) = 2\Pr[b = b'] - 1$. The protocol P is said to be (t, ε)-**AKE-secure** if $\mathcal{A}$'s advantage is smaller than ε for any adversary $\mathcal{A}$ running with time t.

Authentication. Another goal is to consider *unilateral authentication* of either A (A-Auth) or S (S-Auth) wherein the adversary impersonates a party. We denote by $\mathsf{Succ}_P^{\mathsf{A-auth}}(\mathcal{A})$ (resp. $\mathsf{Succ}_P^{\mathsf{S-auth}}(\mathcal{A})$) the probability that $\mathcal{A}$ successfully impersonates an A instance (resp. an S instance) in an execution of P, which means that S (resp. A) agrees on a key, while the latter is shared with no instance of A (resp. S). A protocol P is said to be (t, ε)-**Auth-secure** if $\mathcal{A}$'s success for breaking either A-Auth or S-Auth is smaller than ε for any adversary $\mathcal{A}$ running with time t.

3.1 Computational Diffie-Hellman Assumption

A (t, ε)-$\mathsf{CDH}_{g,\mathbb{G}}$ attacker, in a finite cyclic group $\mathbb{G}$ of prime order q with g as a generator, is a probabilistic machine Δ running in time t such that its success probability $\mathsf{Succ}_{g,\mathbb{G}}^{\mathsf{cdh}}(\Delta)$, given random elements g^x and g^y to output g^{xy}, is greater than ε. As usual, we denote by $\mathsf{Succ}_{g,\mathbb{G}}^{\mathsf{cdh}}(t)$ the maximal success probability over every adversaries running within time t. The CDH-Assumption states that $\mathsf{Succ}_{g,\mathbb{G}}^{\mathsf{cdh}}(t) \leq \varepsilon$ for any t/ε not too large.

4 Security Proof for the OMDHKE Protocol

In this section we show that the OMDHKE protocol distributes session keys that are semantically-secure and provides *unilateral authentication* of the server S. The specification of this protocol is found on Figure 1.

Theorem 1 (AKE/UA Security). *Let us consider the protocol* OMDHKE*, over a group of prime order* q*, where* $\mathsf{Password}$ *is a dictionary equipped with the distribution* $\mathcal{D}_{pw}$*. For any adversary* $\mathcal{A}$ *within a time bound* t*, with less than* q_s *active interactions with the parties (*Send*-queries) and* q_p *passive eavesdroppings (*$\mathsf{Execute}$*-queries), and asking* q_g *and* q_h *hash queries to* $\mathcal{G}$ *and any* $\mathcal{H}_i$ *respectively,*

$$\mathsf{Adv}_{\mathsf{omdhke}}^{\mathsf{ake}}(\mathcal{A}) \leq \frac{2q_s}{2^{\ell_1}} + 12 \times \mathcal{D}_{pw}(q_s) + 12q_h^2 \times \mathsf{Succ}_{g,\mathbb{G}}^{\mathsf{cdh}}(t + 2\tau_e) + \frac{2Q^2}{q},$$

$$\mathsf{Succ}_{\mathsf{omdhke}}^{\mathsf{S-auth}}(\mathcal{A}) \leq \frac{q_s}{2^{\ell_1}} + 3 \times \mathcal{D}_{pw}(q_s) + 3q_h^2 \times \mathsf{Succ}_{g,\mathbb{G}}^{\mathsf{cdh}}(t + 3\tau_e) + \frac{Q^2}{2q},$$

where $Q = q_p + q_s + q_g$ *and* τ_e *denotes the computational time for an exponentiation in* $\mathbb{G}$.

This theorem shows that the protocol is secure against dictionary attacks since the advantage of the adversary essentially grows with the ratio of interactions (number of Send-queries) to the number of passwords.

Proof. In this proof, we incrementally define a sequence of games starting at the real game $\mathbf{G}_0$ and ending up at $\mathbf{G}_5$. We use the Shoup's lemma [14] to bound the probability of each event in these games.

Game $\mathbf{G}_0$: This is the real protocol, in the random-oracle model. We are interested in the two following events:

– S_0 (for semantic security), which occurs if the adversary correctly guesses the bit b involved in the Test-query;
– A_0 (for S-authentication), which occurs if an instance A^i accepts with no partner instance S^j (with the same transcript $((A, X^\star), (S, Y, \mathsf{Auth}))$.)

$$\mathsf{Adv}^{\mathsf{ake}}_{\mathsf{omdhke}}(\mathcal{A}) = 2\Pr[\mathsf{S}_0] - 1 \qquad \mathsf{Succ}^{\mathsf{S-auth}}_{\mathsf{omdhke}}(\mathcal{A}) = \Pr[\mathsf{A}_0]. \tag{1}$$

Actually, in any game $\mathbf{G}_n$ below, we study the event A_n, and the restricted event $\mathsf{SA}_n = \mathsf{S}_n \wedge \neg \mathsf{A}_n$.

Game $\mathbf{G}_1$: In this game, we simulate the hash oracles ($\mathcal{G}$, $\mathcal{H}_0$ and $\mathcal{H}_1$, but also additional hash functions, for $i = 0, 1$: $\mathcal{H}'_i : \{0,1\}^\star \rightarrow \{0,1\}^{\ell_i}$ that will appear in the Game $\mathbf{G}_3$) as usual by maintaining hash lists $\Lambda_\mathcal{G}$, $\Lambda_\mathcal{H}$ and $\Lambda_{\mathcal{H}'}$ (see Figure 2). We also simulate all the instances, as the real players would do, for the Send-queries and for the Execute, Reveal and Test-queries (see Figure 3). From this simulation, we easily see that the game is perfectly indistinguishable from the real attack.

$\mathcal{G}$ and $\mathcal{H}_i$ oracles

For a hash-query $\mathcal{H}_i(q)$ (resp. $\mathcal{H}'_i(q)$), such that a record (i, q, r) appears in $\Lambda_\mathcal{H}$ (resp. $\Lambda_{\mathcal{H}'}$), the answer is r. Otherwise one chooses a random element $r \in \{0,1\}^\ell$, answers with it, and adds the record (i, q, r) to $\Lambda_\mathcal{H}$ (resp. $\Lambda_{\mathcal{H}'}$).

For a hash-query $\mathcal{G}(q)$ such that a record $(q, r, \star)$ appears in $\Lambda_\mathcal{G}$, the answer is r. Otherwise the answer r is defined according to the following rule:

▶**Rule** $\mathcal{G}^{(1)}$

Choose a random element $r \in \mathbb{G}$. The record $(q, r, \perp)$ is added to $\Lambda_\mathcal{G}$.

Note: the third component of the elements of this list will be explained later.

Fig. 2. Simulation of the hash functions

Game $\mathbf{G}_2$: For an easier analysis in the following, we cancel games in which some (unlikely) collisions appear:

– collisions on the partial transcripts $((A, X^\star), (S, Y))$. Note that transcripts involve at least one honest party, and thus one of $X^\star$ or Y is truly uniformly distributed;

Send-queries to A

We answer to the Send-queries to an A-instance as follows:

- A $\mathsf{Send}(A^i, \mathtt{Start})$-query is processed according to the following rule:
  ►Rule A1$^{(1)}$
  Choose a random exponent $\theta \in \mathbb{Z}_q$, compute $X = g^\theta$ and $X^\star = X \times \mathsf{PW}$.
  Then the query is answered with $(A, X^\star)$, and the instance goes to an expecting state.
- If the instance A^i is in an expecting state, a query $\mathsf{Send}(A^i, (S, Y, \mathsf{Auth}))$ is processed by computing the authenticator and the session key. We apply the following rules:
  ►Rule A2$^{(1)}$
  Compute $K_A = Y^\theta$.
  ►Rule A3$^{(1)}$
  Compute the authenticator and the session key:
  $\mathsf{Auth}' = \mathcal{H}_1(A\|S\|X^\star\|Y\|\mathsf{PW}\|K_A)$;
  $sk_A = \mathcal{H}_0(A\|S\|X^\star\|Y\|\mathsf{PW}\|K_A)$.
  If $\mathsf{Auth} = \mathsf{Auth}'$, the instance accepts. In any case, the instance terminates.

Send-queries to S

We answer to the Send-queries to a S-instance as follows:

- A $\mathsf{Send}(S^j, (A, X^\star))$-query is processed according to the following rules:
  ►Rule S1$^{(1)}$
  Choose a random exponent $\varphi \in \mathbb{Z}_q$, compute $Y = g^\varphi$.
  Then, the instance compute the authenticator and session key. We apply the following rules:
  ►Rule S2$^{(1)}$
  Compute $X = X^\star/\mathsf{PW}$ and $K_S = X^\varphi$.
  ►Rule S3$^{(1)}$
  Compute the authenticator and the session key:
  $\mathsf{Auth} = \mathcal{H}_1(A\|S\|X^\star\|Y\|\mathsf{PW}\|K_S)$;
  $sk_S = \mathcal{H}_0(A\|S\|X^\star\|Y\|\mathsf{PW}\|K_S)$.
  Then the query is answered with (S, Y, Auth), and the instance accepts and terminates.

Other queries

An $\mathsf{Execute}(A^i, S^j)$-query is processed using successively the above simulations of the Send-queries: $(A, X^\star) \leftarrow \mathsf{Send}(A^i, \mathtt{Start})$ and $(S, Y, \mathsf{Auth}) \leftarrow \mathsf{Send}(S^j, (A, X^\star))$, and then outputting the transcript $((A, X^\star), (S, Y, \mathsf{Auth}))$.

A $\mathsf{Reveal}(U)$-query returns the session key (sk_A or sk_S) computed by the instance U (if the latter has accepted).

A $\mathsf{Test}(U)$-query first gets sk from $\mathsf{Reveal}(U)$, and flips a coin b. If $b = 1$, we return the value of the session key sk, otherwise we return a random value drawn from $\{0, 1\}^\ell$.

Fig. 3. Simulation of the OMDHKE protocol

– collisions on the output of $\mathcal{G}$.

Both probabilities are bounded by the birthday paradox:

$$\Pr[\mathsf{Coll}_2] \leq \frac{(q_p + q_s)^2}{2q} + \frac{q_g^2}{2q}. \tag{2}$$

Game $\mathbf{G}_3$: We compute the session key sk and the authenticator Auth using the private oracles $\mathcal{H}'_0$ and $\mathcal{H}'_1$ respectively:

▶Rule A3/S3$^{(3)}$

Compute the authenticator $\mathsf{Auth} = \mathcal{H}'_1(A \| S \| X^\star \| Y)$.
Compute the session key $sk_{A/S} = \mathcal{H}'_0(A \| S \| X^\star \| Y)$.

Since we do no longer need to compute the values K_A and K_S, we can simplify the second rules:

▶Rule A2/S2$^{(3)}$

Do nothing.

Finally, one can note that the password is not used anymore either, then we can also simplify the generation of $X^\star$, using the group property of $\mathbb{G}$:

▶Rule A1$^{(3)}$

Choose a random element $x \in \mathbb{Z}_q$ and compute $X^\star = g^x$.

The games $\mathbf{G}_3$ and $\mathbf{G}_2$ are indistinguishable unless some specific hash queries are asked, denoted by event $\mathsf{AskH}_3 = \mathsf{AskH0w1}_3 \vee \mathsf{AskH1}_3$:

- $\mathsf{AskH1}_3$: $\mathcal{A}$ queries $\mathcal{H}_1(A \| S \| X^\star \| Y \| \mathsf{PW} \| K_A)$ or $\mathcal{H}_1(A \| S \| X^\star \| Y \| \mathsf{PW} \| K_S)$ for some execution transcript $((A, X^\star), (S, Y, \mathsf{Auth}))$;
- $\mathsf{AskH0w1}_3$: $\mathcal{A}$ queries $\mathcal{H}_0(A \| S \| X^\star \| Y \| \mathsf{PW} \| K_A)$ or $\mathcal{H}_0(A \| S \| X^\star \| Y \| \mathsf{PW} \| K_S)$ for some execution transcript $((A, X^\star), (S, Y, \mathsf{Auth}))$, where some party has accepted, but event $\mathsf{AskH1}_3$ did not happen.

The authenticator is computed with a random oracle that is private to the simulator, then one can remark that it cannot be guessed by the adversary, better than at random for each attempt, unless the same partial transcript $((A, X^\star), (S, Y))$ appeared in another session with a real instance S^j. But such a case has already been excluded (in Game $\mathbf{G}_2$). A similar remark can be led about the session key:

$$\Pr[\mathsf{A}_3] \leq \frac{q_s}{2^{\ell_1}} \qquad \Pr[\mathsf{SA}_3] = \frac{1}{2}. \tag{3}$$

When collisions of partial transcripts have been excluded, the event $\mathsf{AskH1}$ can be split in 3 disjoint sub-cases:

- $\mathsf{AskH1\text{-}Passive}_3$: the transcript $((A, X^\star), (S, Y, \mathsf{Auth}))$ comes from an execution between instances of A and S ($\mathsf{Execute}$-queries or forward of Send-queries, replay of part of them). This means that both $X^\star$ and Y have been simulated;

– $\mathsf{AskH1\text{-}WithA}_3$: the execution involved an instance of A, but Y has not been sent by any instance of S. This means that $X^\star$ has been simulated, but Y has been produced by the adversary;
– $\mathsf{AskH1\text{-}WithS}_3$: the execution involved an instance of S, but $X^\star$ has not been sent by any instance of A. This means that Y has been simulated, but $X^\star$ has been produced by the adversary.

Game G$_4$: In order to evaluate the above events, we introduce a random Diffie-Hellman instance (P, Q), (with both $P \in \mathbb{G}^\star$ and $Q \in \mathbb{G}^\star$, which are thus generators of $\mathbb{G}$. Otherwise, the Diffie-Hellman problem is easy.) We first modify the simulation of the oracle $\mathcal{G}$, involving the element Q. The simulation introduces values in the third component of the elements of $\Lambda_\mathcal{G}$, but does not use it.

►**Rule** $\mathcal{G}^{(4)}$

Choose a random element $k \in \mathbb{Z}_q^\star$ and compute $r = Q^{-k}$. The record (q, r, k) is added to $\Lambda_\mathcal{G}$.

We introduce the other part P of the Diffie-Hellman instance in the simulation of the party S.

►**Rule S1**$^{(4)}$

Choose a random element $y \in \mathbb{Z}_q^\star$ and compute $Y = P^y$.

It would let the probabilities unchanged, but note that we excluded the cases $\mathsf{PW} = 1$ and $Y = 1$:

$$|\Pr[\mathsf{AskH}_4] - \Pr[\mathsf{AskH}_3]| \leq \frac{q_s + q_p}{q} + \frac{q_g}{q}. \tag{4}$$

Game G$_5$: It is now possible to evaluate the probability of the event AskH (or more precisely, the sub-cases). Indeed, one can remark that the password is never used during the simulation, it can be chosen at the very end only. Then, an information-theoretic analysis can be performed, which simply uses cardinalities of some sets.

To this aim, we first cancel a few more games, wherein for some pairs $(X^\star, Y) \in \mathbb{G}^2$, involved in a communication between **an instance** S^j and either the adversary or an instance A^i, there are two distinct elements PW such that the tuple $(X^\star, Y, \mathsf{PW}, \mathsf{CDH}_{g,\mathbb{G}}(X^\star/\mathsf{PW}, Y))$ is in $\Lambda_\mathcal{H}$ (which event is denoted CollH_5):

$$|\Pr[\mathsf{AskH}_5] - \Pr[\mathsf{AskH}_4]| \leq \Pr[\mathsf{CollH}_5]. \tag{5}$$

Hopefully, event CollH_5 can be upper-bounded, granted the following Lemma:

Lemma 2. *If for some pair $(X^\star, Y) \in \mathbb{G}^2$, involved in a communication with an instance S^j, there are two elements PW_0 and PW_1 such that $(X^\star, Y, \mathsf{PW}_i, Z_i)$ are in $\Lambda_\mathcal{H}$ with $Z_i = \mathsf{CDH}_{g,\mathbb{G}}(X^\star/\mathsf{PW}_i, Y))$, one can solve the computational Diffie-Hellman problem:*

$$\Pr[\mathsf{CollH}_5] \leq q_h^2 \times \mathsf{Succ}^{\mathsf{cdh}}_{g,\mathbb{G}}(t + \tau_e). \tag{6}$$

Proof. Assume there exist such elements $(X^\star, Y = P^y) \in \mathbb{G}^2$, $\mathsf{PW}_0 = Q^{-k_0}$, and $\mathsf{PW}_1 = Q^{-k_1}$. Note that

$$\begin{aligned} Z_i &= \mathsf{CDH}_{g,\mathbb{G}}(X^\star/\mathsf{PW}_i, Y) = \mathsf{CDH}_{g,\mathbb{G}}(X^\star \times Q^{k_i}, Y) \\ &= \mathsf{CDH}_{g,\mathbb{G}}(X^\star, Y) \times \mathsf{CDH}_{g,\mathbb{G}}(Q, Y)^{k_i} = \mathsf{CDH}_{g,\mathbb{G}}(X^\star, Y) \times \mathsf{CDH}_{g,\mathbb{G}}(P, Q)^{yk_i}. \end{aligned}$$

As a consequence, $Z_1/Z_0 = \mathsf{CDH}_{g,\mathbb{G}}(P, Q)^{y(k_1-k_0)}$, and thus $\mathsf{CDH}_{g,\mathbb{G}}(P, Q) = (Z_1/Z_0)^u$, where u is the inverse of $y(k_1 - k_0)$ in $\mathbb{Z}_q$. The latter exists since $\mathsf{PW}_1 \neq \mathsf{PW}_2$, and $y \neq 0$. By guessing the two queries asked to the $\mathcal{H}_i$, one concludes the proof. □

In order to conclude, let us study separately the three sub-cases of AskH1 and then AskH0w1 (keeping in mind the absence of several kinds of collisions: for partial transcripts, for $\mathcal{G}$, and for PW in $\mathcal{H}$-queries):

– AskH1-Passive: About the passive transcripts (in which both $X^\star$ and Y have been simulated), one can state the following lemma:

  Lemma 3. *If for some pair $(X^\star, Y) \in \mathbb{G}^2$, involved in a passive transcript, there is an element* PW *such that $(X^\star, Y, \mathsf{PW}, Z)$ is in $\Lambda_{\mathcal{H}}$, with $Z = \mathsf{CDH}_{g,\mathbb{G}}(X^\star/\mathsf{PW}, Y))$, one can solve the computational Diffie-Hellman problem:*

  $$\Pr[\mathsf{AskH1\text{-}Passive}_5] \leq q_h \times \mathsf{Succ}^{\mathsf{cdh}}_{g,\mathbb{G}}(t + 2\tau_e).$$

  Proof. Assume there exist such elements $(X^\star = g^x, Y = P^y) \in \mathbb{G}^2$ and $\mathsf{PW} = Q^{-k}$. As above,

  $$Z = \mathsf{CDH}_{g,\mathbb{G}}(X^\star, Y) \times \mathsf{CDH}_{g,\mathbb{G}}(Q, Y)^k = P^{xy} \times \mathsf{CDH}_{g,\mathbb{G}}(P, Q)^{yk}.$$

  As a consequence, $\mathsf{CDH}_{g,\mathbb{G}}(P, Q) = (Z/P^{xy})^u$, where u is the inverse of yk in $\mathbb{Z}_q$. The latter exists since we have excluded the cases where $y = 0$ or $k = 0$. By guessing the query asked to the $\mathcal{H}_i$, one concludes the proof. □

– AskH1-WithA: this event may correspond to an attack where the adversary tries to impersonate S to A (break unilateral authentication). But each authenticator sent by the adversary has been computed with at most one PW value. Without any $\mathcal{G}$-collision, it corresponds to at most one pw:

  $$\Pr[\mathsf{AskH1\text{-}WithA}_5] \leq \mathcal{D}_{pw}(q_s).$$

– AskH1-WithS: The above Lemma 2, when applied to games where the event CollH_5 did not happen (and without $\mathcal{G}$-collision), states that for each pair $(X^\star, Y)$ involved in a transcript with an instance S^j, there is at most one element pw such that for $\mathsf{PW} = \mathcal{G}(pw)$ the corresponding tuple is in $\Lambda_{\mathcal{H}}$: the probability over a random password is thus less than $\mathcal{D}_{pw}(q_s)$. As a consequence,

  $$\Pr[\mathsf{AskH1\text{-}WithS}_5] \leq \mathcal{D}_{pw}(q_s).$$

Client A		*Server S*
	Initialization	
	$pw \in \mathsf{Password}$, $\mathsf{PW} = \mathcal{G}(pw) \in \mathbb{G}$	
	$f : \{0,1\}^{k_0+k_1+k_2} \longrightarrow \{0,1\}^k$, a random function with $k_0 + k_1 + k_2 \leq k$	
		$\mathsf{skS} \overset{R}{\leftarrow} \{0,1\}^{\mathsf{mac}}$
$\mathsf{accept} \leftarrow \mathsf{terminate} \leftarrow \mathsf{false}$		
	$\xrightarrow{A}$	$n_S \overset{R}{\leftarrow} \{0,1\}^{k_0}$, $N_S = n_S \Vert \mathsf{date} \in \{0,1\}^{k_0+k_1}$
	$\xleftarrow{S,\, N_S,\, \mathsf{cookie}}$	$\mathsf{cookie} \leftarrow \mathsf{MAC}_{\mathsf{skS}}(A, S, N_S)$
		$\mathsf{accept} \leftarrow \mathsf{terminate} \leftarrow \mathsf{false}$
Find $r \in \{0,1\}^{k_2}$, $f(N_S \Vert r) = 0^k$		
$x \overset{R}{\leftarrow} \mathbb{Z}_q$, $X \leftarrow g^x$, $X^\star \leftarrow X \times \mathsf{PW}$	$\xrightarrow{A,\, X^\star,\, N_S,\, r,\, \mathsf{cookie}}$	Lock record N_S in List Check whether $(N_S, \star, \star) \notin \mathsf{List}$? date is fine? $f(N_S \Vert r) = 0^k$? and $\mathsf{cookie} = \mathsf{MAC}_{\mathsf{skS}}(A, S, N_S)$? $X \leftarrow X^\star/\mathsf{PW}$, $y \overset{R}{\leftarrow} \mathbb{Z}_q$, $Y \leftarrow g^y$ $K_S \leftarrow X^y$ $\mathsf{Auth}_S \leftarrow \mathcal{H}_1(A \Vert S \Vert X^\star \Vert Y \Vert \mathsf{PW} \Vert K_S)$ $\mathsf{Auth}'_A \leftarrow \mathcal{H}_2(A \Vert S \Vert X^\star \Vert Y \Vert \mathsf{PW} \Vert K_S)$ $sk_S \leftarrow \mathcal{H}_0(A \Vert S \Vert X^\star \Vert Y \Vert \mathsf{PW} \Vert K_S)$
$K_A \leftarrow Y^x$	$\xleftarrow{S,\, Y,\, \mathsf{Auth}_S,\, N_S}$	Just store $(N_S, \mathsf{Auth}'_A, sk_S)$ in List
$\mathsf{Auth}_S \overset{?}{=} \mathcal{H}_1(A \Vert S \Vert X^\star \Vert Y \Vert \mathsf{PW} \Vert K_A)$ if true, $\mathsf{accept} \leftarrow \mathsf{true}$		
$\mathsf{Auth}_A \leftarrow \mathcal{H}_2(A \Vert S \Vert X^\star \Vert Y \Vert \mathsf{PW} \Vert K_A)$	$\xrightarrow{A,\, \mathsf{Auth}_A,\, N_S}$	$\mathsf{Auth}_A \overset{?}{=} \mathsf{Auth}'_A$ if true, $\mathsf{accept} \leftarrow \mathsf{true}$
$sk_A \leftarrow \mathcal{H}_0(A \Vert S \Vert X^\star \Vert Y \Vert \mathsf{PW} \Vert K_A)$ $\mathsf{terminate} \leftarrow \mathsf{true}$		$\mathsf{terminate} \leftarrow \mathsf{true}$

Fig. 4. An execution of the protocol OMDHKE, run between a client and a server, enhanced with mutual authentication and a denial-of-service protection.

About AskH0w1 (when the three above events did not happen), it means that only executions with an instance of S (and either A or the adversary) may lead to acceptation. Exactly the same analysis as for AskH1-Passive and AskH1-WithS leads to $\Pr[\mathsf{AskH0w1}_5] \leq \mathcal{D}_{pw}(q_s) + q_h \times \mathsf{Succ}^{\mathsf{cdh}}_{g,\mathbb{G}}(t + 2\tau_e)$. As a conclusion,

$$\Pr[\mathsf{AskH}_5] \leq 3\mathcal{D}_{pw}(q_s) + 2q_h \times \mathsf{Succ}^{\mathsf{cdh}}_{g,\mathbb{G}}(t + 2\tau_e). \tag{7}$$

Combining all the above equations, one gets the announced result. □

5 The DoS-Resistant OMDHKE Protocol

In a computing environment where Distributed DoS attacks are a continual threat, a server needs to protect itself from non-legitimate clients that will exhaust its memory and computational power. Intensive cryptographic computations (i.e. exponentiation), as well as states, are only performed after a client proves to the server that it was able to solve a given "puzzle". The "puzzle" is chosen so that the client can only solve it by exhaustive search while the server can quickly checks whether a given proposition solves it. This "puzzle" is chosen as follows.

The server first picks at random a MAC-symmetric key that it will use to authenticate *cookie*; the MAC-key is used across multiple connections. The server then forms the authenticated *cookie* which is the MAC of a random nonce and the date, and sends it to the client. The precision of the date is determined according to the level of DoS required. The use of a cookie makes the protocol stateless on the server side. Upon receiving the cookie, the client tries to find an input which hashes to the NULL value. Since this hash function is seen as a random oracle, the only way for the client to solve this "puzzle" is to run through all possible prefixed strings and query the random oracle [4]. Later in practice this function is instantiated using specific functions derived from standard hash functions such as SHA1. Once the client has found such a proof of computational effort, it sends it back with the authenticated cookie and its Diffie-Hellman public value to the server. Upon receiving these values the server checks whether the client is launching a DoS attack by initiating several connections in parallel and replaying this proof of computational effort on another connection. The server reaches this aim by locking the cookie and not admitting the same cookie twice (hence the date in this challenge is used to tune the size of the database). If all the checks verify, the server starts saving states and computing the necessary exponentiations to establish a session key. From this point on the protocol works as the original AuthA protocol, adding mutual authentication [2].

6 Conclusion

The above proof does not deal with forward-secrecy. Forward-secrecy entails that the corruption of the password does not compromise the semantic security of previously established session keys.One could easily prove that this scheme achieves forward secrecy, as in [8], while loosing a quadratic factor in the reduction.

In conclusion, this paper provides strong security arguments that support the standardization of the AuthA protocol by the IEEE P1363.2 Standard working group on password-based public key cryptography . We have presented a compact and "elegant" proof of security for the AuthA protocol [3] when the symmetric-encryption primitive is instantiated using a mask generation function, which extends our previous work when the symmetric-encryption primitive is assumed to behave like an ideal cipher [8]. The security of the protocol was indeed stated as an open problem by its designers. In our study, the symmetric encryption basic block takes the form of a multiplication in the Diffie-Hellman group. Our result is a significant departure from previous known results since the security of AuthA can now be based on weaker and more reasonable assumptions involving both the random-oracle model and the computational Diffie-Hellman problem. Moreover, we investigate and propose a practical, reasonable solution to make the protocol secure against DoS attacks. One can also find further studies on the variant in which both flows are encrypted between the client and the server in the full version of this paper [9].

Acknowledgments

The second author was supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, Mathematical Information and Computing Sciences Division, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098. This document is report LBNL-53099. Disclaimer available at http://www-library.lbl.gov/disclaimer.

References

1. W. Aiello, S. M. Bellovin, M. Blaze, R. Canetti, J. Ioannidis, A. D. Keromytis, and O. Reingold. Efficient, DoS-resistant, Secure Key Exchange for Internet Protocols. In *Proc. of the 9th CCS*, pages 48–58. ACM Press, New York, 2002.
2. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated Key Exchange Secure Against Dictionary Attacks. In *Eurocrypt '00*, LNCS 1807, pages 139–155. Springer-Verlag, Berlin, 2000.
3. M. Bellare and P. Rogaway. The AuthA Protocol for Password-Based Authenticated Key Exchange. Contributions to IEEE P1363. March 2000. Available from `http://grouper.ieee.org/groups/1363/`.
4. M. Bellare and P. Rogaway. Random Oracles Are Practical: a Paradigm for Designing Efficient Protocols. In *Proc. of the 1st CCS*, pages 62–73. ACM Press, New York, 1993.
5. S. M. Bellovin and M. Merritt. Encrypted Key Exchange: Password-Based Protocols Secure against Dictionary Attacks. In *Proc. of the Symposium on Security and Privacy*, pages 72–84. IEEE, 1992.
6. S. M. Bellovin and M. Merritt. Augmented Encrypted Key Exchange: A Password-Based Protocol Secure against Dictionary Attacks and Password File Compromise. In *Proc. of the 1st CCS*, pages 244–250. ACM Press, New York, 1993.
7. V. Boyko, P. MacKenzie, and S. Patel. Provably Secure Password Authenticated Key Exchange Using Diffie-Hellman. In *Eurocrypt '00*, LNCS 1807, pages 156–171. Springer-Verlag, Berlin, 2000. Final version available at: `http://cm.bell-labs.com/who/philmac/research/`.
8. E. Bresson, O. Chevassut, and D. Pointcheval. Security Proofs for Efficient Password-Based Key Exchange. In *Proc. of the 10th CCS*. ACM Press, New York, 2003. The full version is available on the Cryptology ePrint Archive 2002/192.
9. E. Bresson, O. Chevassut, and D. Pointcheval. New Security Results on Encrypted Key Exchange. In *PKC '04*, LNCS. Springer-Verlag, Berlin, 2004. Full version available at: `http://www.di.ens.fr/users/pointche/`.
10. C. Dwork, A. Goldberg, and M. Naor. On Memory-Bound Functions for Fighting Spam. In *Crypto '03*, LNCS 2729, pages 426–444. Springer-Verlag, Berlin, 2003.
11. IEEE Standard 1363.2 Study Group. Password-Based Public-Key Cryptography. Available from `http://grouper.ieee.org/groups/1363/passwdPK`.
12. A. Juels and J. Brainard. Client Puzzles: A Cryptographic Defense Against Connection Depletion Attacks. In *Proc. of NDSS '99*, pages 151–165, 1999.
13. M. Luby and Ch. Rackoff. How to Construct Pseudorandom Permutations from Pseudorandom Functions. *SIAM Journal of Computing*, 17(2):373–386, 1988.
14. V. Shoup. OAEP Reconsidered. *Journal of Cryptology*, 15(4):223–249, September 2002.