One-time Verifier-based Encrypted Key Exchange

Michel Abdalla[1], Olivier Chevassut[2], and David Pointcheval[1]

[1] Dépt d'informatique, École normale supérieure, 75230 Paris Cedex 05, France
{Michel.Abdalla,David.Pointcheval}@ens.fr –
http://www.di.ens.fr/users/{mabdalla,pointche}.

[2] Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA,
OChevassut@lbl.gov – http://www.itg.lbl.gov/∼chevassu.

Abstract. "Grid" technology enables complex interactions among computational and data resources; however, to be deployed in production computing environments "Grid" needs to implement additional security mechanisms. Recent compromises of user and server machines at Grid sites have resulted in a need for secure password-authentication key-exchange technologies. AuthA is an example of such a technology considered for standardization by the IEEE P1363.2 working group. Unfortunately in its current form AuthA does not achieve the notion of forward-secrecy in a provably-secure way nor does it allow a Grid user to log into his account using an un-trusted computer. This paper addresses this void by first proving that AuthA indeed achieves this goal, and then by modifying it in such a way that it is secure against attacks using captured user passwords or server data.

1 Introduction

Motivation. Next generation distributed infrastructures integrate the ongoing work in Web Services (WS) with the state-of-the-art in distributed systems to enable seamless interaction among computational and data resources. "Grid" technology for example links computers, storage systems, and other devices through common interfaces and infrastructure to create powerful distributed computing capabilities [9, 11]. In this model of distributed computing, researchers and businesses not only plug into a global network of computer systems to access information but also to access distributed processing power. In parallel with the growth of Grid concepts and software in the scientific communities, commercial interests have been developing Web Services (WS) for the next generation business-to-business applications. Interest in both communities has grown to combine the techniques and concepts of Grid computing with the functionality of WS. This has led to the development of the Web Service Resource Framework (WSRF) specification and other elements of the Open Grid Services Architecture (OGSA) within several standard bodies such as the OASIS [19] and the Global Grid Forum (GGF) [13].

Security is one of the major requirements of Grid computing. Any Grid deployment must provide the security services of authentication, authorization, and secure session establishment. These services are provided by the Grid security infrastructure which was initially built upon the Transport Layer Security (TLS) protocol [10] and with the migration towards Web Services is now being built upon the WS-security primitives [9]. The current implementation of the Grid security infrastructure is based on public-key certificates. Recent security hacks of Grid sites due to the compromise of client and server machines, however, have led to a trend where many Grid sites are changing their security policies. The new policy prohibits long-term private keys from being stored on the Grid user's machines but requires that the keys are stored on servers in data centers where their integrity can be better protected. Grid users will authenticate to the data centers using a (one-time) human-memorable password and be issued short-lived certificates. Human-memorable passwords are short strings (e.g, 4 decimal digits) chosen from a relatively small dictionary so that they can be remembered easily.

The unique requirement of Grid provides security researchers with the opportunity to design and develop "provably-secure" cryptographic technologies that will play an essential role in securing next generation distributed infrastructures. The most immediate cryptographic need is certainly a "provably-secure" One-time Password-authentication and Key-eXchange technology (OPKeyX) for two-party [8].

Contributions. This paper is the third tier in the treatment of *Encrypted Key Exchange* (EKE), where the Diffie-Hellman key-exchange flows are encrypted using a password, in the direct model of Bellare-Pointcheval-Rogaway [1]. The first tier showed that under the computational Diffie-Hellman (CDH) assumption the AuthA password-authenticated key-exchange protocol is secure in both the random-oracle and ideal-cipher models [6]; the encryption primitive used is a password-keyed symmetric cipher. The second tier provided a very "elegant" and compact proof showing that under the CDH assumption the AuthA protocol is secure in the random-oracle model only [7]; the encryption primitive used is a mask generation function. In the present paper, we propose a slightly different variant of AuthA, where both flows are encrypted using separate mask generation functions, similarly to [18]. This *Two-Mask Encrypted Key Exchange* (EKE– both flows are encrypted) was not created for the sake of having one more variant, but simply because it allows us to provide the first complete proof of forward-secrecy for AuthA. The forward-secrecy of AuthA was indeed explicitly stated as an open problem in [2, 18]. Our result shows that under the Gap Diffie-Hellman assumption [20] this variant of AuthA is forward-secure in the random-oracle model. This is a significant achievement over other works which we hope will leverage our work to obtain tighter and more meaningful security measurements for the forward-secrecy of their EKE-like protocols.

We have furthermore augmented the *Two-Mask* protocol with two cryptographic mechanisms to reduce the risk of corruption of the server and the client.

Corruption of a server occurs when an attacker gains access to the server's local database of passwords. If client's passwords are stored directly in the database, then the attacker can immediately use any of these passwords to impersonate these clients. Fortunately, there is a means to prevent an attacker from doing just that: *verifier-based password-authentication.* Of course, this mechanism will not prevent an adversary from mounting (off-line) dictionary attacks but it will slow him or her down and thus give the server's administrator time to react appropriately and to inform its clients. Corruption of a client occurs when a client is using an un-trusted machine which happens frequently these days as hackers run password sniffers on the Internet. There is a means to prevent a client's password from being captured: *one-time password-based authentication.* Passwords sniffed by hackers are of no use since users' passwords change from one session to the other. The end result is a "provably-secure" One-time Password-authentication and Key-eXchange (OPKeyX) technology for Grid computing.

The remainder of the paper is organized as follows. We first present the related work. In Section 2, we define the formal security model which we use through the rest of the paper. In Section 3, we present the computational assumptions upon which the security of *Two-Mask* and, thus, our OPKeyX technology are based upon. In Section 4, we describe the *Two-Mask* protocol itself and prove that the latter is forward-secure via a reduction from the *Two-Mask* protocol to the Gap Diffie-Hellman problem. In Section 5, we augment the *Two-Mask* protocol to reduce the risk of stolen server databases and captured client passwords to construct a technology for OPKeyX.

Related Work. The seminal work in this area is the *Encrypted Key Exchange* (EKE) protocol proposed by Bellovin and Merritt in [3, 4]. EKE is a classical Diffie-Hellman key exchange wherein either or both flows are encrypted using the password as a common symmetric key. The 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 the hash of a password. Bellare et al. sketched a security proof for the flows at the core of the EKE protocol in [1], and specified a EKE-structure (called the AuthA protocol) in [2]. Boyko et al. proposed very similar EKE-structures (called the PAK suite) and proved them secure in Shoup's simulation model [5, 18]. The PPK protocol in the PAK suite is similar to our *Two-Mask Encrypted Key Exchange* protocol; however, arguments in favor of forward-secrecy under the computational Diffie-Hellman (CDH) assumption do not give many guarantees on its use in practice [18]. The KOY protocol [16] is also proved to be forward-secure but it is not efficient enough to be used in practice.

The PAK suite is in the process of being standardization by the IEEE P1363.2 Standard working group [15]. Server machines store images of the password under a one-way function instead of a plaintext password when the "augmented" versions of the PAK suite are used. "Augmented" EKE-like protocols indeed limit the damage due to the corruption of a server machine, but do not protect against attacks replaying captured users' passwords. On the other hand, One-Time Password (OTP) systems protect against the latter kind of attacks but provide neither privacy of transmitted data nor protection against active attacks such as session hijacking [14]. The present paper designs and develops a cryptographic protocol for one-time "augmented" password-authenticated key exchange.

2 Password-based Authenticated Key Exchange

In this section, we recall the security model of Bellare *et al.* [1] for password-based authenticated key exchange protocol.

2.1 Overview

A password-based authenticated key exchange protocol P is a protocol between two parties, a client $A \in \mathsf{client}$ and a server $S \in \mathsf{server}$. Each participant in a protocol may have several *instances*, called oracles, involved in distinct, possibly concurrent, executions of P. We let U^i denote the instance i of a participant U, which is either a client or a server.

Each client $A \in \mathsf{client}$ holds a password pw_A. Each server $S \in \mathsf{server}$ holds a vector $pw_S = \langle pw_S[A] \rangle_{A \in \mathsf{client}}$ with an entry for each client, where $pw_S[A]$ is the derived-password defined in [1]. In the symmetric model, $pw_S[C] = pw_C$, but they may be different in general, as in our verifier-based scheme. pw_C and pw_S are also referred to as the long-lived keys of client C and server S. Each password pw_A is considered to be a low-entropy string, drawn from the dictionary $\mathsf{Password}$ according to the distribution $\mathcal{PW}$. As in [7], we let $\mathcal{PW}(q)$ denote the probability to be in the most probable set of q passwords:

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

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

2.2 The Security Model

The interaction between an adversary $\mathcal{A}$ and the protocol participants occurs only via oracle queries, which model the adversary capabilities in a real attack (see literature for more details [1, 7].) The types of oracles available to the adversary are as follows:

- $\mathsf{Execute}(A^i, S^j)$: The output of this query consists of the messages exchanged during the honest execution of the protocol.
- $\mathsf{Reveal}(U^i)$: This query is only available to $\mathcal{A}$ if the attacked instance actually "holds" a session key and it releases the latter to $\mathcal{A}$.
- $\mathsf{Send}(U^i, m)$: The output of this query is the message that the instance U^i would generate upon receipt of message m. A query $\mathsf{Send}(A^i, \mathtt{Start})$ initializes the key exchange protocol, and thus the adversary receives the initial flow that client instance A^i would send to the server S.

2.3 Security Notions

In order to define a notion of security for the key exchange protocol, we consider a game in which the protocol P is executed in the presence of the adversary $\mathcal{A}$. In this game, we first draw a password pw from $\mathsf{Password}$ according to the distribution $\mathcal{PW}$, provide coin tosses and oracles to $\mathcal{A}$, and then run the adversary, letting it ask any number of queries as described above, in any order.

AKE Security. In order to model the privacy (semantic security) of the session key, we consider a new game $\mathbf{Game}^{\mathsf{ake}}(\mathcal{A}, P)$, in which an additional oracle is available to the adversary: the $\mathsf{Test}(U^i)$ oracle.

– $\mathsf{Test}(U^i)$: This query tries to capture the adversary's ability to tell apart a real session key from a random one. In order to answer it, we first flip a (private) coin b and then forward to the adversary either the session key sk held by U^i (i.e., the value that a query $\mathsf{Reveal}(U^i)$ would output) if $b = 1$ or a random key of the same size if $b = 0$.

The Test-oracle can be queried at most once by the adversary $\mathcal{A}$ and is only available to $\mathcal{A}$ if the attacked instance U^i is **Fresh** (which roughly means that the session key is not "obviously" known to the adversary). When playing this game, the goal of the adversary is to guess the hidden bit b involved in the Test-query, by outputting a guess b'. Let Succ denote the event in which the adversary is successful and correctly guesses the value of b. The **AKE advantage** of an adversary $\mathcal{A}$ is then defined as $\mathsf{Adv}_P^{\mathsf{ake}}(\mathcal{A}) = 2\Pr[\mathsf{Succ}] - 1$. 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. Note that the advantage of an adversary that simply guesses the bit b is 0 in the above definition due to the rescaling of the probabilities.

Forward-Secrecy. One additional security property to consider is that of forward secrecy. A key exchange protocol is said to be forward-secure if the security of a session key between two participants is preserved even if one of these participants is later compromised. In order to consider forward secrecy, one has to account for a new type of query, the $\mathsf{Corrupt}$-query, which models the compromise of a participant by the adversary. This query is defined as follows:

– $\mathsf{Corrupt}(U)$: This query returns to the adversary the long-lived key pw_U for participant U. As in [1], we assume the weak corruption model in which the internal states of all instances of that user are not returned to the adversary.

In order to define the success probability in the presence of this new type of query, one should extend the notion of freshness so as not to consider those cases in which the adversary can trivially break the security of the scheme. In this new setting, we say that a session key sk is **FS-Fresh** if all of the following hold: (1) the instance holding sk has accepted, (2) no $\mathsf{Corrupt}$-query has been asked since the beginning of the experiment; and (3) no Reveal-query has been asked to the instance holding sk or to its partner (defined according to the session identification). In other words, the adversary can only ask Test-queries to instances which had accepted before the Corrupt query is asked.

Let Succ denote the event in which the adversary successfully guesses the hidden bit b used by Test oracle. The **FS-AKE advantage** of an adversary $\mathcal{A}$ is then defined as $\mathsf{Adv}_P^{\mathsf{ake-fs}}(\mathcal{A}) = 2\Pr[\mathsf{Succ}] - 1$. The protocol P is said to be (t, ε)-**FS-AKE-secure** if $\mathcal{A}$'s advantage is smaller than ε for any adversary $\mathcal{A}$ running with time t.

Verifier-Based and One-Time-Password Protocols. In order to mitigate the amount of damage that can be caused by corruptions in the server and in the client, we consider two extensions to the standard notion of EKE protocols which we call *Verifier-Based* and *One-Time-Password* protocols.

In a Verifier-Based protocol, the goal is to keep the attacker capable of corrupting the server from obtaining the password for all the clients in the system. To achieve this goal, we need to adopt the asymmetric model in which the server no longer knows the password of a user, but only a function of it, which we call the verifier. In other words, only the client should know its password in a verifier-based protocol. Even though off-line dictionary attacks cannot be avoided in this case, the main idea of such protocols is to force an adversary who breaks into a server to have to perform an off-line dictionary attack for each password that it wants to crack based on its verifier. Therefore, the security of verifier-based protocols is directly related to the difficulty of recovering the original password from the verifier. In a One-Time-Password protocol, on the other hand, the goal is to limit the damage caused by an attacker who breaks into a client's machine or sniffs the password. This is achieved by forcing the user to use a different password in each session. That is, passwords are good for one session only and cannot be reused.

3 Algorithmic Assumptions

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 $\mathbb{G}$.

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:

$$\mathsf{Succ}_{g,\mathbb{G}}^{\mathsf{cdh}}(\Delta) = \Pr[\Delta(g^x, g^y) = g^{xy}] \geq \varepsilon.$$

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.

A (t, n, ε)-$\mathsf{GDH}_{g,\mathbb{G}}$ attacker is a (t, ε)-$\mathsf{CDH}_{g,\mathbb{G}}$ attacker, with access to an additional oracle: a DDH-oracle, which on any input (g^x, g^y, g^z) answers whether

$$
\begin{array}{rcl}
\underline{\textit{Client}} & & \underline{\textit{Server}} \\[6pt]
\multicolumn{3}{c}{pw \in \mathsf{Password}, \mathsf{PW}^{\mathsf{as}} = \mathcal{G}(A\|S\|pw), \mathsf{PW}^{\mathsf{sa}} = \mathcal{G}(S\|A\|pw) \in \mathbb{G}} \\[6pt]
\mathsf{accept} \leftarrow \mathsf{false} & & \mathsf{accept} \leftarrow \mathsf{false} \\[6pt]
x \stackrel{R}{\leftarrow} \mathbb{Z}_q & & y \stackrel{R}{\leftarrow} \mathbb{Z}_q \\
X \leftarrow g^x & & Y \leftarrow g^y \\
X^\star \leftarrow X \times \mathsf{PW}^{\mathsf{as}} & \xrightarrow{A, X^\star} & X \leftarrow X^\star / \mathsf{PW}^{\mathsf{as}} \\
Y \leftarrow Y^\star / \mathsf{PW}^{\mathsf{sa}} & \xleftarrow{S, Y^\star} & Y^\star \leftarrow Y \times \mathsf{PW}^{\mathsf{sa}} \\[6pt]
sk = \mathcal{H}(A\|S\|X^\star\|Y^\star\|pw\|Y^x) & & sk = \mathcal{H}(A\|S\|X^\star\|Y^\star\|pw\|X^y) \\[6pt]
\mathsf{accept} \leftarrow \mathsf{true} & & \mathsf{accept} \leftarrow \mathsf{true}
\end{array}
$$

Fig. 1. An execution of the EKE protocol.

$z = xy \bmod q$. Its number of queries is limited to n. As usual, we denote by $\mathsf{Succ}^{\mathsf{gdh}}_{g,\mathbb{G}}(t)$ the maximal success probability over every adversaries running within time t. The GDH-Assumption states that $\mathsf{Succ}^{\mathsf{gdh}}_{g,\mathbb{G}}(t) \leq \varepsilon$ for any t/ε not too large.

4 The EKE Protocol: Encrypted Key Exchange

4.1 Description of the Scheme

A hash function from $\{0,1\}^\star$ to $\{0,1\}^\ell$ is denoted $\mathcal{H}$. While $\mathcal{G}$ denotes a full-domain hash function from $\{0,1\}^\star$ into $\mathbb{G}$. As illustrated on Figure 1 (with an honest execution of the EKE protocol), the protocol runs between two parties A and S, and the session-key space **SK** associated to this protocol is $\{0,1\}^\ell$ equipped with a uniform distribution. It works as follows. The client chooses at random a private random exponent x and computes its Diffie-Hellman public value g^x. The client encrypts the latter value using a password-based mask, as the product of a Diffie-Hellman value with a full-domain hash of the password, and sends it to the server. The server in turn chooses at random a private random exponent y and computes its Diffie-Hellman public value g^y which it encrypts using another password-based mask[3]. The client (resp. server) then decrypts the flow it has received and computes the session key.

4.2 Security Result

In this section, we assert that under the intractability of the Diffie-Hellman problem, the EKE protocol, securely distributes session keys: the key is semantically

[3] this differs from the classical EKE protocol, which uses a common mask [7]. But this helps to improve the security result.

secure. The proof, which is an improvement of [7], can be found in the full version of this paper.

Theorem 1 (AKE Security). *Let us consider the above* EKE *protocol, over a group of prime order q, where* Password *is a dictionary equipped with the distribution $\mathcal{PW}$. Let $\mathcal{A}$ be an adversary against the AKE security within a time bound t, with less than q_s active interactions with the parties (*Send*-queries) and q_p passive eavesdroppings (*Execute*-queries), and, asking q_g and q_h hash queries to $\mathcal{G}$ and $\mathcal{H}$ respectively. Then we have*

$$\mathsf{Adv}_{\mathsf{eke}}^{\mathsf{ake}}(\mathcal{A}) \leq 2 \times \mathcal{PW}(q_s) + 4q_h^2 \times \mathsf{Succ}_{g,\mathbb{G}}^{\mathsf{cdh}}(t + 5\tau_e) + \frac{(q_p + q_s)^2 + 3(q_g + q_h)^2}{2q},$$

where τ_e denotes the computational time for an exponentiation in $\mathbb{G}$.

Let us now enhance the result to cover forward-secrecy. The proof will be different from previous proofs for EKE-like protocols since the simulation still must be independent of any password (so that we can say that the adversary has a minute of chance to guess the correct one), while after a corruption the adversary will be able to check the consistency. To reach this aim, we will need to rely on a stronger assumption: the Gap Diffie-Hellman problem. The Decisional Diffie-Hellman oracle will be used to identify the public random oracle $\mathcal{H}$ to the private one $\mathcal{H}'$ when the input is a valid Diffie-Hellman value.

Theorem 2 (FS-AKE Security). *Let us consider the above* EKE *protocol, over a group of prime order q, where* Password *is a dictionary equipped with the distribution $\mathcal{PW}$. Let $\mathcal{A}$ be an adversary against the FS-AKE security within a time bound t, with less than q_s active interactions with the parties (*Send*-queries) and q_p passive eavesdroppings (*Execute*-queries), and, asking q_g and q_h hash queries to $\mathcal{G}$ and $\mathcal{H}$ respectively. Then we have*

$$\mathsf{Adv}_{\mathsf{eke}}^{\mathsf{ake-fs}}(\mathcal{A}) \leq 2 \times \mathcal{PW}(q_s) + 4 \times \mathsf{Succ}_{g,\mathbb{G}}^{\mathsf{gdh}}(q_h, t + 5\tau_e) + \frac{(q_p + q_s)^2 + 3(q_g + q_h)^2}{2q},$$

where τ_e denotes the computational time for an exponentiation in $\mathbb{G}$.

Proof. As usual, we incrementally define a sequence of games starting at the real game $\mathbf{G}_0$ and ending up at $\mathbf{G}_5$. We are interested in the event S, which occurs if the adversary correctly guesses the bit b involved in the Test-query. Let us remember that in this attack game, the adversary is provided with the Corrupt-query.

GAME $\mathbf{G}_0$: This is the real protocol, in the random-oracle model. By definition of event S_0, which means that the adversary correctly guesses the bit b involved in the Test-query, we have

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

GAME $\mathbf{G}_1$: In this game, we simulate the hash oracles ($\mathcal{G}$ and $\mathcal{H}$, but also an additional hash function $\mathcal{H}' : \{0,1\}^\star \to \{0,1\}^\ell$ that will appear in the Game

$\mathcal{G}$ and $\mathcal{H}$ oracles	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.
	For a hash-query $\mathcal{H}(q)$ such that a record (q, r) appears in $\Lambda_{\mathcal{H}}$, the answer is r. Otherwise, q is parsed as $(A\|S\|X^\star\|Y^\star\|pw\|K)$, one first asks for $\mathcal{G}(A\|S\|pw)$ and $\mathcal{G}(S\|A\|pw)$, using the above simulation, then the answer r is defined according to the following rule: ►**Rule** $\mathcal{H}^{(1)}$ Choose a random element $r \in \{0,1\}^\ell$. One adds the record (q, r) to $\Lambda_{\mathcal{H}}$.
	For a hash-query $\mathcal{H}'(q)$, such that a record (q, r) appears in $\Lambda_{\mathcal{H}'}$, the answer is r. Otherwise, one chooses a random element $r \in \{0,1\}^\ell$, answers with it, and adds the record (q, r) to $\Lambda_{\mathcal{H}'}$.

Fig. 2. Simulation of the EKE protocol (random oracles)

$\mathbf{G}_3$) as usual by maintaining hash lists $\Lambda_{\mathcal{G}}$, $\Lambda_{\mathcal{H}}$ and $\Lambda_{\mathcal{H}'}$ (see Figure 2). Except that we query $\mathcal{G}(A\|S\|pw)$ and $\mathcal{G}(S\|A\|pw)$ as soon as A, S and pw appear in a $\mathcal{H}$-query. This just increases the number of $\mathcal{G}$ queries. We also simulate all the instances, as the real players would do, for the Send-queries and for the Execute, Reveal, Test and Corrupt-queries (see Figure 3).

From this simulation, we easily see that the game is perfectly indistinguishable from the real attack.

Game $\mathbf{G}_2$: First, we cancel games in which some collisions appear:

- collisions on the transcripts $((A, X^\star), (S, Y^\star))$;
- collisions on the output of $\mathcal{G}$.

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

Game $\mathbf{G}_3$: In this game, we do not compute the session key sk using the oracle $\mathcal{H}$, but using the private oracle $\mathcal{H}'$ so that the value sk is completely independent not only from $\mathcal{H}$, but also from pw and thus from both K_A and K_S. We reach this aim by using the following rule:

►**Rule A3/S3**$^{(3)}$

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

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

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 $X^\star = X \times \mathsf{PW}^{\mathsf{as}}$.
  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 $\mathsf{Send}(A^i, (S, Y^\star))$ is processed by computing the session key. We apply the following rules:
  ▶**Rule A2**$^{(1)}$
  Compute $Y = Y^\star / \mathsf{PW}^{\mathsf{sa}}$ and $K_A = Y^\theta$.
  ▶**Rule A3**$^{(1)}$
  Compute the session key $sk_A = \mathcal{H}(A \| S \| X^\star \| Y^\star \| pw \| K_A)$.
  Finally the instance accepts.

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 $Y = g^\varphi$ and $Y^\star = Y \times \mathsf{PW}^{\mathsf{sa}}$.
  Then the query is answered with $(S, Y^\star)$, and the instance applies the following rules.
  ▶**Rule S2**$^{(1)}$
  Compute $X = X^\star / \mathsf{PW}^{\mathsf{as}}$ and $K_S = X^\varphi$.
  ▶**Rule S3**$^{(1)}$
  Compute the session key $sk_S = \mathcal{H}(A \| S \| X^\star \| Y^\star \| pw \| K_S)$.
  Finally, the instance accepts.

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 $(S, Y^\star) \leftarrow \mathsf{Send}(S^j, (A, X^\star))$, and outputting the transcript $((A, X^\star), (S, Y^\star))$.

A $\mathsf{Reveal}(U)$-query returns the session key (sk_A or sk_S) computed by the instance I (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 $\{0,1\}^\ell$.

A $\mathsf{Corrupt}(U)$-query returns password pw of the user U.

Fig. 3. Simulation of the EKE protocol (Send, Reveal, Execute, Test and Corrupt queries)

►**Rule A2/S2**$^{(3)}$

Do nothing.

The games $\mathbf{G}_3$ and $\mathbf{G}_2$ are indistinguishable unless $\mathcal{A}$ queries the hash function $\mathcal{H}$ on either $A\|S\|X^\star\|Y^\star\|pw\|K_A$ or $A\|S\|X^\star\|Y^\star\|pw\|K_S$, for some execution transcript $((A, X^\star), (S, Y^\star))$. We hope to prove that for all the transcripts of accepted sessions, the probability of such an event is negligible. However, there is no hope for proving it about sessions accepted *after* the corruption of the password, since the adversary may know the x and thus K_A (or y and K_S). One should note that sessions accepted *after* the corruption may have been started *before*. There is no way in our simulation to anticipate different answers for the Send-queries according to that. Therefore, we have to make answers from $\mathcal{H}$ and $\mathcal{H}'$ (when they correspond to the same query, which can be checked with the DDH-oracle) to be the same for sessions accepted after the corruption of the password:

►**Rule** $\mathcal{H}^{(3)}$

- Before the corruption, randomly choose $r \in \{0,1\}^\ell$.
- After the corruption, knowing the correct password, if
  - pw is the correct password;
  - $A, S, X^\star, Y^\star$ corresponds to the session ID of a session accepted after the corruption;
  - $K = \mathsf{CDH}_{g,\mathbb{G}}(X^\star/\mathsf{PW}^{\mathsf{as}}, Y^\star/\mathsf{PW}^{\mathsf{sa}})$ (checked using the DDH-oracle);

  then r is set to $\mathcal{H}'(A\|S\|X^\star\|Y^\star)$.
  Else, choose a random element $r \in \{0,1\}^\ell$.

This new rule for the simulation of $\mathcal{H}$ just replaces some random values by other random values. The games $\mathbf{G}_3$ and $\mathbf{G}_2$ are now indistinguishable unless $\mathcal{A}$ queried the hash function $\mathcal{H}$ on either $A\|S\|X^\star\|Y^\star\|pw\|K_A$ or $A\|S\|X^\star\|Y^\star\|pw\|K_S$, for some accepted-session transcript $((A, X^\star), (S, Y^\star))$, *before* corrupting the password: event AskHbC. This means that, for *some transcript* $((A, X^\star), (S, Y^\star))$, the tuple $(A, S, X^\star, Y^\star, pw, \mathsf{CDH}_{g,\mathbb{G}}(X^\star/\mathsf{PW}^{\mathsf{as}}, Y^\star/\mathsf{PW}^{\mathsf{sa}}))$ lies in the list $\Lambda_{\mathcal{H}}$.

On the other hand, the session key (associated to a session accepted *before* the corruption) is computed with a random oracle that is private to the simulator, then one can remark that it cannot be distinguished by the adversary unless the same transcript $((A, X^\star), (S, Y^\star))$ appeared in another session, for which a Reveal-query has been asked (which event has been excluded in the previous game). The adversary correctly guesses the bit b involved in the Test-query (event S_3) only by chance: $\Pr[\mathsf{S}_3] = 1/2$.

Actually, one does not need the Diffie-Hellman values K_A or K_S for computing sk, but the password: we can formally simplify again some rules but thus without modifying anything w.r.t. the probabilities:

►**Rule A1**$^{(3)}$

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

▶**Rule S1**$^{(3)}$

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

<u>Game $\mathbf{G}_4$:</u> In order to evaluate the probability of event AskHbC, let us modify the simulation of the oracle $\mathcal{G}$, with two random elements $P, Q \in \mathbb{G}\backslash\{1\}$ (which are thus generators of $\mathbb{G}$, since the latter has a prime order q). The simulation introduces values in the third component of the elements of $\Lambda_{\mathcal{G}}$, but does not use it. It would let the probabilities unchanged, but we exclude the cases $\mathsf{PW}^{\mathsf{as}} = 1$ or $\mathsf{PW}^{\mathsf{sa}} = 1$:

▶**Rule** $\mathcal{G}^{(4)}$

- If $q =$ "$A\|S\|\star$", randomly choose $k \in \mathbb{Z}_q^\star$, and compute $r = P^{-k}$;
- If $q =$ "$S\|A\|\star$", randomly choose $k \in \mathbb{Z}_q^\star$, and compute $r = Q^{-k}$;
- Else, choose a random element $r \in \mathbb{G}$, and set $k = \perp$.

The record (q, r, k) is added to $\Lambda_{\mathcal{G}}$.

Since we just exclude $k = 0$, we have:

$$|\Pr[\mathsf{AskHbC}_4] - \Pr[\mathsf{AskHbC}_3]| \leq \frac{q_g + q_h}{q}.$$

<u>Game $\mathbf{G}_5$:</u> It is now possible to evaluate the probability of the event AskHbC. Indeed, one can remark that the password is never used during the simulation, before the corruption. It thus does not need to be chosen in advance, but at the time of the corruption (or at the very end only). At that time, one can check whether the event AskHbC happened or not. To make this evaluation easier, we cancel the games wherein for some pair $(X^\star, Y^\star) \in \mathbb{G}^2$, involved in a communication, there are two passwords pw such that the tuple $(A, S, X^\star, Y^\star, pw, \mathsf{CDH}_{g,\mathbb{G}}(X^\star/\mathsf{PW}^{\mathsf{as}}, Y^\star/\mathsf{PW}^{\mathsf{sa}}))$ is in $\Lambda_{\mathcal{H}}$ (which event is denoted CollH_5). Hopefully, event CollH_5 can be upper-bounded, granted the following Lemma:

Lemma 1. *For any pair $(X^\star, Y^\star)$ involved in a communication, there is at most one password pw such that $(A, S, X^\star, Y^\star, pw, \mathsf{CDH}_{g,\mathbb{G}}(X^\star/\mathsf{PW}^{\mathsf{as}}, Y^\star/\mathsf{PW}^{\mathsf{sa}}))$ is in $\Lambda_{\mathcal{H}}$, unless one can solve the Diffie-Hellman problem:*

$$\Pr[\mathsf{CollH}_5] \leq \mathsf{Succ}^{\mathsf{gdh}}_{g,\mathbb{G}}(q_h, t + 5\tau_e).$$

Proof. Assume there exist $(X^\star = g^x, Y^\star = g^y) \in \mathbb{G}^2$ involved in a communication, $\mathsf{PW}_0^{\mathsf{as}} = P^{-k_0} \neq 1$, $\mathsf{PW}_0^{\mathsf{sa}} = Q^{-k'_0} \neq 1$, and $\mathsf{PW}_1^{\mathsf{as}} = P^{-k_1} \neq 1$, $\mathsf{PW}_1^{\mathsf{sa}} = Q^{-k'_1} \neq 1$ such that the two following tuples (for $i = 0, 1$) are in $\Lambda_{\mathcal{H}}$:

$$(A, S, X^\star, Y^\star, pw_i, Z_i = \mathsf{CDH}_{g,\mathbb{G}}(X^\star/\mathsf{PW}_i^{\mathsf{as}}, Y^\star/\mathsf{PW}_i^{\mathsf{sa}})).$$

Then, $Z_i = \mathsf{CDH}_{g,\mathbb{G}}(X^\star \times P^{k_i}, Y^\star \times Q^{k'_i})$. Since $(X^\star, Y^\star) \in \mathbb{G}^2$ has been involved in a communication (either from Send-queries or an $\mathsf{Execute}$-query), one of $X^\star =$

g^x or $Y^\star = g^y$, has been simulated: at least one of x or y is known. Without loss of generality, we can assume we know x:

$$\begin{aligned} Z_i &= (Y^\star \times Q^{k'_i})^x \times \mathsf{CDH}_{g,\mathbb{G}}(Y^\star, P)^{k_i} \times \mathsf{CDH}_{g,\mathbb{G}}(P,Q)^{k_i k'_i} \\ Z_1^{k_0}/Z_0^{k_1} &= \left(Y^{\star\, k_0-k_1} \times \mathsf{PW}_0^{\mathsf{sa}\, k_1}/\mathsf{PW}_1^{\mathsf{sa}\, k_0}\right)^x \times \mathsf{CDH}_{g,\mathbb{G}}(P,Q)^{k_0k_1(k'_1-k'_0)} \\ \mathsf{CDH}_{g,\mathbb{G}}(P,Q) &= \left(((\mathsf{PW}_1^{\mathsf{sa}}/Y^\star)^x Z_1)^{k_0} \,/\, ((\mathsf{PW}_0^{\mathsf{sa}}/Y^\star)^x Z_0)^{k_1}\right)^u, \end{aligned}$$

where u is the inverse of $k_0k_1(k'_1 - k'_0)$ in $\mathbb{Z}_q$. The latter exists since $\mathsf{PW}_0^{\mathsf{as}}$, $\mathsf{PW}_0^{\mathsf{sa}}$, $\mathsf{PW}_1^{\mathsf{as}}$, $\mathsf{PW}_1^{\mathsf{sa}} \neq 1$, and they are all distinct from each other (we have excluded collisions for $\mathcal{G}$). Since we have access to a DDH-oracle, one can find the two useful $\mathcal{H}$-queries. □

For a more convenient analysis, we can split the event AskHbC in two disjoint sub-cases:

1. AskHbC-Passive, where the transcript $((A, X^\star), (S, Y^\star))$ involved in the crucial $\mathcal{H}$-query comes as an answer from an Execute-query;
2. AskHbC-Active, the other cases.

About the active case (the event $\mathsf{AskHbC\text{-}Active}_5$), the above Lemma 1 applied to games where the event CollH_5 did not happen states that for each pair $(X^\star, Y^\star)$ involved in an active transcript, there is at most one pw such that the corresponding tuple is in $\Lambda_{\mathcal{H}}$:

$$\Pr[\mathsf{AskHbC\text{-}Active}_5] \leq \mathcal{PW}(q_s).$$

Moreover, in the particular case of passive transcripts, one can state a stronger result:

Lemma 2. *For any pair $(X^\star, Y^\star) \in \mathbb{G}^2$, involved in a passive transcript, there is no password pw such that $(A, S, X^\star, Y^\star, pw, \mathsf{CDH}_{g,\mathbb{G}}(X^\star/\mathsf{PW}^{\mathsf{as}}, Y^\star/\mathsf{PW}^{\mathsf{sa}}))$ is in $\Lambda_{\mathcal{H}}$, unless one can solve the Diffie-Hellman problem:*

$$\Pr[\mathsf{AskHbC\text{-}Passive}_5] \leq \mathsf{Succ}^{\mathsf{gdh}}_{g,\mathbb{G}}(q_h, t + 4\tau_e).$$

Proof. Assume there exist $(X^\star = g^x, Y^\star = g^y) \in \mathbb{G}^2$ involved in a passive transcript, and values $\mathsf{PW}^{\mathsf{as}} = P^{-k} \neq 1$, $\mathsf{PW}^{\mathsf{sa}} = Q^{-k'} \neq 1$ such that the tuple

$$(A, S, X^\star, Y^\star, pw, Z = \mathsf{CDH}_{g,\mathbb{G}}(X^\star/\mathsf{PW}^{\mathsf{as}}, Y^\star/\mathsf{PW}^{\mathsf{sa}}))$$

is in $\Lambda_{\mathcal{H}}$. Then, as above (but with x and y known),

$$\mathsf{CDH}_{g,\mathbb{G}}(P,Q) = \left(Z \times \mathsf{PW}^{\mathsf{sa}\,x} \times \mathsf{PW}^{\mathsf{as}\,y}/g^{xy}\right)^u,$$

where u is the inverse of kk' in $\mathbb{Z}_q$. By using the DDH-oracle, one easily gets the crucial $\mathcal{H}$-query. □

As a conclusion,

$$\Pr[\mathsf{AskHbC}_5] \leq \mathsf{Succ}^{\mathsf{gdh}}_{g,\mathbb{G}}(q_h, t + 4\tau_e) + \mathcal{PW}(q_s).$$

Combining all the above equations, one gets

$$\mathsf{Adv}^{\mathsf{ake-fs}}_{\mathsf{eke}}(\mathcal{A}) \leq 2 \times \begin{pmatrix} \mathcal{PW}(q_s) + \mathsf{Succ}^{\mathsf{gdh}}_{g,\mathbb{G}}(q_h, t + 4\tau_e) + \mathsf{Succ}^{\mathsf{gdh}}_{g,\mathbb{G}}(q_h, t + 5\tau_e) \\ + \dfrac{q_g + q_h}{q} + \dfrac{(q_g + q_h)^2}{2q} + \dfrac{(q_p + q_s)^2}{2q} \end{pmatrix}.$$

□

5 The OPKeyX Protocol

The basic EKE protocol withstands password corruption, by providing forward-secrecy. But this just protects the secrecy of session keys established before the corruption. Nothing is guaranteed for future sessions. We can even show that one easily breaks the semantic security of their session keys, by simply impersonating one of the parties with the knowledge of the password.

In the above protocol, the password can be extracted from both machines: the server and the client. And moreover, the server stores many passwords (since its is aimed at establishing sessions with many clients), then the corruption of the server does not just leak one password, but a huge number of them. This would be quite useful to be able to reduce the damages of such a corruption. We propose below two different ways to achieve this task.

5.1 Stealing the Server Database

In a verifier-based protocol, the client owns a password, but the server just knows a verifier of the latter (which is actually a hash value, or the image by a one-way function), not the password itself. Hence, the corruption of the server just reveals this verifier. Of course, an off-line dictionary attack thereafter leads to the password. Such an exhaustive search cannot be prevented but should be the most efficient one: by including salts (sent back to the client by the server in the first flow) would reduce even more the impact of the corruption, since a specific dictionary attack should be performed towards each specific user, and could not be generic.

A verifier-based enhancement of EKE is proposed on Figure 4. It is basically the previous EKE scheme using first the verifier as common password. Then, the client furthermore proves his knowledge of the password which matches the password-verifier relation. In our proposal, the relation is the pairs (x, g^x), and thus the proof is a Schnorr-like proof of knowledge of a discrete logarithm [21], with a multi-collision resistant function f [12]. To prevent dictionary attacks, we introduce the Diffie-Hellman secret in the hash input to get the challenge e, so that the latter can be computed by the two parties only: it is semantically secure for external adversaries for exactly the same reasons the session key is. Because of

$$
\begin{array}{rcl}
\underline{\textit{Client}} & & \underline{\textit{Server}} \\
\mathsf{pw} \in \mathbb{Z}_q & & pw = g^{\mathsf{pw}} \\
\multicolumn{3}{c}{\mathsf{PW}^{\mathsf{as}} = \mathcal{G}(A\|S\|pw), \mathsf{PW}^{\mathsf{sa}} = \mathcal{G}(S\|A\|pw) \in \mathbb{G}} \\
\mathsf{accept} \leftarrow \mathsf{false} & & \mathsf{accept} \leftarrow \mathsf{false} \\
x \stackrel{R}{\leftarrow} \mathbb{Z}_q, X \leftarrow g^x & & y \stackrel{R}{\leftarrow} \mathbb{Z}_q, Y \leftarrow g^y \\
Y \leftarrow Y^\star/\mathsf{PW}^{\mathsf{sa}} & \xleftarrow{\quad S, Y^\star \quad} & Y^\star \leftarrow Y \times \mathsf{PW}^{\mathsf{sa}} \\
X^\star \leftarrow X \times \mathsf{PW}^{\mathsf{as}} & & \\
r \stackrel{R}{\leftarrow} \mathbb{Z}_q, R \leftarrow g^r, \rho = f(R) & & \\
e = \mathcal{H}_1(A\|S\|X^\star\|Y^\star\|\rho\|pw\|Y^x) & & \\
s = r - e \cdot \mathsf{pw} \bmod q & \xrightarrow{A, X^\star, \rho, s} & X \leftarrow X^\star/\mathsf{PW}^{\mathsf{as}} \\
& & e = \mathcal{H}_1(A\|S\|X^\star\|Y^\star\|\rho\|pw\|Y^x) \\
& & \text{if } \rho = f(g^s pw^e), \\
& & \quad \text{then } \mathsf{accept} \leftarrow \mathsf{true} \\
sk = \mathcal{H}(A\|S\|X^\star\|Y^\star\|\rho\|pw\|Y^x) & & sk = \mathcal{H}(A\|S\|X^\star\|Y^\star\|\rho\|pw\|X^y) \\
\mathsf{accept} \leftarrow \mathsf{true} & &
\end{array}
$$

Fig. 4. An execution of the VB-EKE protocol.

this semantic security, dictionary attacks are still prevented, since the additional proof of knowledge does not reveal any information: the verification relation is actually secret, because of the secrecy of e. As a consequence, the private property of e makes that the proof does not leak any information about both the password and the verifier to external parties. The zero-knowledge property of this proof makes that even the server does not learn any additional information about the password.

To improve efficiency, we also swapped the flows, so that the protocol remains a 2-pass one. Indeed, the client has to be the last, since it has to send its proof of knowledge of the password. By swapping the two flows of the basic EKE protocol, the latter proof of knowledge can be concatenated to the last flow, which does not increase the communication cost.

From a more practical point of view, this inversion better suits the Transport Layer Security (TLS) protocol [22]. The flows of the VB-EKE protocol thus have to comply with the key-exchange phase, which happens right after the *hello* flows (the first is from the client to the server, then the second goes back from the server to the client) and precedes the *finish* phase (the first *finish* message is again from the client to the server). In short, the first message of the VB-EKE protocol would simply map to the *ServerKeyExchange* flows while the second message to the *ClientKeyExchange* message.

Client		*Server*
$\mathsf{pw} \in \mathsf{Password}, n, pw_n = f^n(\mathsf{pw})$		$n, pw = f(pw_n)$
$\mathsf{PW}^{\mathsf{as}} = \mathcal{G}(A\|S\|pw), \mathsf{PW}^{\mathsf{sa}} = \mathcal{G}(S\|A\|pw) \in \mathbb{G}$		
$\mathsf{accept} \leftarrow \mathsf{false}$		$\mathsf{accept} \leftarrow \mathsf{false}$
$x \overset{R}{\leftarrow} \mathbb{Z}_q, X \leftarrow g^x$		$y \overset{R}{\leftarrow} \mathbb{Z}_q, Y \leftarrow g^y$
n correct?	$\xleftarrow{S, Y^\star, n}$	$Y^\star \leftarrow Y \times \mathsf{PW}^{\mathsf{sa}}$
$Y \leftarrow Y^\star / \mathsf{PW}^{\mathsf{sa}}$		
$X^\star \leftarrow X \times \mathsf{PW}^{\mathsf{as}}$		
$s = \mathcal{H}_1(A\|S\|X^\star\|Y^\star\|pw\|Y^x)$		
$c = E_s(pw_n)$	$\xrightarrow{A, X^\star, c}$	$X \leftarrow X^\star / \mathsf{PW}^{\mathsf{as}}$
		$s = \mathcal{H}_1(A\|S\|X^\star\|Y^\star\|pw\|Y^x)$
		$p = D_s(c)$, if $pw = f(p)$,
		then $pw \leftarrow p, n \leftarrow n-1$,
		$\mathsf{accept} \leftarrow \mathsf{true}$
$sk = \mathcal{H}(A\|S\|X^\star\|Y^\star\|pw\|Y^x)$		$sk = \mathcal{H}(A\|S\|X^\star\|Y^\star\|pw\|X^y)$
$\mathsf{accept} \leftarrow \mathsf{true}$		

Fig. 5. An execution of the OPKeyX protocol.

5.2 Capturing the Client Password

The above modified scheme does not really increase the communication cost, since additional data can be concatenated to existing flows. But both parties have more computation to do, and namely a few exponentiations. The password-verifier relation can be more efficient, using any one-way function. However, for such a general function, a zero-knowledge proof of knowledge of the password may not be easy to perform. But the zero-knowledge property is not required, if we move to the one-time password scenario: $f(pw)$ is first used as a common password, then the client eventually reveals the password, which will thereafter be the future common data (or verifier) if $pw = f^n(\mathsf{seed})$ [17]. The computation of $f^n(pw)$ is performed by a one-time password generator which derives successive passwords from a seed. Since one-time password generators do not require reader devices they are much more adapted for the Grid environment than contact tokens (e.g, smart-card, USB tokens). This discussion leads to the One-time Password-enhanced version of VB-EKE which is proposed on Figure 5. The communication of the password has indeed to be sent in a private way, since it will become the future common data, hence the use of an ephemeral session key, which is trivially semantically secure (due to Theorem 2).

6 Conclusion

This paper provides strong security arguments to support the EKE-like protocols being standardized by the IEEE P1363.2 Standard working group (namely the PPK series). We have reached this aim by slightly modifying the original AuthA protocol (the two encryption primitives are instantiated using separate mask generation functions but derived from a unique shared password) to be able to achieve the security notion of forward-secrecy in a provably-secure way. Our result is a slight departure from previously known results on EKE-like structures since the security of AuthA is now based on the Gap Diffie-Hellman problem. Moreover, we have extended AuthA into a One-time Password-authentication and Key eXchange (OPKeyX) technology which allows a user to securely log into his account using a remote un-trusted computer and limits the damages of corruption of the server.

Acknowledgments

The authors would like to thanks Frank Siebenlist for invaluable discussions related to Grid computing. The first and third authors have been supported in part by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT. 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-56212. Disclaimer available at `http://www-library.lbl.gov/disclaimer`.

References

1. 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.
2. M. Bellare and P. Rogaway. The AuthA Protocol for Password-Based Authenticated Key Exchange. Contributions to IEEE P1363. March 2000.
3. 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.
4. 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.
5. 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.
6. E. Bresson, O. Chevassut, and D. Pointcheval. Security proofs for an efficient password-based key exchange. In *Proc. of the 10th CCS*, pages 241–250. ACM Press, New York, 2003.
7. E. Bresson, O. Chevassut, and D. Pointcheval. New Security Results on Encrypted Key Exchange. In *PKC '04*, LNCS, pages 145–159. Springer-Verlag, Berlin, 2004.

8. L. Fang, S. Meder, O. Chevassut, and F. Siebenlist. Secure Password-based Authenticated key Exchange for Web Services In *Proc. of the ACM Workshop on Secure Web Services*, 2004.
9. I. Foster and C. Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2004.
10. I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. Security Architecture for Computational Grids. In *Proc. of the 5th CCS*, pages 83–92. ACM Press, New York, 1998.
11. I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International J. Supercomputer Applications*, 15(3), 2001.
12. M. Girault and J. Stern. On the Length of Cryptographic Hash-Values used in Identification Schemes. In *Crypto '94*, LNCS 839, pages 202–215. Springer-Verlag, Berlin, 1994.
13. The Global Grid Forum (GGF). `http://www.ggf.org`.
14. N. Haller, C. Metz, P. Nesser, and M. Straw. *RFC 2289: A One-Time Password System*. Internet Activities Board, February 1998.
15. IEEE Standard 1363.2 Study Group. Password-Based Public-Key Cryptography. `http://grouper.ieee.org/groups/1363/passwdPK`.
16. J. Katz, R. Ostrovsky, and M. Yung. Forward secrecy in password-only key exchange protocols. In *SCN'02*, LNCS 2576, pages 29–44. Springer-Verlag, Berlin, 2002.
17. L. Lamport. Password Authentication with Insecure Communication. *Communications of the ACM 24*, 11:770–771, November 1981.
18. P. D. MacKenzie. The PAK suite: Protocols for password-authenticated key exchange. Technical Report 2002-46, DIMACS, 2002.
19. The Oasis standard body. `http://www.oasis-open.org`.
20. T. Okamoto and D. Pointcheval. The Gap-Problems: a New Class of Problems for the Security of Cryptographic Schemes. In *PKC '01*, LNCS 1992. Springer-Verlag, Berlin, 2001.
21. C. P. Schnorr. Efficient Signature Generation by Smart Cards. *Journal of Cryptology*, 4(3):161–174, 1991.
22. M. Steiner, P. Buhler, T. Eirich, and M. Waidner. Secure Password-Based Cipher Suite for TLS. *ACM Transactions on Information and System Security (TISSEC)*, 4(2):134–157, 2001.