This is the full version of the extended abstract which appears in
Proceedings of the 10th ACM Conference on Computer and Communications Security, pages ???–???
(October 27 – 30, 2003, Washington, DC, USA.)

Security Proofs for an Efficient Password-Based Key Exchange

E. Bresson[1], O. Chevassut[2], and D. Pointcheval[3]

[1] Département Cryptologie, CELAR, 35174 Bruz Cedex, France
Emmanuel.Bresson@m4x.org.
[2] Ernest Orlando Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA
OChevassut@lbl.gov.
[3] CNRS–École normale supérieure, 75230 Paris Cedex 05, France
David.Pointcheval@ens.fr.

Abstract. Password-based key exchange schemes are designed to provide entities communicating over a public network, and sharing a (short) password only, with a session key (e.g, the key is used for data integrity and/or confidentiality). The focus of the present paper is on the analysis of very efficient schemes that have been proposed to the IEEE P1363 Standard working group on password-based authenticated key-exchange methods, but for which actual security was an open problem. We analyze the AuthA key exchange scheme and give a complete proof of its security. Our analysis shows that the AuthA protocol and its multiple modes of operation are provably secure under the computational Diffie-Hellman intractability assumption, in both the random-oracle and the ideal-cipher models.

1 Introduction

Problem. The need for secure authentication seems obvious when two entities—a client and a server—communicate on the wired-Internet, but proving an identity over a public link is complex. The method deployed by the engineers of the Secure Shell protocol (SSH) [2] to determine a client's identity to log him/her into another computer, execute commands on a remote machine, and move files from one machine to another is to ask him to type-in a password. The remote machine maintains the association between the client name and the password. Another method is to take advantage of a public-key infrastructure (PKI) to check that an entity knows the secret-key corresponding to the public-key embedded in a certificate. This method was adopted by the IETF TLS Working Group to secure the traffic between a web browser and a bank server over the wired-Internet, but work is currently under way to enrich this "transport layer" security protocol (TLS) with password-based authentication methods [18].

The primary *raison d'être* for password-based authentication is to enable clients to identify themselves to servers through a lightweight process since no security infrastructure or special hardware to carry the passwords is required. One example is when a password is used as a means to establish a secure communication channel from the computing device a human relies on to the remote machine he wants to talk to. This process, or password-authenticated key-exchange as it is often termed [6, 7, 21], provides the two computing devices with a session key to implement an authenticated communication channel within which messages sent over the wire are cryptographically protected. Humans directly benefit from this approach since they only need to remember a low-quality string (i.e. 4 decimal digits) chosen from a relatively small dictionary rather than a high-quality symmetric encryption key.

The fundamental security goal for a password-authenticated key exchange protocol to achieve is security against dictionary attacks. One can not actually prevent the adversary from guessing a value for the password and using this value in an attempt to impersonate a player. If the attack fails, the adversary can eliminate this value from the list of possible passwords. However, one would like this attack to be the only one the adversary can mount: after n active interactions with some participants the adversary should not be able to eliminate a greater number of passwords than n. Namely, a passive eavesdropping should be of no help to the adversary since an off-line exhaustive search on the password should not get any bias on the actual password. The off-line exhaustive search is called a *dictionary attack*.

The need for lightweight authentication processes is even greater in the case of the wireless-Internet. Wireless nodes are devices with particular mobility, computation and bandwidth requirements (diskless base station, cellular phone, pocket PC, palm pilot, laptop computer, base station gateway) that place severe restrictions when designing cryptographic mechanisms. The TLS protocol has been enriched with elliptic-curve cipher suites to run on low-power devices [8] and has within the WAP Forum evolved into a "transport layer" security protocol to secure mobile-commerce (WTLS) [22]. The Wired Equivalent Privacy (WEP) protocol, which is part of the IEEE 802.11 standard, does rely on high-quality symmetric encryption keys for protecting the wireless local-area network (WLAN) traffic between a mobile device equipped with a wireless ethernet-card and a fixed access point, but the WEP does not specify how the keys are established [9]. Currently, the IEEE 802.11 standard does not specify any method for key exchange.

Contributions. This paper examines the security of the AuthA password-authenticated key exchange protocol proposed to the IEEE P1363 Study Group on standard specifications for public-key cryptography [20]. Although AuthA has been conjectured cryptographically secure by its authors, it has still not been proven to resist dictionary attacks [4]. In this paper we provide a complete proof of security for the AuthA protocol. We work out our proofs by first defining the execution of AuthA in the communication model of Bellare *et al.* [3] and then adapting the proof techniques recently published by Bresson *et al.* [12] for the password-based group key exchange.

We have defined the execution of AuthA in 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. Our analysis shows that some of the AuthA modes of operation achieve provable security against dictionary attacks in both the random oracle and ideal-cipher models [3, 5] under the computational Diffie-Hellman intractability assumption.

Related Work. The IEEE P1363 Standard working group on password-based authenticated key-exchange methods [21] 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.* [3] and Boyko *et al.* [11], wherein formal models and security goals for password-based key agreement were first formulated. Bellare *et al.* analyzed the EKE protocol [6] (where EKE stands for *Encrypted Key Exchange*), a classical Diffie-Hellman key exchange wherein the two flows are encrypted using the password as common symmetric key. While they announced a security result of this "elegant" and efficient structure in both the random oracle and ideal-cipher models, the full proof never appeared anywhere. On the other hand, Boyko *et al.* [11] provided such a proof, but it was in another security model, the multi-party simulatability model. We thus provide a complete proof in the Bellare *et al.* security model, in a model where both a random oracle and an ideal-cipher are available.

One should note that Boyko *et al.*'s security result [11] holds in the random oracle model, while Bellare *et al.*'s one [3] holds in both the random oracle model and the ideal-cipher one together. More recent works provided password-based schemes for which security holds in the standard model only [15–17]. These are either based on general computational assumptions, or on the Decisional Diffie-Hellman problem (using a variant of the Cramer-Shoup encryption scheme [14].) While relying on a strong computational assumption, they are neither practical nor very efficient.

These provably secure schemes in the standard model are from a theoretical point of view very interesting, but fails to be practical. Ideal models (i.e. random-oracle, ideal-cipher) have thus been defined to provide alternative security results. While not being formal proofs, they give strong evidence that the schemes are not flawed. They often rely on weaker computational assumptions (e.g. the computational Diffie-Hellman problem instead of the decisional one.)

More interestingly, EKE later evolved into the proposal AuthA [4], which is formally modeled by the One-Encryption Key-Exchange (OEKE) in the present paper: only one flow is encrypted (using either a symmetric-encryption primitive or a multiplicative function as the product of a Diffie-Hellman value with a hash of the password). The advantage of such a scheme over the classical EKE, wherein the two Diffie-Hellman values are encrypted, is its easyness of integration. An OEKE cipher enables us to avoid many compatibility problems when adding password-based capabilities to existing network security protocols since the initial messages of the security protocols do not need to be modified. This argument in favor of OEKE was put forward when discussions were under way to enrich the Transport Layer Security (TLS) protocol with password-based key-exchange cipher suites [18, 19]. In a TLS One-Encryption Key-Exchange initiated by the server, the server does not need to know the client's name (a name is mapped to a password by the server using a local database) to compute and send out the server's TLS key-exchange message, but does need it to process the incoming client's TLS key-exchange message. Therefore, engineers embodied the client's name in the client's TLS key-exchange message rather than embodying it in the client's TLS hello message [18]. OEKE is thus of great practical interest, but none of the previous security analyses ever dealt with it.

Our paper is organized as follows. In Section 2, we recall the model and the definitions that should be satisfied by a password-based key exchange protocol. In Section 3, we show that OEKE, a "simplified" variant of a AuthA mode of operation, is secure. In Section 4, we build on this result to show that some of the AuthA modes of operation proposed to the IEEE P1363 Study Group are secure.

2 Model

In this section we recall the formal model for security against dictionary attacks where the adversary's capabilities are modeled through queries. In this model, the players do not deviate from the protocol and the adversary is not a player, but does control all the network communications.

2.1 Security Model

Players. We denote a *server* S and a user, or *client*, U 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 client instances and server instances by U^i and S^j (or by I when we consider any kind of instance).

The client and the server share a low-entropy secret pw which is (uniformly) drawn from a small dictionary Password of size N. The assumption of the uniform distribution for the password is just to make notations simpler, but everything would work with any other distribution, replacing the probability q/N by the sum of the probabilities of the q most probable passwords.

Abstract Interface. The protocol $Auth$A consists of the following algorithm:

– The *key exchange* algorithm $\textsc{KeyExch}(U^i, S^j)$ is an interactive protocol between U^i and S^j that provides the instances of U and S with a session key sk.

Queries. The adversary $\mathcal{A}$ interacts with the participants by making various queries. Let us explain the capability that each query captures:

– $\mathsf{Execute}(U^i, S^j)$: This query models passive attacks, where the adversary gets access to honest executions of P between U^i and S^j by eavesdropping.
– $\mathsf{Reveal}(I)$: This query models the misuse of the session key by instance I. The query is only available to $\mathcal{A}$ if the targetted instance actually "holds" a session key and it releases sk to $\mathcal{A}$.

– $\mathsf{Send}(I, m)$: This query models $\mathcal{A}$ sending a message to instance I. The adversary $\mathcal{A}$ gets back the response I generates in processing the message m according to the protocol P. A query $\mathsf{Send}(U^i, \mathtt{Start})$ initializes the key exchange algorithm, and thus the adversary receives the flow the client should send out to the server.

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 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 may have built by itself, and thus the number of passwords it would have tried.

2.2 Security Notions

Freshness. The freshness notion captures the intuitive fact that a session key is not "obviously" known to the adversary. An instance is said to be **Fresh** in the current protocol execution if the instance has accepted and neither it nor the other instance with the same session tag have been asked for a Reveal-query.

The* Test*-query. The semantic security of the session key is modeled by an additional query $\mathsf{Test}(I)$. 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 I is **Fresh**. This query is answered as follows: one flips a (private) coin b and forwards sk (the value $\mathsf{Reveal}(I)$ would output) if $b = 1$, or a random value if $b = 0$.

AKE Security. The security notions take place in the context of executing P in the presence of the adversary $\mathcal{A}$. The game $\mathbf{Game}^{\mathsf{ake}}(\mathcal{A}, P)$ is initialized by drawing a password pw from Password, providing coin tosses to $\mathcal{A}$, all oracles, and then running the adversary by letting it asking a polynomial number of queries as described above. At the end of the game, $\mathcal{A}$ outputs its guess b' for the bit b involved in the Test-query.

We denote the **AKE advantage** as the probability that $\mathcal{A}$ correctly guesses the value of b; more precisely we define $\mathsf{Adv}_P^{\mathsf{ake}}(\mathcal{A}) = 2\Pr[b = b'] - 1$, where the probability space is over all the random coins of the adversary and all the oracles. The protocol P is said to be **AKE-secure** if $\mathcal{A}$'s advantage is negligible in the security parameter.

Authentication. Another goal of the adversary is to impersonate the client or the server. In the present paper, we focus on unilateral authentication of the client, thus we denote by $\mathsf{Succ}_P^{\mathsf{c-auth}}(\mathcal{A})$ the probability that $\mathcal{A}$ successfully impersonates a client instance in an execution of P: this means that a server would accept a key while the latter is shared with no client. The protocol P is said to be **C-Auth-secure** if such a probability is negligible in the security parameter.

2.3 Computational Diffie-Hellman Assumption

Let $\mathbb{G} = \langle g \rangle$ be a finite cyclic group of order a ℓ-bit prime number q, where the operation is denoted multiplicatively. A (t, ε)-CDH attacker in $\mathbb{G}$ is a probabilistic machine Δ running in time t such that

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

where the probability is taken over the random values x and y. The CDH-Problem is (t, ε)**-intractable** if there is no (t, ε)-attacker in $\mathbb{G}$. The CDH-assumption states that is the case for all polynomial t and any non-negligible ε.

3 One-Encryption Key Exchange

In this section, we describe OEKE, a "simplified" variant of a AuthA mode of operation [4], and prove its security in the random oracle and the ideal-cipher models. At the core of this variant resides only

Client U		*Server S*
pw		pw
accept $\leftarrow$ false		accept $\leftarrow$ false
terminate $\leftarrow$ false		terminate $\leftarrow$ false
$x \stackrel{R}{\leftarrow} [1, q-1]$		$y \stackrel{R}{\leftarrow} [1, q-1]$
$X \leftarrow g^x$	$\xrightarrow{U, X}$	$Y \leftarrow g^y$
$Y \leftarrow \mathcal{D}_{pw}(Y^\star)$	$\xleftarrow{S, Y^\star}$	$Y^\star \leftarrow \mathcal{E}_{pw}(Y)$
$K_U \leftarrow Y^x$		$K_S \leftarrow X^y$
$Auth \leftarrow \mathcal{H}_1(U\|S\|X\|Y\|K_U)$		
$sk_U \leftarrow \mathcal{H}_0(U\|S\|X\|Y\|K_U)$		
accept $\leftarrow$ true	$\xrightarrow{Auth}$	$Auth \stackrel{?}{=} \mathcal{H}_1(U\|S\|X\|Y\|K_S)$
		if true, accept $\leftarrow$ true
		$sk_S \leftarrow \mathcal{H}_0(U\|S\|X\|Y\|K_S)$
terminate $\leftarrow$ true		terminate $\leftarrow$ true

Fig. 1. An execution of the protocol OEKE, run by the client U and the server S. The session key is $sk = \mathcal{H}_0(U\|S\|X\|Y\|Y^x) = \mathcal{H}_0(U\|S\|X\|Y\|X^y)$.

one flow of the basic Diffie-Hellman key exchange encrypted under the password and two protocol entities holding the same password. It therefore slightly differs from the original EKE [3, 6] in the sense that only one flow is encrypted using the password; instead of the two as usually done. But then, it is clear that at least one authentication flow has to be sent. We prove this is enough to satisfy the above security notions.

3.1 Description of the Scheme

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. Hash functions from $\{0,1\}^\star$ to $\{0,1\}^{\ell_0}$ and $\{0,1\}^{\ell_1}$ are denoted $\mathcal{H}_0$ and $\mathcal{H}_1$. A block cipher is denoted $(\mathcal{E}_k, \mathcal{D}_k)$ where $k \in \mathsf{Password}$. We also define $\bar{\mathbb{G}}$ to be equal to $\mathbb{G}\backslash\{1\}$, thus $\bar{\mathbb{G}} = \{g^x \mid x \in \mathbb{Z}_q^\star\}$.

As illustrated on Figure 1 (with an honest execution of the OEKE protocol), the protocol runs between a client U and a server S, and the session-key space $\mathbf{SK}$ associated to this protocol is $\{0,1\}^{\ell_0}$ equipped with a uniform distribution. Client and server initially share a low-quality string pw, the password, uniformly drawn from the dictionary Password.

The protocol consists of three flows. The client chooses a random exponent x and computes the value g^x which he sends to the server. The server in turn chooses a random exponent y, computes the value g^y, and encrypts the latter under the password pw before to send it out on the wire. Upon receiving the client's flow, the server computes the Diffie-Hellman secret value g^{xy}, and from it the session key sk. Upon receiving the server's flow, the client decrypts the ciphertext, computes the Diffie-Hellman secret value, and an authentication tag $Auth$ for client-to-server unilateral authentication. The client then sends out this authenticator. If the authenticator verifies on the server side, the client and the server have successfully exchanged the session key sk.

3.2 Semantic Security

In this section, we assert that under reasonable and well-defined intractability assumptions the protocol securely distributes session keys. More precisely, in this section, we deal with the semantic security goal. We consider the unilateral authentication goal in the next section. In the proof below, we do not consider forward-secrecy, for simplicity, but the semantic security still holds in this context, with slightly different bounds. The details can be found in the Appendix D. However, remember that any security result considers concurrent executions.

Theorem 1. *Let us consider the* OEKE *protocol, where* **SK** *is the session-key space and* Password *is a finite dictionary of size N equipped with the uniform distribution. Let $\mathcal{A}$ be an adversary against the AKE security of* OEKE *within a time bound t, with less than q_s interactions with the parties and q_p passive eavesdroppings, and, asking q_h hash-queries and q_e encryption/decryption queries. Then we have*

$$\mathsf{Adv}^{\mathsf{ake}}_{\mathsf{oeke}}(\mathcal{A}) \leq 3 \times \frac{q_s}{N} + 8q_h \times \mathsf{Succ}^{\mathsf{cdh}}_{\mathbb{G}}(t') + \frac{(2q_e + 3q_s + 3q_p)^2}{q-1} + \frac{q_h^2 + 4q_s}{2^{\ell_1}}.$$

where $t' \leq t + (q_s + q_p + q_e + 1) \cdot \tau_{\mathbb{G}}$, with $\tau_{\mathbb{G}}$ denoting the computational time for an exponentiation in $\mathbb{G}$. (Recall that q is the order of $\mathbb{G}$.)

This theorem shows that the OEKE 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. This is particularly significant in practice since a password may expire once a number of failed interactions has been achieved, whereas adversary's capability to enumerate passwords off-line is only limited by its computational power. Of course, this security result only holds provided that the adversary does not solve the computational Diffie-Hellman problem.

Proof (of Theorem 1). In this section we incrementally define a sequence of games starting at the real game $\mathbf{G}_0$ and ending up at $\mathbf{G}_8$.

Game* $\mathbf{G}_0$*: This is the real attack game, in the random oracle and ideal-cipher models. Several oracles are thus available to the adversary: two hash oracles ($\mathcal{H}_0$ and $\mathcal{H}_1$), the encryption/decryption oracles ($\mathcal{E}$ and $\mathcal{D}$), and all the instances U^i and S^j (in order to cover concurrent executions). We define several events in any game $\mathbf{G}_n$:

- event S_n occurs if $b = b'$, where b is the bit involved in the Test-query, and b' is the output of the AKE-adversary;
- event $\mathsf{Encrypt}_n$ occurs if $\mathcal{A}$ submits a data it has encrypted by itself using the password;
- event Auth_n occurs if $\mathcal{A}$ submits an authenticator $Auth$ that will be accepted by the server and that has been built by the adversary itself.

By definition,

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

In the games below, we furthermore assume that when the game aborts or stops with no answer b' outputted by the adversary $\mathcal{A}$, we choose this bit b' at random, which in turn defines the actual value of the event S_k. Moreover, if the adversary has not finished playing the game after q_s Send-queries or lasts for more than time t, we stop the game (and choose a random bit b'), where q_s and t are predetermined upper-bounds.

Game* $\mathbf{G}_1$*: In this game, we simulate the hash oracles ($\mathcal{H}_0$ and $\mathcal{H}_1$, but also two additional hash functions $\mathcal{H}_2 : \{0,1\}^\star \to \{0,1\}^{\ell_2}$ and $\mathcal{H}_3 : \{0,1\}^\star \to \{0,1\}^{\ell_3}$, with $\ell_2 = \ell_0$ and $\ell_3 = \ell_1$, that will appear in Game $\mathbf{G}_7$) and the encryption/decryption oracles, as usual by maintaining a hash list $\Lambda_{\mathcal{H}}$ (and another list $\Lambda_{\mathcal{A}}$ containing the hash-queries asked by the adversary itself) and an encryption list $\Lambda_{\mathcal{E}}$ (see Figure 2) We also simulate all the instances, as the real players would do, for the Send-queries (see Figure 3) and for the Execute, Reveal and Test-queries (see Figure 4).

From this simulation, we easily see that the game is perfectly indistinguishable from the real attack, unless the permutation property of $\mathcal{E}$ or $\mathcal{D}$ does not hold. One could have avoided collisions but this

For a hash-query $\mathcal{H}_i(q)$ (with $i \in \{0,1,2,3\}$), such that a record (i,q,r) appears in $\Lambda_{\mathcal{H}}$, the answer is r. Otherwise the answer r is defined according to the following rule:

►**Rule** $\mathcal{H}^{(1)}$ – Choose a random element $r \in \{0,1\}^{\ell_i}$.

The record (i,q,r) is added to $\Lambda_{\mathcal{H}}$. If the query is directly asked by the adversary, one adds (i,q,r) to $\Lambda_{\mathcal{A}}$.

For an encryption-query $\mathcal{E}_k(Z)$, such that a record $(k,Z,*,*,Z^\star)$ appears in $\Lambda_{\mathcal{E}}$, the answer is $Z^\star$. Otherwise the answer $Z^\star$ is defined according to the following rule:

►**Rule** $\mathcal{E}^{(1)}$ – Choose a random element $Z^\star \in \bar{\mathbb{G}}$.

Then one adds the record $(k,Z,\perp,\mathcal{E},Z^\star)$ to $\Lambda_{\mathcal{E}}$.

For a decryption-query $\mathcal{D}_k(Z^\star)$, such that a record $(k,Z,*,*,Z^\star)$ appears in $\Lambda_{\mathcal{E}}$, the answer is Z. Otherwise, one applies the following rule to obtain the answer Z:

►**Rule** $\mathcal{D}^{(1)}$ – Choose a random element $\varphi \in \mathbb{Z}_q^\star$, compute the answer $Z = g^\varphi$ and add the record $(k,Z,\varphi,\mathcal{D},Z^\star)$ to $\Lambda_{\mathcal{E}}$.

Fig. 2. Simulation of the random oracles, and the encryption/decryption oracles

We answer to the Send-queries to the client as follows:

- A $\mathsf{Send}(U^i, \mathtt{Start})$-query is processed according to the following rule:
  ►**Rule U1**$^{(1)}$ – Choose a random exponent $\theta \in \mathbb{Z}_q^\star$ and compute $X = g^\theta$.
  Then the query is answered with U, X, and the client instance goes to an expecting state.
- If the client instance U^i is in an expecting state, a query $\mathsf{Send}(U^i, (S, Y^\star))$ is processed by computing the session key and producing an authenticator. We apply the following rules:
  ►**Rule U2**$^{(1)}$ – Compute $Y = \mathcal{D}_{pw}(Y^\star)$ and $K_U = Y^\theta$.
  ►**Rule U3**$^{(1)}$ – Compute the authenticator $Auth = \mathcal{H}_1(U\|S\|X\|Y\|K_U)$ and the session key $sk_U = \mathcal{H}_0(U\|S\|X\|Y\|K_U)$.
  Finally the query is answered with $Auth$, the client instance accepts and terminates. Our simulation also adds $((U,X),(S,Y^\star),Auth)$ to Λ_Ψ. The variable Λ_Ψ keeps track of the exchanged messages.

We answer to the Send-queries to the server as follows:

- A $\mathsf{Send}(S^j, (U, X))$-query is processed according to the following rule:
  ►**Rule S1**$^{(1)}$ – Choose a random exponent $\varphi \in \mathbb{Z}_q^\star$, compute $Y = g^\varphi$, $Y^\star = \mathcal{E}_{pw}(Y)$ and $K_S = X^\varphi$.
  Finally, the query is answered with $S, Y^\star$ and the server instance goes to an expecting state.
- If the server instance S^j is in an expecting state, a query $\mathsf{Send}(S^j, H)$ is processed according to the following rules:
  ►**Rule S2**$^{(1)}$ – Compute $H' = \mathcal{H}_1(U\|S\|X\|Y\|K_S)$, and check whether $H = H'$. If the equality does not hold, the server instance terminates without accepting.
  If equality holds, the server instance accepts and goes on, applying the following rule:
  ►**Rule S3**$^{(1)}$ – Compute the session key $sk_S = \mathcal{H}_0(U\|S\|X\|Y\|K_S)$.
  Finally, the server instance terminates.

Fig. 3. Simulation of the Send-queries

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

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

A $\mathsf{Test}(I)$-query first gets sk from $\mathsf{Reveal}(I)$, 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 $\{0,1\}^{\ell_0}$.

Fig. 4. Simulation of the Execute, Reveal and Test-queries

happens with probability at most $q_{\mathcal{E}}^2/2(q-1)$ since $|\bar{\mathbb{G}}| = (q-1)$, where $q_{\mathcal{E}}$ is the size of $\Lambda_{\mathcal{E}}$:

$$|\Pr[\mathsf{S}_1] - \Pr[\mathsf{S}_0]| \leq \frac{q_{\mathcal{E}}^2}{2(q-1)}. \tag{2}$$

Game* $\mathbf{G}_2$*: We define game $\mathbf{G}_2$ by modifying the way the server processes the Send-queries so that the adversary will be the only one to encrypt data. We use the following rule:

- ►**Rule** $\mathbf{S1}^{(2)}$ – Choose a random $Y^\star \in \bar{\mathbb{G}}$ and compute $Y = \mathcal{D}_{pw}(Y^\star)$. Look for the record $(pw, Y, \varphi, *, Y^\star)$ in the list $\Lambda_{\mathcal{E}}$ to define φ (we thus have $Y = g^\varphi$), and finally compute $K_S = X^\varphi$.

The two games $\mathbf{G}_2$ and $\mathbf{G}_1$ are perfectly indistinguishable unless $\varphi = \perp$. This happens when $Y^\star$ has been previously obtained as the ciphertext returned by an encryption-query. Note that this may happen when processing a Send-query, but also during a passive simulation when processing an $\mathsf{Execute}$-query:

$$|\Pr[\mathsf{S}_2] - \Pr[\mathsf{S}_1]| \leq \frac{q_S q_{\mathcal{E}}}{q-1}, \tag{3}$$

where q_S is the number of involved server instances: $q_S \leq q_s + q_p$. Furthermore note that from now, only the adversary may ask encryption queries, since the server is simulated using the decryption oracle.

Game* $\mathbf{G}_3$*: In this game, we avoid collisions amongst the hash queries asked by the adversary to $\mathcal{H}_1$, amongst the passwords and the ciphertexts, and amongst the output of the Send-queries. We play the game in a way that: no collision has been found by the adversary for $\mathcal{H}_1$; no encrypted data corresponds to multiple identical plaintext; at most one password corresponds to each plaintext-ciphertext pair; abort if two instances of the server have used the same random values. This will help us later on to prove Lemma 2, the key step in proving Theorem 1. We use the following rules:

- ►**Rule** $\mathcal{H}^{(3)}$ – Choose a random element $r \in \{0,1\}^{\ell_i}$. If $i = 1$, this query is directly asked by the adversary, and $(1, *, r) \in \Lambda_{\mathcal{A}}$, then we abort the game.

Then, for any H, $\#\{(1, *, H) \in \Lambda_{\mathcal{A}}\} \leq 1$. But this rule may make the game to abort with probability bounded by $q_h^2/2^{\ell_1+1}$

- ►**Rule** $\mathcal{E}^{(3)}$ – Choose a random element $Z^\star \in \bar{\mathbb{G}}$. If $(*, *, \perp, \mathcal{E}, Z^\star) \in \Lambda_{\mathcal{E}}$, we abort the game.

Then, for any $Z^\star$, $\#\{(*, *, \perp, \mathcal{E}, Z^\star) \in \Lambda_{\mathcal{E}}\} \leq 1$. But this rule may make the game to abort with probability bounded by $q_{\mathcal{E}}^2/2(q-1)$.

- ►**Rule** $\mathcal{D}^{(3)}$ – Choose a random element $\varphi \in \mathbb{Z}_q^\star$ and compute $Z = g^\varphi$. If $(*, Z, *, *, Z^\star) \in \Lambda_{\mathcal{E}}$, we abort the game. Otherwise, we add the record $(k, Z, \varphi, \mathcal{D}, Z^\star)$ to $\Lambda_{\mathcal{E}}$.

Then, for any pair $(Z, Z^\star)$, $\#\{(*, Z, *, *, Z^\star) \in \Lambda_{\mathcal{E}}\} \leq 1$. But this rule may make the game to abort with probability bounded by $q_{\mathcal{E}}^2/2(q-1)$.

- ►**Rule** $\mathbf{S1}^{(3)}$ – Choose a random $Y^\star \in \bar{\mathbb{G}}$. If $(*, Y^\star) \in \Lambda_S$, abort the game, otherwise add the record $(j, Y^\star)$ to Λ_S. Then, compute $Y = \mathcal{D}_{pw}(Y^\star)$, look for the record $(pw, Y, \varphi, *, Y^\star)$ in $\Lambda_{\mathcal{E}}$ to define φ (we thus have $Y = g^\varphi$), and compute $K_S = X^\varphi$. The variable Λ_S keeps track of the messages sent out by the server S.

Then, there is no collision among the $Y^\star$ outputted by the server instances (and thus the used Y). But this rule may make the game to abort with probability bounded by the birthday paradox, $q_S^2/2(q-1)$, where q_S is again the number of involved server instances.

The two games $\mathbf{G}_3$ and $\mathbf{G}_2$ are perfectly indistinguishable unless one of the above rules make the game to abort:

$$|\Pr[\mathsf{S}_3] - \Pr[\mathsf{S}_2]| \leq \frac{2q_{\mathcal{E}}^2 + q_S^2}{2(q-1)} + \frac{q_h^2}{2^{\ell_1+1}}. \tag{4}$$

Game* $\mathbf{G}_4$*: We define game $\mathbf{G}_4$ by aborting the executions wherein the adversary may have guessed the password and used it to send an encrypted data to the client. We achieve this aim by modifying the way the client processes the queries. We use the following rule:

►**Rule U2**$^{(4)}$ – Look for $(pw, *, \perp, \mathcal{E}, Y^\star) \in \Lambda_{\mathcal{E}}$. If the record is found, define $\mathsf{Encrypt}_4$ as true and abort the game. Otherwise, compute $Y = \mathcal{D}_{pw}(Y^\star)$ and $K_U = Y^\theta$.

The two games $\mathbf{G}_4$ and $\mathbf{G}_3$ are perfectly indistinguishable unless event $\mathsf{Encrypt}_4$ occurs:

$$|\Pr[\mathsf{S}_4] - \Pr[\mathsf{S}_3]| \leq \Pr[\mathsf{Encrypt}_4]. \tag{5}$$

Game* $\mathbf{G}_5$*: We define game $\mathbf{G}_5$ by aborting the executions wherein the adversary may have been lucky in guessing the authenticator (that is, without asking the corresponding hash query). We reach this aim by modifying the way the server processes the queries:

►**Rule S2**$^{(5)}$ – Check whether $H = H'$, where $H' = \mathcal{H}_1(U\|S\|X\|Y\|K_S)$. If the equality does hold, check if $(1, U\|S\|X\|Y\|K_S, H) \in \Lambda_{\mathcal{A}}$ or $((U,X),(S,Y^\star),H) \in \Lambda_\Psi$. If these two latter tests fail, then reject the authenticator: terminate, without accepting. If this rule does not make the server to terminate, the server accepts and moves on.

This rule ensures that all accepted authenticators will come from either the simulator, or an adversary that has correctly decrypted $Y^\star$ into Y, (computed K_S) and asked the query to the oracle $\mathcal{H}_1$. The two games $\mathbf{G}_5$ and $\mathbf{G}_4$ are perfectly indistinguishable unless the server rejects a valid authenticator. Since Y did not appear in a previous session (since the Game $\mathbf{G}_3$), this happens only if the authenticator had been correctly guessed by the adversary without asking $\mathcal{H}_1(U\|S\|X\|Y\|K_S)$:

$$|\Pr[\mathsf{Encrypt}_5] - \Pr[\mathsf{Encrypt}_4]| \leq \frac{q_s}{2^{\ell_1}} \qquad |\Pr[\mathsf{S}_5] - \Pr[\mathsf{S}_4]| \leq \frac{q_s}{2^{\ell_1}}. \tag{6}$$

Game* $\mathbf{G}_6$*: We define game $\mathbf{G}_6$ by aborting the executions wherein the adversary may have guessed the password (that is the adversary has correctly decrypted $Y^\star$ into Y) and then used it to build and send a valid authenticator to the server. We reach this aim by modifying the way the server processes the queries:

►**Rule S2**$^{(6)}$ – Check if $((U,X),(S,Y^\star),H) \in \Lambda_\Psi$. If this is not the case, then reject the authenticator: terminate, without accepting. Check if $(1, U\|S\|X\|Y\|*, H) \in \Lambda_{\mathcal{A}}$. If this is the case, we define the event Auth'_6 to be true, and abort the game.

This rule ensures that all accepted authenticators come from the simulator. The two games $\mathbf{G}_6$ and $\mathbf{G}_5$ are perfectly indistinguishable unless either $(1, U\|S\|X\|Y\|K_S, H) \in \Lambda_{\mathcal{A}}$ or $(1, U\|S\|X\|Y\|*, H) \in \Lambda_{\mathcal{A}}$, which both lead to Auth'_6 to be true:

$$|\Pr[\mathsf{Encrypt}_6] - \Pr[\mathsf{Encrypt}_5]| \leq \Pr[\mathsf{Auth}'_6] \qquad |\Pr[\mathsf{S}_6] - \Pr[\mathsf{S}_5]| \leq \Pr[\mathsf{Auth}'_6]. \tag{7}$$

Game* $\mathbf{G}_7$*: In this game, we do no compute the authenticator $Auth$ and the session key sk using the oracles $\mathcal{H}_0$ and $\mathcal{H}_1$, but using the private oracles $\mathcal{H}_2$ and $\mathcal{H}_3$ so that the values $Auth$ and sk are completely independent from $\mathcal{H}_0$ and $\mathcal{H}_1$, but also Y, pw and any of K_U or K_S. We reach this aim by using the following rules:

►**Rule U3**$^{(7)}$ – Compute the session key $sk_U = \mathcal{H}_2(U\|S\|X\|Y^\star)$ and the authenticator $Auth = \mathcal{H}_3(U\|S\|X\|Y^\star)$.

►**Rule S3**$^{(7)}$ – Compute the session key $sk_S = \mathcal{H}_2(U\|S\|X\|Y^\star)$.

Since we do no longer need to compute the values K_U and K_S, we can also simplify the way client and server process the queries:

►**Rule U2**$^{(7)}$ – Look for a record $(pw, *, \perp, \mathcal{E}, Y^\star)$ in $\Lambda_\mathcal{E}$. If the record is found, we define $\mathsf{Encrypt}_7$ as true and abort the game.

►**Rule S1**$^{(7)}$ – Choose a random $Y^\star \in \bar{\mathbb{G}}$. If $(*, Y^\star) \in \Lambda_S$, one aborts the game, otherwise adds the record $(j, Y^\star)$ to Λ_S. Then, compute $Y = \mathcal{D}_{pw}(Y^\star)$.

The games $\mathbf{G}_7$ and $\mathbf{G}_6$ are indistinguishable unless the following event AskH occurs: $\mathcal{A}$ queries the hash functions $\mathcal{H}_0$ or $\mathcal{H}_1$ on $U\|S\|X\|Y\|K_U$ or on $U\|S\|X\|Y\|K_S$, that is on the common value $U\|S\|X\|Y\|\mathsf{CDH}(X,Y)$:

$$|\Pr[\mathsf{Encrypt}_7] - \Pr[\mathsf{Encrypt}_6]| \leq \Pr[\mathsf{AskH}_7] \qquad |\Pr[\mathsf{S}_7] - \Pr[\mathsf{S}_6]| \leq \Pr[\mathsf{AskH}_7]$$
$$\left|\Pr[\mathsf{Auth}'_7] - \Pr[\mathsf{Auth}'_6]\right| \leq \Pr[\mathsf{AskH}_7]. \tag{8}$$

Lemma 2. *The probabilities of the events* S_7, $\mathsf{Encrypt}_7$, *and* Auth'_7 *in game* $\mathbf{G}_7$ *can be upper-bounded by the following values:*

$$\Pr[\mathsf{S}_7] = \frac{1}{2} \qquad \Pr[\mathsf{Encrypt}_7] \leq \frac{q_s}{2N} \qquad \Pr[\mathsf{Auth}'_7] \leq \frac{q_s}{2N}. \tag{9}$$

Proof. The formal proof of this lemma can be found in the Appendix A.1. The main idea in simulating this game is to choose the password pw at the end of the game. The password pw is in fact only needed to determine whether the events $\mathsf{Encrypt}_7$ or Auth'_7 have occurred, and it turns out that determining whether these events have occurred can be postponed until the time limit has been reached or the adversary has asked q_s queries. The probabilities of $\mathsf{Encrypt}_7$ or Auth'_7 can then be easily upper-bounded since no information, in the information theoretical sense, about the password pw is known by the adversary along this simulation. □

Game $\mathbf{G}_8$***:*** In this game, we simulate the executions using the random self-reducibility of the Diffie-Hellman problem, given one CDH instance (A, B). We do not need to known the values of θ and φ, since the values K_U or K_S are no longer needed to compute the authenticator and the session keys:

►**Rule U1**$^{(8)}$ – Choose a random element $\alpha \in \mathbb{Z}_q^\star$, and compute $X = A^\alpha$. Also add the record (α, X) to Λ_A.

►**Rule** $\mathcal{D}^{(8)}$ – Choose a random element $\beta \in \mathbb{Z}_q^\star$, and compute the answer $Z = B^\beta$. Also add the record (β, Z) to Λ_B. If $(*, Z, *, *, Z^\star) \in \Lambda_\mathcal{E}$ then we abort the game; otherwise we add the record $(k, Z, \perp, \mathcal{D}, Z^\star)$ to $\Lambda_\mathcal{E}$.

$$\Pr[\mathsf{AskH}_8] = \Pr[\mathsf{AskH}_7]. \tag{10}$$

Remember that AskH_8 means that the adversary $\mathcal{A}$ had queried the random oracles $\mathcal{H}_0$ or $\mathcal{H}_1$ on $U\|S\|X\|Y\|Z$, where $Z = \mathsf{CDH}(X,Y)$. By picking randomly in the $\Lambda_\mathcal{A}$-list we can get the Diffie-Hellman secret value with probability $1/q_h$. This is a triple $(X, Y, \mathsf{CDH}(X,Y))$. We can then simply look in the lists Λ_A and Λ_B to find the values α and β such that $X = A^\alpha$ and $Y = B^\beta$:

$$\mathsf{CDH}(X,Y) = \mathsf{CDH}(A^\alpha, B^\beta) = \mathsf{CDH}(A,B)^{\alpha\beta}.$$

Thus:

$$\Pr[\mathsf{AskH}_8] \leq q_h \mathsf{Succ}^{\mathsf{cdh}}_{\mathbb{G}}(t'). \tag{11}$$

This concludes the proofs (the details of the computations can be found in the Appendix A.2. Simply note that $q_{\mathcal{E}}$ is the size of $\Lambda_{\mathcal{E}}$, which contains all the encryption/decryption queries directly asked by the adversary, but also all the decryption queries made by our simulation: at most one per Send-query (direct or through $\mathsf{Execute}$-queries), which makes $q_{\mathcal{E}} \leq q_e + q_s + q_p$. Similarly, q_S is the number of involved server instances, and thus $q_S \leq q_s + q_p$. Furthermore, one can easily see that in this last game, $t' \leq t + (q_s + q_p + q_e + 1) \cdot \tau_{\mathbb{G}}$. □

3.3 Unilateral Authentication

The following theorem shows that the OEKE protocol furthermore ensures authentication from client to server, in the sense that a server instance will never accept an authenticator that has not actually been sent by the corresponding/expected client instance with probability significantly greater than q_s/N.

Theorem 3. *Let us consider the OEKE protocol, where* **SK** *is the session-key space and* $\mathsf{Password}$ *a finite dictionary of size N equipped with the uniform distribution. Let $\mathcal{A}$ be an adversary against the AKE security of OEKE within a time bound t, with less than q_s interactions with the parties and q_p passive eavesdroppings, and, asking q_h hash-queries and q_e encryption/decryption queries. Then we have*

$$\mathsf{Adv}^{\mathsf{c-auth}}_{\mathsf{oeke}}(\mathcal{A}) \leq \frac{3}{2} \times \frac{q_s}{N} + 3q_h \times \mathsf{Succ}^{\mathsf{cdh}}_{\mathbb{G}}(t') + \frac{(2q_e + 3q_s + 3q_p)^2}{2(q-1)} + \frac{q_h^2 + 4q_s}{2^{\ell_1+1}}.$$

where $t' \leq t + (q_s + q_p + q_e + 1)\tau_{\mathbb{G}}$, with $\tau_{\mathbb{G}}$ denoting the computational time for an exponentiation in $\mathbb{G}$. (Recall that q is the order of $\mathbb{G}$.)

Proof. The proof is similar to the previous one. But one can find more details in the Appendix B. □

4 Applications

We describe some applications of our security results. We first show that some of the AuthA modes of operations [4] proposed to the IEEE P1363 Standard working group encompass particular cases of OEKE. Then, we make the ideal-cipher model more concrete.

4.1 Verifier-based Key Exchange

The AuthA protocol standardized by the IEEE organization is slightly different from our protocol since client and server do not share a password pw. The AuthA has an added mechanism preventing an adversary corrupting the password table of a server from impersonating a client at once. The AuthA protocol takes advantage of the asymmetric cryptography principles when generating the passwords hold by the client and the server. The client holds a derived password $pw_U = \mathcal{H}'(U\|S\|\mathsf{PW})$ (where PW is the actual password, and pw_U has the same entropy but in $\mathbb{Z}_q^\star$) and the server holds a value pw_S derived from the latter password as follows $pw_S = g^{pw_U}$. It has the same entropy as PW too. It is then straightforward to modify our protocol to make use of these values pw_U and pw_S rather than just the shared password pw (see Figure 5): pw_S plays the role of the common password, and

$$\mathcal{H}_0(U\|S\|X\|Y\|Z) \leftarrow \mathcal{H}(\mathcal{H}(U\|S\|X\|Y\|Z)\|0) \quad \mathcal{H}_1(U\|S\|X\|Y\|Z) \leftarrow \mathcal{H}(\mathcal{H}(U\|S\|X\|Y\|Z)\|Y^{pw_U}).$$

As a consequence, one can claim exactly the same security results about this scheme as the ones stated in the Theorems 1 and 3. More details can be found in the Appendix C.

Client		*Server*
pw_U		$pw_S = g^{pw_U}$
accept $\leftarrow$ false terminate $\leftarrow$ false		accept $\leftarrow$ false terminate $\leftarrow$ false
$x \xleftarrow{R} [1, q-1]$ $X \leftarrow g^x, \quad X^\star \leftarrow X$		$y \xleftarrow{R} [1, q-1]$ $Y \leftarrow g^y, \quad Y^\star \leftarrow \mathcal{E}_{pw_S}(Y)$
	$\xrightarrow{U, X^\star} \cdots \xrightarrow{U, \underline{X^\star}}$	
	$\xleftarrow{S, \underline{Y^\star}} \cdots \xleftarrow{S, Y^\star}$	
$\underline{Y} \leftarrow \mathcal{D}_{pw_S}(\underline{Y^\star}), \quad K_U \leftarrow \underline{Y}^x$ $\mathsf{PW}_U \leftarrow \underline{Y}^{pw_U}$		$\underline{X} \leftarrow \underline{X^\star}, \quad K_S \leftarrow \underline{X}^y$ $\mathsf{PW}_S \leftarrow {pw_S}^y$
$MK_U \leftarrow \mathcal{H}(U \| S \| X \| \underline{Y} \| K_U)$ $Auth \leftarrow \mathcal{H}(MK_U \| \mathsf{PW}_U)$ $sk_U \leftarrow \mathcal{H}(MK_U \| 0)$ accept $\leftarrow$ true		
	$\xrightarrow{Auth} \cdots \xrightarrow{\underline{Auth}}$	
		$MK_S \leftarrow \mathcal{H}(U \| S \| \underline{X} \| Y \| K_S)$ $\underline{Auth} \stackrel{?}{=} \mathcal{H}(MK_S \| \mathsf{PW}_S)$ if true, accept $\leftarrow$ true $sk_S \leftarrow \mathcal{H}(MK_S \| 0)$
terminate $\leftarrow$ true		terminate $\leftarrow$ true

Fig. 5. The AuthA protocol run by the client U and the server S – The session key for U is $sk_U = \mathcal{H}(\mathcal{H}(U\|S\|X\|\underline{Y}\|\underline{Y}^x)\|0)$. The session key for S is $sk_S = \mathcal{H}(\mathcal{H}(U\|S\|\underline{X}\|Y\|\underline{X}^y)\|0)$.

4.2 The AuthA Modes of Operation

When engineers choose a password-based key exchange scheme, they take into account its security, computation and communication efficiency, and easiness of integration. Since they do not all face the same computing environment, they may want to operate the AuthA protocol in different ways: encrypt both flows of the basic Diffie-Hellman key exchange; achieve mutual-authentication; the server sends out the first protocol flow. These different ways have already been described in [4] and do not seem to alter the security of the AuthA protocol. But more precise security analyses similar to the above ones should be performed before actually using the other modes.

4.3 Instantiating the Encryption Function

It is clear that a simple block-cipher can not be used in place of the ideal-cipher required by the security result. We indeed need permutations onto $\mathbb{G}$ for all the secret keys, otherwise partition attacks can be mounted [10]. In specific cases where the encoding of the elements is compact, on can use the iterated technique [1]: one encrypts the element, and reencrypts the result, until one finally falls in the group $\mathbb{G}$. Decryption operates the same way. With well-chosen elliptic curves, the average number of iterations can be bounded by 2. Furthermore, the size of the blocks can thus be less than 256 bits. However, one must be careful in the implementation to prevent timing attacks.

A promising avenue is to also instantiate the encryption primitive as the product of a Diffie-Hellman value with a hash of the password, as suggested in AuthA [4]. Preliminary investigations have shown that this multiplicative function leads to a password-based key-exchange scheme secure in the random-oracle model only [13].

5 Conclusion

The reductions presented in this paper are not optimal, but our intend was to present easy to read, understand and meaningful proofs rather than very efficient ones. We think that the terms $3q_s/2N$ or $3q_s/N$ can be improved to q_s/N, but the proof would then in turn becomes very intricate. For technical reasons the hash function $\mathcal{H}_1$ used to build the authenticator has to be collision-resistant in our proofs, but the authors of AuthA [4] suggest to use a 64-bit authenticator. This may turn out to be enough in practice, but the proof presented in the paper would then need to be modified. It, however, seems a bad idea to use the same hash function $\mathcal{H}$ everywhere in AuthA.

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-51868. Disclaimer available at `http://www-library.lbl.gov/disclaimer`.

References

1. M. Bellare, A. Boldyreva, A. Desai, and D. Pointcheval. Key-Privacy in Public-Key Encryption. In *Asiacrypt '01*, LNCS 2248, pages 566–582. Springer-Verlag, Berlin, 2001.
2. M. Bellare and T. Kohno and C. Namprempre. Authenticated Encryption in SSH: Provably Fixing the SSH Binary Packet Protocol. In *Proc. of the 9th CCS*. ACM Press, New York, 2002.
3. 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.
4. 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/`.
5. 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.
6. 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.
7. 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.
8. S. Blake-Wilson, V. Gupta, C. Hawk, and B. Moeller. ECC Cipher Suites for TLS, February 2002. IEEE RFC 20296.
9. N. Borisov, I. Goldberg, and D. Wagner. Intercepting Mobile Communications: The Insecurity of 802.11. In *Proc. of ACM International Conference on Mobile Computing and Networking (MobiCom'01)*, 2001.
10. C. Boyd, P. Montague, and K. Nguyen. Elliptic Curve Based Password Authenticated Key Exchange Protocols. In *ACISP '01*, LNCS 2119, pages 487–501. Springer-Verlag, Berlin, 2001.
11. 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.
12. E. Bresson, O. Chevassut, and D. Pointcheval. Group Diffie-Hellman Key Exchange Secure against Dictionary Attacks. In *Asiacrypt '02*, LNCS 2501, pages 497–514. Springer-Verlag, Berlin, 2002.
13. E. Bresson, O. Chevassut, and D. Pointcheval. Encrypted Key Exchange using Mask Generation Function. Work in progress.
14. R. Cramer and V. Shoup. A Practical Public Key Cryptosystem Provably Secure against Adaptive Chosen Ciphertext Attack. In *Crypto '98*, LNCS 1462, pages 13–25. Springer-Verlag, Berlin, 1998.
15. O. Goldreich and Y. Lindell. Session-Key Generation Using Human Passwords Only. In *Crypto '01*, LNCS 2139, pages 408–432. Springer-Verlag, Berlin, 2001.
16. J. Katz, R. Ostrovsky, and M. Yung. Efficient Password-Authenticated Key Exchange Using Human-Memorizable Passwords. In *Eurocrypt '01*, LNCS 2045, pages 475–494. Springer-Verlag, Berlin, 2001.
17. J. Katz, R. Ostrovsky, and M. Yung. Forward Secrecy in Password-only Key Exchange Protocols. In *Proc. of SCN '02*, 2002.
18. 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.
19. D. Taylor. Using SRP for TLS Authentication, november 2002. Internet Draft.

20. IEEE Standard 1363–2000. Standard Specifications for Public Key Cryptography. IEEE. Available from `http://grouper.ieee.org/groups/1363`, August 2000.
21. IEEE Standard 1363.2 Study Group. Password-Based Public-Key Cryptography. Available from `http://grouper.ieee.org/groups/1363/passwdPK`.
22. Wireless Application Protocol. Wireless Transport Layer Security Specification, February 2000. *WAP TLS, WAP-199 WTLS.*

A Complements for the Proof of Theorem 1

A.1 Proof of Lemma 2

Game $\mathbf{G}_7$***:*** In this game, we compute the authenticator sk_U and the session key sk_S using the private oracles $\mathcal{H}_2$ and $\mathcal{H}_3$ as depicted on Figure 6. Generating these values by querying the private oracles only X and $Y^\star$ enable us to no longer need to compute the values Y, K_U, and K_S for the simulation, but just to compute them at the end with the actual value of pw for defining the events $\mathsf{Encrypt}_7$ and Auth'_7.

The **Rule U2**$^{(7)}$, **Rule S1**$^{(7)}$ and **Rule S2**$^{(7)}$ can indeed be rewritten as rules that do not need the password along the simulation, but only make use of it at the end of the simulation. One can easily see on Figure 7 that the **Rule U2+**$^{(7)}$ and **Rule S2+**$^{(7)}$ are not useful for the simulation, but that they are only useful to determine whether events $\mathsf{Encrypt}_7$ or Auth'_7 occurred. They can thus be postponed until the adversary has asked q_s queries, or time limit expired. But then, one can note that the password pw is not used anymore, until these last rules are proceeded: one can run the simulation, without any password, and just choose it before processing these two rules.

Let us denote by $R(U)$ the set of $Y^\star$ received by a client instance, and by $R(S)$ the set of $(H, Y^\star)$ used by a server instance. From an information theoretical point of view, since we have avoided collisions in the Game $\mathbf{G}_3$,

$$\Pr[\mathsf{Encrypt}_7] = \Pr_{pw}[\exists Y^\star \in R(U), (pw, *, \perp, \mathcal{E}, Y^\star) \in \Lambda_{\mathcal{E}}] \leq \frac{\#R(U)}{N}$$

$$\Pr[\mathsf{Auth}'_7] = \Pr_{pw}[\exists (H, Y^\star) \in R(S), Y \leftarrow \mathcal{D}_{pw}(Y^\star), (1, U\|S\|X\|Y\|*, H) \in \Lambda_{\mathcal{A}}] \leq \frac{\#R(S)}{N}.$$

By definition of the sets $R(U)$ and $R(S)$, since $Y^\star$ is received in the second query to the user, and H in the second query to the server, the cardinalities are both upper-bounded by $q_s/2$.

Moreover, the session keys are random, independent from any other data (from an information theoretical point of view, since $\mathcal{H}_2$ and $\mathcal{H}_3$ are private random oracles). Then, $\Pr[\mathsf{S}_7] = 1/2$. □

A.2 Conclusion of the Proof of Theorem 1

By summing up all the relations, one completes the proof. From Equations (1), (2), (3), (4) and (5),

$$\begin{aligned} |\Pr[\mathsf{S}_4] - \Pr[\mathsf{S}_0]| &\leq \frac{q_{\mathcal{E}}^2}{2(q-1)} + \frac{q_S q_{\mathcal{E}}}{q-1} + \frac{2q_{\mathcal{E}}^2 + q_S^2}{2(q-1)} + \frac{q_h^2}{2^{\ell_1+1}} + \Pr[\mathsf{Encrypt}_4] \\ &\leq \frac{(2q_{\mathcal{E}} + q_S)^2}{2(q-1)} + \frac{q_h^2}{2^{\ell_1+1}} + \Pr[\mathsf{Encrypt}_4] \end{aligned}$$

From Equations (6 – 8), $|\Pr[\mathsf{Encrypt}_7] - \Pr[\mathsf{Encrypt}_4]|$ and $|\Pr[\mathsf{S}_7] - \Pr[\mathsf{S}_4]|$ are both upper-bounded by

$$\frac{q_s}{2^{\ell_1}} + \Pr[\mathsf{Auth}'_6] + \Pr[\mathsf{AskH}_7] \leq \frac{q_s}{2^{\ell_1}} + \Pr[\mathsf{Auth}'_7] + 2\Pr[\mathsf{AskH}_7]. \tag{12}$$

Then,

$$\begin{aligned} |\Pr[\mathsf{S}_7] - \Pr[\mathsf{S}_0]| \leq{} & \frac{(2q_{\mathcal{E}} + q_S)^2}{2(q-1)} + \frac{q_h^2}{2^{\ell_1+1}} + \frac{2q_s}{2^{\ell_1}} \\ & + \Pr[\mathsf{Encrypt}_7] + 2\Pr[\mathsf{Auth}'_7] + 4\Pr[\mathsf{AskH}_7]. \end{aligned}$$

We answer to the Send-queries to the client as follows:

- A $\mathsf{Send}(U^i, \mathtt{Start})$-query is processed according to the following rule:
  ►**Rule U1**$^{(7)}$ – Choose a random exponent $\theta \in \mathbb{Z}_q^\star$ and compute $X = g^\theta$.
  Then the query is answered with U, X, and the client instance goes to an expecting state.
- If the client instance U^i is in an expecting state, a query $\mathsf{Send}(U^i, (S, Y^\star))$ is processed by computing the session key and producing an authenticator. We apply the following rules:
  ►**Rule U2**$^{(7)}$ – Lookup $(pw, *, \perp, \mathcal{E}, Y^\star) \in \Lambda_\mathcal{E}$. If found, define $\mathsf{Encrypt}_7$ as true and abort the game.

  ►**Rule U3**$^{(7)}$ – Compute the session key $sk_U = \mathcal{H}_2(U\|S\|X\|Y^\star)$ and the authenticator $Auth = \mathcal{H}_3(U\|S\|X\|Y^\star)$.

  Finally the query is answered with $Auth$, the client instance accepts and terminates. Our simulation also adds $((U, X), (S, Y^\star), Auth)$ to Λ_Ψ.

We answer to the Send-queries to the server as follows:

- A $\mathsf{Send}(S^j, (U, X))$-query is processed according to the following rule:
  ►**Rule S1**$^{(7)}$ – Choose a random $Y^\star \in \bar{\mathbb{G}}$. If $(*, Y^\star) \in \Lambda_S$, one aborts the game, otherwise adds the record $(j, Y^\star)$ to Λ_S. Then, compute $Y = \mathcal{D}_{pw}(Y^\star)$.
  Finally, the query is answered with $S, Y^\star$ and the server instance goes to an expecting state.
- If the server instance S^j is in an expecting state, a query $\mathsf{Send}(S^j, H)$ is processed according to the following rules:
  ►**Rule S2**$^{(7)}$ – Check if $(X, Y^\star, H) \in \Lambda_\Psi$. If this is not the case, then reject the authenticator: terminate, without accepting. Check if $(1, U\|S\|X\|Y\|*, H) \in \Lambda_\mathcal{A}$. If this is the case, we define the event Auth'_7 to be true, and abort the game.
  If the server instance has not terminated, it accepts and moves on to apply the following rule:
  ►**Rule S3**$^{(7)}$ – Compute the session key $sk_S = \mathcal{H}_2(U\|S\|X\|Y^\star)$.
  Finally, the server instance terminates.

Fig. 6. Simulation of the Send-queries in $\mathbf{G}_7$

We first rewrite the **Rule U2**:

►**Rule U2-**$^{(7)}$ – Does nothing.

►**Rule U2+**$^{(7)}$ – Lookup $(pw, *, \perp, \mathcal{E}, Y^\star) \in \Lambda_\mathcal{E}$. If found, define $\mathsf{Encrypt}_7$ as true (and abort the game).

We then modify the organization of the **Rule S1** and the **Rule S2**:

►**Rule S1-**$^{(7)}$ – Choose a random $Y^\star \in \bar{\mathbb{G}}$. If $(*, Y^\star) \in \Lambda_S$, one aborts the game, otherwise adds the record $(j, Y^\star)$ to Λ_S.

►**Rule S2-**$^{(7)}$ – Check if $((U, X), (S, Y^\star), H) \in \Lambda_\Psi$. If this is not the case, then reject the authenticator: terminate, without accepting.

►**Rule S2+**$^{(7)}$ – Compute $Y = \mathcal{D}_{pw}(Y^\star)$, and lookup $(1, U\|S\|X\|Y\|*, H) \in \Lambda_\mathcal{A}$. If found, define Auth'_7 as true (and abort the game).

Fig. 7. Rewriting of some Rules in $\mathbf{G}_7$

From Equations (9), (10) and (11), one gets

$$\Pr[\mathsf{Encrypt}_7] \le \frac{q_s}{2N} \qquad \Pr[\mathsf{Auth}'_7] \le \frac{q_s}{2N} \qquad \Pr[\mathsf{AskH}_7] \le q_h \mathsf{Succ}^{\mathsf{cdh}}_{\mathbb{G}}(t'), \tag{13}$$

which concludes the proof. □

B Proof of Theorem 3

We can actually use the proof presented in Section 3.2, since

$$\mathsf{Adv}^{\mathsf{c-auth}}_{\mathsf{oeke}}(\mathcal{A}) = \Pr[\mathsf{Auth}_0],$$

and see that in game $\mathbf{G}_6$, $\Pr[\mathsf{Auth}_6] = 0$, and Equations (2), (3), (4), (5), (6), and (7) extends to

$$\begin{aligned}
|\Pr[\mathsf{Auth}_1] - \Pr[\mathsf{Auth}_0]| &\le \frac{q_{\mathcal{E}}^2}{2(q-1)} & |\Pr[\mathsf{Auth}_2] - \Pr[\mathsf{Auth}_1]| &\le \frac{q_S q_{\mathcal{E}}}{q-1} \\
|\Pr[\mathsf{Auth}_3] - \Pr[\mathsf{Auth}_2]| &\le \frac{2q_{\mathcal{E}}^2 + q_S^2}{2(q-1)} + \frac{q_h^2}{2^{\ell_1+1}} & |\Pr[\mathsf{Auth}_4] - \Pr[\mathsf{Auth}_3]| &\le \Pr[\mathsf{Encrypt}_4] \\
|\Pr[\mathsf{Auth}_5] - \Pr[\mathsf{Auth}_4]| &\le \frac{q_s}{2^{\ell_1}} & |\Pr[\mathsf{Auth}_6] - \Pr[\mathsf{Auth}_5]| &\le \Pr[\mathsf{Auth}'_6].
\end{aligned}$$

Then, using Equations (12) from the conclusion of the previous proof, and Equation (8), one gets,

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{c-auth}}_{\mathsf{oeke}}(\mathcal{A}) &\le \frac{q_{\mathcal{E}}^2}{2(q-1)} + \frac{q_S q_{\mathcal{E}}}{q-1} + \frac{2q_{\mathcal{E}}^2 + q_S^2}{2(q-1)} + \frac{q_h^2}{2^{\ell_1+1}} + \Pr[\mathsf{Encrypt}_4] + \frac{q_s}{2^{\ell_1}} + \Pr[\mathsf{Auth}'_6] \\
&\le \frac{(2q_{\mathcal{E}} + q_S)^2}{2(q-1)} + \frac{q_h^2 + 2q_s}{2^{\ell_1+1}} \\
&\quad + \left(\Pr[\mathsf{Encrypt}_7] + \frac{q_s}{2^{\ell_1}} + \Pr[\mathsf{Auth}'_7] + 2\Pr[\mathsf{AskH}_7]\right) \\
&\quad + \left(\Pr[\mathsf{Auth}'_7] + \Pr[\mathsf{AskH}_7]\right) \\
&\le \frac{(2q_{\mathcal{E}} + q_S)^2}{2(q-1)} + \frac{q_h^2 + 4q_s}{2^{\ell_1+1}} + \Pr[\mathsf{Encrypt}_7] + 2\Pr[\mathsf{Auth}'_7] + 3\Pr[\mathsf{AskH}_7],
\end{aligned}$$

which concludes the proof, using Equation (13). □

C Security Proof of **AuthA**

Proving the security of this new protocol follows the same path as the one in Section 3.2, until the Game $\mathbf{G}_8$:

Game $\mathbf{G}_8$***:*** In that game, we simulate the executions using the random self-reducibility of the Diffie-Hellman problem, given one Diffie-Hellman instance (A, B). We first choose a random element $\gamma \in \mathbb{Z}_q^\star$ and define $pw_S = A^\gamma$. We also add the record (γ, pw_S) to Λ_A.

- ►**Rule U1**$^{(8)}$ – Choose a random element $\alpha \in \mathbb{Z}_q^\star$, and compute $X = A^\alpha$. Also add the record (α, X) to Λ_A.

- ►**Rule** $\mathcal{D}^{(8)}$ – Choose a random element $\beta \in \mathbb{Z}_q^\star$, and compute the answer $Z = B^\beta$. Also add the record (β, Z) to Λ_B. If $(*, Z, *, *, Z^\star) \in \Lambda_{\mathcal{E}}$, one aborts the game, otherwise adds the record $(k, Z, \perp, \mathcal{D}, Z^\star)$ to $\Lambda_{\mathcal{E}}$.

$$\Pr[\mathsf{AskH}_8] = \Pr[\mathsf{AskH}_7]. \tag{14}$$

Remember that AskH_8 means that the adversary $\mathcal{A}$ queried the random oracles $\mathcal{H}_0$ or $\mathcal{H}_1$ on $U\|S\|X\|Y\|\mathsf{CDH}(X,Y)$, and thus $\mathcal{H}$ on $U\|S\|X\|Y\|\mathsf{CDH}(X,Y)$ or $*\|\mathsf{CDH}(pw_S,Y)$. By picking randomly in the $\Lambda_{\mathcal{A}}$-list, with probability $1/q_h$, we can get the Diffie-Hellman secret value. This is a triple $(X,Y,\mathsf{CDH}(X,Y))$. One then simply looks up into Λ_A and Λ_B to get α and β such that $X = A^\alpha$ and $Y = B^\beta$:

$$\mathsf{CDH}(X,Y) = \mathsf{CDH}(A^\alpha, B^\beta) = \mathsf{CDH}(A,B)^{\alpha\beta}.$$

Thus:

$$\Pr[\mathsf{AskH}_8] \le q_h \mathsf{Succ}^{\mathsf{cdh}}_{\mathbb{G}}(t'). \tag{15}$$

This concludes the proof. □

D Forward-Secrecy

The previous security results and proofs do not deal with forward-secrecy. Considering forward-secrecy requires to take into account a new kind of query that we call the $\mathsf{Corrupt}$-query (any other kinds of queries can still be asked, before but also after this one):

- $\mathsf{Corrupt}(I)$: This query models the attacks resulting in the password pw of this party I to be revealed. $\mathcal{A}$ gets back from its query pw but does not get any internal data of I.

Then we define a new flavor of freshness, saying that an instance is **Fresh** (or holds a **Fresh** key sk) if the following conditions hold. First, the instance has computed and accepted a session key. Second, no $\mathsf{Corrupt}$-query has been made by the adversary since the beginning of the game (before the session key is accepted). Third, neither it nor its partner have been asked for a Reveal-query.

This security level means that the adversary does not learn any information about *previously* established session keys when making a $\mathsf{Corrupt}$-query. We thus denote by $\mathsf{Adv}^{\mathsf{ake-fs}}_P(\mathcal{A})$ the advantage an adversary can get on a fresh key, in the protocol P, with the ability to make a $\mathsf{Corrupt}$-query.

Theorem 4 (AKE-FS Security). *Let us consider the* OEKE *protocol, where* **SK** *is the session-key space and* $\mathsf{Password}$ *a finite dictionary of size N equipped with the uniform distribution. Let $\mathcal{A}$ be an adversary against the AKE security of* OEKE *within a time bound t, with less than q_s interactions with the parties and q_p passive eavesdroppings, and, asking q_h hash-queries and q_e encryption/decryption queries. Then we have*

$$\mathsf{Adv}^{\mathsf{ake-fs}}_{\mathsf{oeke}}(\mathcal{A}) \le 3 \times \frac{q_s}{N} + 4q_h(1+(q_s+q_p)^2) \times \mathsf{Succ}^{\mathsf{cdh}}_{\mathbb{G}}(t') + \frac{(2q_e+3q_s+3q_p)^2}{q-1} + \frac{q_h^2+4q_s}{2^{\ell_1}}.$$

where $t' \le t + (q_s+q_p+q_e)\cdot\tau_{\mathbb{G}}$, with $\tau_{\mathbb{G}}$ denoting the computational time for an exponentiation in $\mathbb{G}$. (Recall that q is the order of $\mathbb{G}$.)

Proof. To deal with forward-secrecy, we define event $\mathsf{Corrupted}$ as the event that $\mathcal{A}$ asks a $\mathsf{Corrupt}$-query, and we refine events $\mathsf{Encrypt}$, Auth, Auth' and AskH respectively into $\mathsf{EncryptBC}$, AuthBC, AuthBC' and AskHBC respectively:

$$\begin{aligned} \mathsf{EncryptBC}_k &:= \mathsf{Encrypt}_k \prec \mathsf{Corrupted} \qquad & \mathsf{AuthBC}_k &:= \mathsf{Auth}_k \prec \mathsf{Corrupted} \\ \mathsf{AuthBC}'_k &:= \mathsf{Auth}'_k \prec \mathsf{Corrupted} \qquad & \mathsf{AskHBC}'_k &:= \mathsf{AskH}_k \prec \mathsf{Corrupted} \end{aligned}$$

that is $\mathsf{EncryptBC}_k$, AuthBC_k, AuthBC'_k or AskHBC_k respectively occur if $\mathsf{Encrypt}_k$, Auth_k, Auth'_k or AskH_k respectively occur **before** corrupting a player.

We can base the proof on a similar sequence of games as before, but just modifying some rules before any corruption:

►**Rule S2**$^{(6)}$ – If $(X, Y^\star, H) \notin \Lambda_\Psi$, and either $\mathsf{Corrupted} = \mathsf{false}$ or ($\mathsf{Corrupted} = \mathsf{true}$ and $(1, U\|S\|X\|Y\|K_S, H) \notin \Lambda_\mathcal{A}$), then reject the authenticator: terminate, without accepting. Moreover, if $\mathsf{Corrupted} = \mathsf{false}$ and $(1, U\|S\|X\|Y\|*, H) \in \Lambda_\mathcal{A}$ we define the event AuthBC'_6 to be true, and abort the game.

►**Rule U3**$^{(7)}$ – If $\mathsf{Corrupted} = \mathsf{false}$, then compute the session key $sk_U = \mathcal{H}_2(U\|S\|X\|Y^\star)$ and the authenticator $Auth = \mathcal{H}_3(U\|S\|X\|Y^\star)$. Otherwise, compute the session key $sk_U = \mathcal{H}_0(U\|S\|X\|Y\|K_U)$ and the authenticator $Auth = \mathcal{H}_1(U\|S\|X\|Y\|K_U)$.

►**Rule S3**$^{(7)}$ – If $\mathsf{Corrupted} = \mathsf{false}$, then compute the session key $sk_S = \mathcal{H}_2(U\|S\|X\|Y^\star)$. Otherwise, compute the session key $sk_S = \mathcal{H}_0(U\|S\|X\|Y\|K_S)$.

►**Rule U2**$^{(7)}$ – Lookup $(pw, *, \perp, \mathcal{E}, Y^\star) \in \Lambda_\mathcal{E}$. If found, define $\mathsf{Encrypt}_7$ as true and abort the game. Otherwise, compute $Y = \mathcal{D}_{pw}(Y^\star)$. If $\mathsf{Corrupted} = \mathsf{false}$, furthermore define $K_U = Y^\theta$.

►**Rule S1**$^{(7)}$ – Choose a random $Y^\star \in \bar{\mathbb{G}}$. If $(*, Y^\star) \in \Lambda_S$, one aborts the game, otherwise adds the record $(j, Y^\star)$ to Λ_S. Then, compute $Y = \mathcal{D}_{pw}(Y^\star)$. If $\mathsf{Corrupted} = \mathsf{false}$, furthermore lookup $(pw, Y, \varphi, *, Y^\star) \in \Lambda_\mathcal{E}$ to define φ (we thus have $Y = g^\varphi$), and compute $K_S = X^\varphi$.

By evaluating the events $\mathsf{Encrypt}_7$ and Auth_7 at the corruption time, one gets as before

$$|\Pr[\mathsf{S}_6] - \Pr[\mathsf{S}_0]| \le \frac{(2q_\mathcal{E} + q_S)^2}{2(q-1)} + \frac{q_h^2}{2^{\ell_1+1}} + \Pr[\mathsf{EncryptBC}_4] + \frac{q_s}{2^{\ell_1}} + \Pr[\mathsf{AuthBC}'_6],$$

$$\Pr[\mathsf{EncryptBC}_4] \le \frac{q_s}{N} + \frac{q_s}{2^{\ell_1}} + q_h \mathsf{Succ}^{\mathsf{cdh}}_{\mathbb{G}}(t') \qquad \Pr[\mathsf{AuthBC}'_6] \le \frac{q_s}{2N} + q_h \mathsf{Succ}^{\mathsf{cdh}}_{\mathbb{G}}(t').$$

As a consequence,

$$|\Pr[\mathsf{S}_6] - \Pr[\mathsf{S}_0]| \le \frac{3q_s}{2N} + 2q_h \times \mathsf{Succ}^{\mathsf{cdh}}_{\mathbb{G}}(t') + \frac{(2q_\mathcal{E} + q_S)^2}{2(q-1)} + \frac{q_h^2}{2^{\ell_1+1}} + \frac{2q_s}{2^{\ell_1}}. \tag{16}$$

We now go back the game $\mathbf{G}_6$, as presented on Figure 8. We furthermore abort the game where the events $\mathsf{EncryptBC}_6$ or AuthBC'_6 happen to be true.

Game* $\mathbf{G}_7$*: We now have to make a different analysis: we need to know the private exponents of (almost) all the instances of the parties, since the adversary may send the authenticator after making the $\mathsf{Corrupt}$-query, and thus knowing the password. Otherwise, a later Reveal-query would not be perfect. Therefore, one first bets on an execution (passive or active) to be tested: one chooses a random index $\mu \in \{1, \ldots, q_s + q_p\}$ and a random index $\nu \in \{1, \ldots, q_s + q_p\}$. If the Test-query does not correspond to the client involved in the μ-th Send-query, and the server involved in the ν-th Send-query, then one aborts the game, outputting a random bit b'. Since the Test-query can only be asked to an instance that has accepted before any corruption and that only simulated keys can be asked,

$$\Pr[\mathsf{S}_7] = \frac{1}{(q_s + q_p)^2} \times \Pr[\mathsf{S}_6] + \left(1 - \frac{1}{(q_s + q_p)^2}\right) \times \frac{1}{2}.$$

Then,

$$\left|\Pr[\mathsf{S}_6] - \frac{1}{2}\right| = (q_s + q_p)^2 \times \left|\Pr[\mathsf{S}_7] - \frac{1}{2}\right|. \tag{17}$$

Game* $\mathbf{G}_8$*: We now inject a CDH instance into this specific execution: we are given (A, B), with the discrete logarithms a and b

►**Rule U1**$^{(8)}$ – If this corresponds to the μ-th instance of the client, set $\theta = a$, otherwise, choose a random element $\theta \in \mathbb{Z}_q^\star$. Then compute $X = g^\theta$.

We answer to the Send-queries to the client as follows:

- A $\mathsf{Send}(U^i, \mathtt{Start})$-query is processed according to the following rule:
  ►**Rule U1**$^{(6)}$ – Choose a random exponent $\theta \in \mathbb{Z}_q^\star$ and compute $X = g^\theta$.
  Then the query is answered with U, X, and the client instance goes to an expecting state.
- If the client instance U^i is in an expecting state, a query $\mathsf{Send}(U^i, (S, Y^\star))$ is processed by computing the session key and producing an authenticator. We apply the following rules:
  ►**Rule U2**$^{(6)}$ – Lookup $(pw, *, \perp, \mathcal{E}, Y^\star) \in \Lambda_\mathcal{E}$. If found, define $\mathsf{Encrypt}_6$ as true. Otherwise, compute $Y = \mathcal{D}_{pw}(Y^\star)$. Furthermore define $K_U = Y^\theta$.
  ►**Rule U3**$^{(6)}$ – Compute the session key $sk_U = \mathcal{H}_0(U\|S\|X\|Y\|K_U)$ and the authenticator $Auth = \mathcal{H}_1(U\|S\|X\|Y\|K_U)$.
  Finally the query is answered with $Auth$, the client instance accepts and terminates. Our simulation also adds $(X, Y^\star, Auth)$ to Λ_Ψ.

We answer to the Send-queries to the server as follows:

- A $\mathsf{Send}(S^j, (U, X))$-query is processed according to the following rule:
  ►**Rule S1**$^{(6)}$ – Choose a random $Y^\star \in \bar{\mathbb{G}}$. If $(*, Y^\star) \in \Lambda_S$, one aborts the game, otherwise adds the record $(j, Y^\star)$ to Λ_S. Then, compute $Y = \mathcal{D}_{pw}(Y^\star)$, lookup $(pw, Y, \varphi, *, Y^\star) \in \Lambda_\mathcal{E}$ to define φ (we thus have $Y = g^\varphi$), and compute $K_S = X^\varphi$.
  Finally, the query is answered with $S, Y^\star$ and the server instance goes to an expecting state.
- If the server instance S^j is in an expecting state, a query $\mathsf{Send}(S^j, H)$ is processed according to the following rules:
  ►**Rule S2**$^{(6)}$ – If $(X, Y^\star, H) \notin \Lambda_\Psi$, and either $\mathsf{Corrupted} = \mathsf{false}$ or ($\mathsf{Corrupted} = \mathsf{true}$ and $(1, U\|S\|X\|Y\|K_S, H) \notin \Lambda_\mathcal{A}$), then reject the authenticator: terminate, without accepting. Moreover, if $(1, U\|S\|X\|Y\|*, H) \in \Lambda_\mathcal{A}$ we define the event Auth'_6 to be true.
  If the server instance has not terminated, it accepts and goes on, applying the following rule:
  ►**Rule S3**$^{(6)}$ – Compute the session key $sk_S = \mathcal{H}_0(U\|S\|X\|Y\|K_S)$.
  Finally, the server instance terminates.

Fig. 8. Simulation of the Send-queries in $\mathbf{G}_6$

►**Rule** $\mathcal{D}^{(8)}$ – If this corresponds to the ν-th instance of the server, set $\varphi = b$, otherwise choose a random element $\varphi \in \mathbb{Z}_q^\star$. Then compute $Z = B^\varphi$. If $(*, Z, *, *, Z^\star) \in \Lambda_\mathcal{E}$, one aborts the game. One finally adds the record $(k, Z, \varphi, \mathcal{D}, Z^\star)$ to $\Lambda_\mathcal{E}$.

The games $\mathbf{G}_8$ and $\mathbf{G}_7$ are perfectly indistinguishable:

$$\Pr[\mathsf{S}_7] = \Pr[\mathsf{S}_8]. \tag{18}$$

Game $\mathbf{G}_9$***:*** In that game, the session key and the authenticator of this specific execution of the protocol is defined using private random oracles $\mathcal{H}_2$ and $\mathcal{H}_3$, independent from $\mathcal{H}_0$ and $\mathcal{H}_1$. For that, we modify the following rules:

►**Rule U2**$^{(9)}$ – Lookup $(pw, *, \perp, \mathcal{E}, Y^\star) \in \Lambda_\mathcal{E}$. If found, define $\mathsf{Encrypt}_9$ as true. If this does not correspond to the μ-th instance of the client, one computes $Y = \mathcal{D}_{pw}(Y^\star)$ and defines $K_U = Y^\theta$ (otherwise we won't need it).

►**Rule U3**$^{(9)}$ – If this corresponds to the μ-th instance of the client, one computes the session key $sk_U = \mathcal{H}_2(U\|S\|X\|Y^\star)$ and the authenticator $Auth = \mathcal{H}_3(U\|S\|X\|Y^\star)$. Otherwise, compute the session key $sk_U = \mathcal{H}_0(U\|S\|X\|Y\|K_U)$ and the authenticator $Auth = \mathcal{H}_1(U\|S\|X\|Y\|K_U)$.

►**Rule S1**$^{(9)}$ – Choose a random $Y^\star \in \bar{\mathbb{G}}$. If $(*, Y^\star) \in \Lambda_S$, one aborts the game, otherwise adds the record $(j, Y^\star)$ to Λ_S. If this does not correspond to the ν-th instance of the server, one computes $Y = \mathcal{D}_{pw}(Y^\star)$, looks up $(pw, Y, \varphi, *, Y^\star) \in \Lambda_\mathcal{E}$ to define φ (we thus have $Y = g^\varphi$), and computes $K_S = X^\varphi$ (otherwise we won't need it).

▶**Rule S3**$^{(9)}$ – If this corresponds to the ν-th instance of the server, one computes the session key $sk_S = \mathcal{H}_2(U\|S\|X\|Y^\star)$. and the authenticator $Auth = \mathcal{H}_3(U\|S\|X\|Y^\star)$. Otherwise, compute the session key $sk_U = \mathcal{H}_0(U\|S\|X\|Y\|K_S)$ and the authenticator $Auth = \mathcal{H}_1(U\|S\|X\|Y\|K_S)$.

The games $\mathbf{G}_9$ and $\mathbf{G}_8$ are indistinguishable unless the following event AskH_9 occurs: $\mathcal{A}$ queries the hash functions $\mathcal{H}_0$ or $\mathcal{H}_1$ on $U\|S\|X\|Y\|\mathsf{CDH}(X,Y)$:

$$|\Pr[\mathsf{S}_9] - \Pr[\mathsf{S}_8]\,| \le \Pr[\mathsf{AskH}_9]. \tag{19}$$

Game $\mathbf{G}_{10}$***:*** Now, we are not given the discrete logarithms a and b anymore:

▶**Rule U1**$^{(10)}$ – If this corresponds to the μ-th instance of the client, set $X = A$, otherwise, choose a random element $\theta \in \mathbb{Z}_q^\star$ and compute $X = g^\theta$.

▶**Rule** $\mathcal{D}^{(10)}$ – If this corresponds to the ν-th instance of the server, set $Z = B$ and $\varphi = \perp$, otherwise choose a random element $\varphi \in \mathbb{Z}_q^\star$ and compute $Z = B^\varphi$. Finally, if $(*, Z, *, *, Z^\star) \in \Lambda_\mathcal{E}$, one aborts the game. One then adds the record $(k, Z, \varphi, \mathcal{D}, Z^\star)$ to $\Lambda_\mathcal{E}$.

Since K_U and K_S are not required for this execution of the protocol (the session key and the authenticator are defined using independent private random oracles on X and $Y^\star$ only), the two games are indistinguishable:

$$\Pr[\mathsf{S}_9] = \Pr[\mathsf{S}_{10}] \qquad \Pr[\mathsf{AskH}_9] = \Pr[\mathsf{AskH}_{10}]. \tag{20}$$

Furthermore, it is now clear that

$$\Pr[\mathsf{AskH}_{10}] = q_h \times \mathsf{Succ}^{\mathsf{cdh}}_{\mathbb{G}}(t'). \tag{21}$$

As a conclusion, from the Equations (16), (17), (18), (19), (20) and (21),

$$\left| \Pr[\mathsf{S}_6] - \frac{1}{2} \right| \le 2(q_s + q_p)^2 \times \Pr[\mathsf{AskH}_9] \le 2(q_s + q_p)^2 q_h \times \mathsf{Succ}^{\mathsf{cdh}}_{\mathbb{G}}(t').$$

This security result can definitely be improved using the random self-reducibility of the Diffie-Hellman problem. Namely, one could remove the factor $(q_s + q_p)^2$, but this would make the reduction much more intricate. □