Group Diffie-Hellman Key Exchange Secure Against Dictionary Attacks

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

[1] École normale supérieure, 75230 Paris Cedex 05, France
http://www.di.ens.fr/∼{bresson,pointche},
{Emmanuel.Bresson,David.Pointcheval}@ens.fr.
[2] Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA,
http://www.itg.lbl.gov/∼chevassu, OChevassut@lbl.gov.

Abstract. Group Diffie-Hellman schemes for password-based key exchange are designed to provide a pool of players communicating over a public network, and sharing just a human-memorable password, with a session key (e.g, the key is used for multicast data integrity and confidentiality). The fundamental security goal to achieve in this scenario is security against dictionary attacks. While solutions have been proposed to solve this problem no formal treatment has ever been suggested. In this paper, we define a security model and then present a protocol with its security proof in both the random oracle model and the ideal-cipher model.

1 Introduction

Group Diffie-Hellman schemes for password-based key exchange are designed to provide a pool of players, communicating over a public network, and holding a shared human-memorable password with a session key to be used to implement secure multicast sessions. A human-memorable password *pw* is a (short) string chosen from a relatively small dictionary to be easily memorized and typed-in by a human.

Consider mission-critical applications such as emergency rescue and military operations [18, 19, 21], or even commercial applications like conferencing/meeting [1, 19] and personal networking [5, 13], where a (small) group of people collaborate. These applications operate in a highly mobile environment characterized by the lack of any fixed network and security infrastructure. At the same time, these are applications where secure multicast sessions may be needed. Due to the absence of fixed infrastructure, session keys can be computed via a group Diffie-Hellman key exchange bootstrapped from a password. A password usually chosen by the participants may be a low-quality one (i.e. 4 decimal digits) easier to memorize than a high-quality one (i.e. 56-bit, 192-bit).

The fundamental security goal for a group Diffie-Hellman protocol designed for such a scenario to achieve is security against dictionary attacks. One can not actually prevent the adversary from guessing a value for *pw* 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 values for pw. 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 pw should not get any bias on the actual password – such a bias could be later used in on-line interactions. The off-line exhaustive search is called *dictionary attack*.

Contributions. This paper represents the first formal treatment of the authenticated group Diffie-Hellman key exchange problem when the parties share a human-memorable password. We start from the model of Bresson et al. [10] and enhance it to capture dictionary attacks. In our model, the parties are modeled through oracles and the various types of attacks are modeled by queries to these oracles. The model is equipped with the ability to obtain honest protocol executions to enable a treatment of dictionary attacks.

Our model is used to define the execution of a password-based group Diffie-Hellman protocol which we refer to as EKE (*Encrypted Key Exchange*, see [3]). Converting a provably authenticated group Diffie-Hellman protocol [10] into a password-based group Diffie-Hellman protocol is not an easy task. The trivial conversion consisting in substituting a signature scheme by a symmetric encryption scheme, using the password as secret key as for the two-party case [2, 7], does not provide security against dictionary attacks. We have, in effect, to perform several modifications to the protocol of Bresson et al. [10]. The modifications cost only one more exponentiation per player however we also notice that the cost of the signatures and verifications is replaced by the cost of a symmetric encryption, which is very low. The flows are moreover shorter since there is no longer a signature.

The security against dictionary attacks shows up in Theorem 1 which asserts the security of EKE in both the random oracle model and the ideal-cipher model. Security against dictionary attacks depends on how many interactions the adversary carries out against the instances rather than on the adversary's computational power. The theorem exhibits a reduction from the semantic security of an EKE session key to reasonable and well-defined computational problems.

Our paper is organized as follows. In the remainder of this section we summarize the related work. In Section 2, we define our model and the definitions that should be satisfied by a group Diffie-Hellman scheme secure against dictionary attacks. In Section 3, we present the intractability assumptions we use in this paper. We present the EKE protocol in Section 4 and assert its security in both the random oracle model and the ideal-cipher model in Section 5. We then prove its security in Section 6. Finally, some extensions are provided: we briefly deal with forward-secrecy in Section 7 and with mutual authentication in Section 8.

Related Work. Several 2-party Diffie-Hellman key exchange protocols aimed to distribute a session key among two parties when the parties share a password. Recently, Bellare et al. [2] presented a formal model for this problem and a protocol secure in the ideal-cipher model. Our work extends their work to the multi-party setting. Security proofs in the ideal-cipher model see a (keyed) cipher as a family of random permutations which are queried via an oracle to encrypt and decrypt. The oracle produces a truly random value for each new query and identical answers if the same query is asked twice; furthermore, for each key, the injectivity is satisfied. In practice, the ideal-cipher [4] is instantiated using deterministic symmetric encryption function such as AES [17]. Although these encryption functions have been designed with different criteria from being an ideal-cipher, AES has been designed with unpredictability in mind.

Security proofs in these two models together (both the random oracle and the ideal-cipher models) are superior to those provided by *ad-hoc* protocol designs although they do not provide the same security guarantees as those in the random oracle and the standard models. However, the ideal-cipher model allows for "elegant" and more efficient protocols. Boyko et al. [7, 15] provided (2-party) Diffie-Hellman key exchange protocols proved secure in the random oracle model using the multi-party simulatability technique. Katz et al. [14], and Goldreich et al. [12] designed two-party key exchange protocols secure in the standard model.

Several papers have extended the Diffie-Hellman protocol [11] to the multi-party setting and thus aimed to distribute a session key among parties aggregated into a group. Bresson et al. [10] presented a formal model to securely design protocols for a scenario wherein each party holds a pair of matching public/private keys. A logical follow up to this work is a formal model for a scenario wherein the parties share a human-memorable password. This latter scenario was suggested by Asokan et al. [1] as well as protocols with informal security analysis.

2 Model

In this section we define a formal model for security against dictionary attacks where the adversary's capabilities are modeled through queries. In our model, the players do not deviate from the protocol and the adversary is not a player. We define the security notion that a password-based group Diffie-Hellman protocol should achieve. In Authenticated Key Exchange (with implicit authentication), each player is assured that an adversary not in the group is unable to learn any information about the session key. Another important notion is mutual authentication, which guarantees to each player that it actually shares a session key with all the others.

2.1 Security Model

Players. We fix a nonempty set $\mathcal{U}$ of players that can participate in a group Diffie-Hellman key exchange protocol P. A player $U_i \in \mathcal{U}$ may have many *instances* called oracles involved in distinct, but possibly concurrent, executions of P. We denote by Π_i^t the t-th instance of player U_i, for any $t \in \mathbb{N}$.

The players share a low-entropy secret pw taken from a small dictionary $\mathsf{Password}$ of size N. In the following, we assume that this password pw follows a uniform distribution in the $\mathsf{Password}$ set.

Abstract Interface. Let us define the basic structure of a password-based group Diffie-Hellman protocol P. The protocol consists of two algorithms:

- The *password generation* algorithm $\textsc{PwdGen}(1^\ell)$ is a probabilistic algorithm which, on input a security parameter 1^ℓ, provides each player in $\mathcal{U}$ with a common password pw uniformly distributed in $\mathsf{Password}$.
- The *key exchange* algorithm $\textsc{KeyExch}(\mathcal{U})$ is an interactive multi-party protocol providing the instances of players in $\mathcal{U}$, holding a common password, with a session key sk.

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

- $\mathsf{Execute}(\mathcal{U})$: This query models passive attacks, where the adversary gets access to honest executions of P by eavesdropping. Therefore, $\mathcal{A}$ gets back the protocol flows of an honest execution of P between the players in $\mathcal{U}$.
- $\mathsf{Send}(\Pi_i^t, m)$: This query models $\mathcal{A}$ sending a message to an instance. The adversary $\mathcal{A}$ gets back the response oracle Π_i^t generates in processing the message m according to the protocol P. A query $\mathsf{Send}(\Pi_1^t$, "$\texttt{Start}$") initializes the key exchange algorithm, and thus the adversary receives the flow the first player should send out to the second one.
- $\mathsf{Reveal}(\Pi_i^t)$: This query models the misuse of the session key by the players. The query is only available to $\mathcal{A}$ if oracle Π_i^t holds a session key. The Reveal-query unconditionally forces oracle Π_i^t to release $sk_{\Pi_i^t}$ which is otherwise hidden to $\mathcal{A}$.
- $\mathsf{Test}(\Pi_i^t)$: This query models the semantic security of the session key sk. The Test-query can be asked at most once by the adversary $\mathcal{A}$ and is only available to $\mathcal{A}$ if Π_i^t is **Fresh** (see below). This query is answered as follows: one flips a coin b and forwards $\mathsf{Reveal}(\Pi_i^t)$ if $b = 1$ or a random value if $b = 0$.

The $\mathsf{Execute}$-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 $\mathsf{Execute}$-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 $\mathsf{Execute}$-queries. Therefore, q_s represents the number of flows the adversary may have built by himself, and thus the number of passwords he would have tried.

The security notions take place in the context of executing P in the presence of the adversary $\mathcal{A}$. In this game $\mathbf{Game}^{\mathsf{ake}}(\mathcal{A}, P)$, $\mathcal{A}$ plays against the players using the above queries in order to defeat the security of P. The game is initialized by providing coin tosses to $\textsc{PwdGen}$, $\mathcal{A}$, all Π_i^t, and then

1. $\textsc{PwdGen}$ is run to set the value pw of the password,
2. Initialize any Π_i^t with $sk_{\Pi_i^t} \leftarrow \textsc{null}$,
3. Initialize adversary $\mathcal{A}$ with 1^ℓ and access to all Π_i^t,
4. Run adversary $\mathcal{A}$ and answer queries made by $\mathcal{A}$,
5. At the end of the game, $\mathcal{A}$ outputs its guess b' for the bit b involved in the Test-query.

2.2 Security Notions

Freshness. An oracle Π_i^t is **Fresh** (or holds a **Fresh** key sk) if Π_i^t has computed a session key $sk \neq$ NULL and neither Π_i^t nor one of its partners has been asked for a Reveal-query. Intuitively, the partners of an instance Π_i^t are all the instances that "should" hold the same session key as Π_i^t at the end of the protocol. We give a more formal definition in the full version of this paper [8].

AKE Security. In an execution of P, we say an adversary $\mathcal{A}$ *wins* if it asks a single Test-query to a **Fresh** player U and correctly guesses the bit b used in the game $\mathbf{Game}^{\mathsf{ake}}(\mathcal{A}, P)$. We denote the **AKE advantage** as $\mathsf{Adv}_P^{\mathsf{ake}}(\mathcal{A}) = 2\Pr[b = b'] - 1$, where the probability space is over all the random coins of the adversary and all the oracles.

3 Assumptions

Before presenting the protocol, let us remind the algorithmic assumptions on which its security will be based on. These assumptions were shown in [9] to be reasonable by relating them to the DDH and CDH.

Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of prime order q and $n \in \mathbb{N}$. Let I_n be $\{1, \ldots, n\}$, $\mathcal{P}(I_n)$ be the set of all subsets of I_n and Γ be any subset of $\mathcal{P}(I_n)$. We define the *Group Diffie-Hellman distribution* relative to Γ as:

$$\mathsf{GDH}_\Gamma = \{\mathcal{D}_\Gamma(x_1, \ldots, x_n) \mid x_1, \ldots, x_n \in_R \mathbb{Z}_q\},$$

where

$$\mathcal{D}_\Gamma(x_1, \ldots, x_n) = \left\{ \left(J, g^{\prod_{j \in J} x_j}\right) \middle| \, J \in \Gamma \right\}.$$

Our protocol in this paper is based on the triangular structure $\mathcal{T}_n$ for Γ, we illustrate for $n = 4$ on Figure 1:

$$\begin{aligned} \mathcal{T}_n &= \bigcup_{2 \le j \le n} \{\, \{i \,|\, 1 \le i \le j, i \neq k\} \mid 1 \le k \le j\} \\ &= \big\{\{\}; \{2\}, \{1\}; \{2,3\}, \{1,3\}, \{1,2\}; \{2,3,4\}, \{1,3,4\}, \{1,2,4\}, \{1,2,3\}; \ldots \big\}. \end{aligned}$$

g			
g^{x_2}	g^{x_1}		
$g^{x_2 x_3}$	$g^{x_1 x_3}$	$g^{x_1 x_2}$	
$g^{x_2 x_3 x_4}$	$g^{x_1 x_3 x_4}$	$g^{x_1 x_2 x_4}$	$g^{x_1 x_2 x_3}$

Fig. 1. Trigon defined by $\mathcal{T}_n$ when $n = 4$.

Trigon Group Computational Diffie-Hellman Assumption (TG-CDH). A (T, ε)-$\mathsf{TG\text{-}CDH}_n$-attacker for $\mathbb{G}$ is a probabilistic Turing machine Δ running in time T that given $\mathcal{D} = \mathcal{D}_{\mathcal{T}_n}(x_1, \ldots, x_n) \in \mathsf{GDH}_{\mathcal{T}_n}$ outputs $g^{x_1 \cdots x_n}$ with probability greater than ε. We denote this success probability $\mathsf{Succ}_{\mathbb{G}}^{\mathsf{tgcdh}_n}(\Delta)$.

Multi Decisional Diffie-Hellman Assumption (M-DDH). In the analysis of the protocol EKE we need an equivalent version of the DDH assumption. Let us define the two following distributions:

$$\begin{aligned}
\mathsf{M\text{-}DH}_n &= \{(g^{x_1}, \ldots, g^{x_n}, g^{rx_1}, \ldots, g^{rx_n}) \,|\, x_1, \ldots, x_n, r \in_R \mathbb{Z}_q\}\,, \\
\mathsf{Rand}_n &= \{(g^{x_1}, \ldots, g^{x_n}, g^{y_1}, \ldots, g^{y_n}) \,|\, x_1, \ldots, x_n, y_1, \ldots, y_n \in_R \mathbb{Z}_q\}\,.
\end{aligned}$$

A $(T, \varepsilon) - \mathsf{M\text{-}DDH}_n$-distinguisher for $\mathbb{G}$ is a probabilistic Turing machine Δ running in time T that is able to distinguish the two distributions with advantage $\mathsf{Adv}_{\mathbb{G}}^{\mathsf{mddh}_n}(\Delta)$ greater than ε.

Lemma 1. *For any group $\mathbb{G}$ and any integer n, $\mathsf{Adv}_{\mathbb{G}}^{\mathsf{mddh}_n}(T) \le (n-1)\,\mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(T)$ and $\mathsf{Adv}_{\mathbb{G}}^{\mathsf{mddh}_n}(T) \le \mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(T + (4n-6)\tau_{\mathbb{G}})$, where $\tau_{\mathbb{G}}$ is the computational time for an exponentiation in $\mathbb{G}$.*

Proof. The first result easily comes using a hybrid argument [16], while the second one uses the random self-reducibility. Indeed, from a decisional Diffie-Hellman instance $(g^{x_2}, g^{r_1}, g^{r_2 x_2})$, where $r_2 = r_1$, one derives (with 2 exponentiations performed by raising two values to the power of x_1) a 4-tuple $(A_1 = g^{x_1}, A_2 = g^{x_2}, B_1 = g^{r_1 x_1}, B_2 = g^{r_2 x_2})$. Then, one easily gets a $2n$-tuple $(A_1 = g^{x_1}, \ldots, A_n = g^{x_n}, B_1 = g^{r_1 x_1}, \ldots, B_n = g^{r_n x_n})$ where either all the r_i are equal (if $r_2 = r_1$), or the r_i are independent from one another (if $r_2 \neq r_1$). To this aim, one chooses a random pair (u_i, v_i), and computes

$$\begin{aligned}
A_i &= A_1^{u_i} A_2^{v_i} = g^{x_1 u_i + x_2 v_i} = g^{x_i} \\
B_i &= B_1^{u_i} B_2^{v_i} = g^{r_1 x_1 u_i + r_2 x_2 v_i} = g^{r_1 x_i + (r_2 - r_1) x_2 v_i}.
\end{aligned}$$

□

4 A Password-based Group Diffie-Hellman Protocol

In the following theorems and proofs we assume both the random oracle model and the ideal-cipher model, and the arithmetic is in a finite cyclic group $\mathbb{G} = \langle g \rangle$ of order a ℓ_1-bit prime number q, where the operation is denoted multiplicatively. More precisely, we consider $\bar{\mathbb{G}} = \mathbb{G} \backslash \{1\}$. It has the particularity that for any $h \in \bar{\mathbb{G}}$, $\bar{\mathbb{G}} = \{h^r \,|\, r \in \{1, \ldots, q-1\}\}$, but it is no longer a group.

We then use a hash function $\mathcal{H}$ from $\{0,1\}^*$ to $\{0,1\}^{\ell_2}$ and consider several block ciphers, depending on the size of the input: for each integer $i \ge 2$, we define two families $\mathcal{E}^i = \{\mathcal{E}_k^i\}$ and $\mathcal{E}'^i = \{\mathcal{E}'^i_k\}$ of keyed permutations over $\bar{\mathbb{G}}^i$, where $k \in \mathsf{Password}$. The inverse of $\mathcal{E}_k^i$ (resp. $\mathcal{E}'^i_k$) is denoted $\mathcal{D}_k^i$ (resp. $\mathcal{D}'^i_k$).

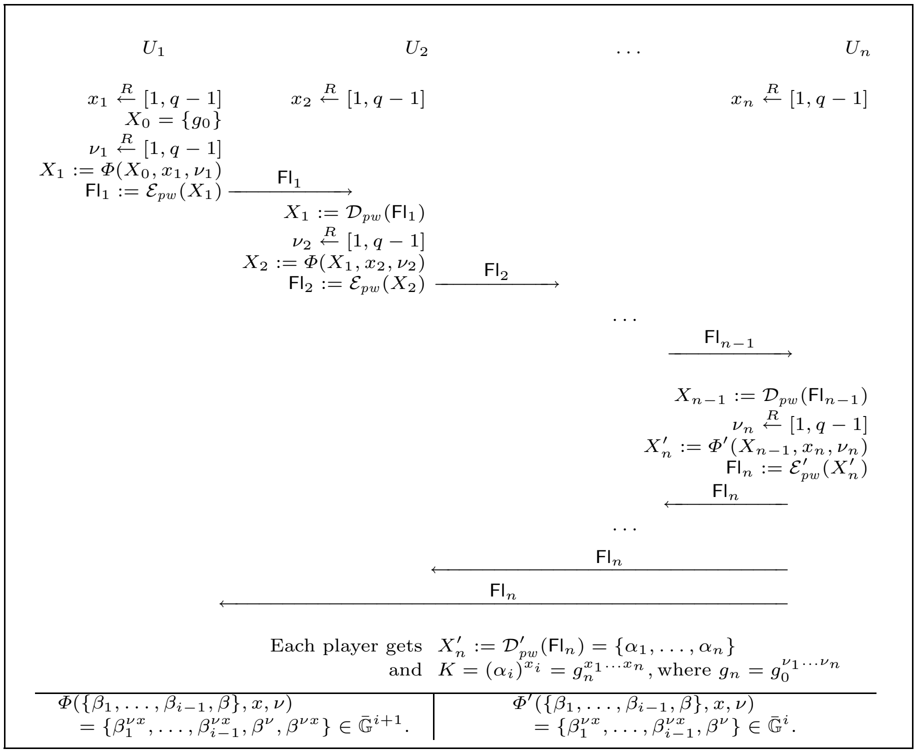

Fig. 2. Protocol EKE. The multicast group is $\mathcal{U} = \{U_1, U_2, \dots, U_n\}$ and the session key is $sk = \mathcal{H}(\mathcal{U} \| \mathsf{Fl}_n \| K)$.

In practice, such encryption schemes are instantiated with CBC mode so that each part of the plaintext depends on the entire ciphertext. Following this idea, we (abusively) denote $\mathcal{E}_k(X)$ (resp. $\mathcal{E}'_k(X)$) the encryption of a plaintext $X \in \bar{\mathbb{G}}^i$ for some i under key k using $\mathcal{E}^i_k$ (resp. ${\mathcal{E}'}^i_k$) without explicitly specifying the length of X.

4.1 Algorithm

As illustrated on Figure 2, the protocol EKE consists of a set of players arranged in a ring and the flows are encrypted under the password pw. The session-key space **SK** associated to this protocol is $\{0,1\}^{\ell_2}$ equipped with a uniform distribution. Moreover, EKE consists of two stages: several up-flows (which are encrypted using $\mathcal{E}$) and the down-flow (which is encrypted using $\mathcal{E}'$).

In the up-flow, player U_i (for $1 \le i < n$) receives a ciphertext $\mathsf{Fl}_{i-1} \in \bar{\mathbb{G}}^i$ and decrypts it using $\mathcal{D}_{pw}$ into the plaintext $X_{i-1} \in \bar{\mathbb{G}}^i$ (by convention, U_1 just receives Fl_0 = "`Start`", and thus builds $X_0 = \{g_0\}$, where g_0 is a random element in $\bar{\mathbb{G}}$). Player U_i then generates at random two (private) values (x_i, ν_i) in $\mathbb{Z}_q^\star$ and gets $X_i := \Phi(X_{i-1}, x_i, \nu_i) \in \bar{\mathbb{G}}^{i+1}$ by processing the plaintext X_{i-1}

according to the operator Φ (described below). Player U_i finally encrypts the value X_i using $\mathcal{E}_{pw}$ and forwards the ciphertext Fl_i to the next player in the ring.

The down-flow takes place when player U_n receives the last up-flow $\mathsf{Fl}_{n-1} \in \bar{\mathbb{G}}^n$. It decrypts it using $\mathcal{D}_{pw}$ into the plaintext $X_{n-1} \in \bar{\mathbb{G}}^n$. It then generates at random two (private) values (x_n, ν_n) in $\mathbb{Z}_q^\star$ and gets $X'_n := \Phi'(X_{n-1}, x_n, \nu_n) \in \bar{\mathbb{G}}^n$ by processing the plaintext X_{n-1} according to the operator Φ' (described below). Player U_n finally encrypts the value X'_n using $\mathcal{E}'_{pw}$ and broadcasts the ciphertext Fl_n.

Finally, each player can compute the session key $sk = \mathcal{H}(\mathcal{U}\|\mathsf{Fl}_n\|K)$, where $K = (g_0^{\nu_1\cdots\nu_n})^{x_1\cdots x_n}$. Indeed, if everything worked correctly, player U_i can compute K by decrypting the broadcast Fl_n using $\mathcal{D}'_{pw}$ into $X'_n \in \bar{\mathbb{G}}^n$ and raising the i-th term α_i of X'_n to the power of its private exponent x_i.

4.2 Operators Φ and Φ'

We now describe the operators Φ and Φ', and see that finally all the players agree on the same value K. The operator Φ takes as inputs a set $\{\beta_1, \ldots, \beta_{i-1}, \beta\} \in \bar{\mathbb{G}}^i$, for some i, a private exponent $x \in \mathbb{Z}_q^\star$ and a blinding exponent $\nu \in \mathbb{Z}_q^\star$. Then,

$$\Phi(\{\beta_1, \ldots, \beta_{i-1}, \beta\}, x, \nu) = \{\beta_1^{\nu x}, \ldots, \beta_{i-1}^{\nu x}, \beta^\nu, \beta^{\nu x}\} \in \bar{\mathbb{G}}^{i+1}.$$

The operator Φ' does exactly the same transformation but returns the i first elements only:

$$\Phi'(\{\beta_1, \ldots, \beta_{i-1}, \beta\}, x, \nu) = \{\beta_1^{\nu x}, \ldots, \beta_{i-1}^{\nu x}, \beta^\nu\} \in \bar{\mathbb{G}}^i.$$

Therefore, if all the computations are performed correctly, the flows between 4 players include the plaintexts X_1, X_2 and X_3 presented on Figure 3. The plaintext X'_4 is the 4 first elements of X_4 only while the last element of X_4 is $K = (g_0^{\nu_1\nu_2\nu_3\nu_4})^{x_1x_2x_3x_4} = g_4^{x_1x_2x_3x_4}$.

g_0					$= X_0$
g_1	$g_1^{x_1}$				$= X_1 = \Phi(X_0, x_1, \nu_1)$
$g_2^{x_2}$	$g_2^{x_1}$	$g_2^{x_1x_2}$			$= X_2 = \Phi(X_1, x_2, \nu_2)$
$g_3^{x_2x_3}$	$g_3^{x_1x_3}$	$g_3^{x_1x_2}$	$g_3^{x_1x_2x_3}$		$= X_3 = \Phi(X_2, x_3, \nu_3)$
$g_4^{x_2x_3x_4}$	$g_4^{x_1x_3x_4}$	$g_4^{x_1x_2x_4}$	$g_4^{x_1x_2x_3}$	$g_4^{x_1x_2x_3x_4}$	$= X_4 = \Phi(X_3, x_4, \nu_4)$

Fig. 3. Honest execution, when $n = 4$. We denote $\nu_i = \log_{g_{i-1}}(g_i)$.

One can indeed check by induction that the j-th element of X_i is $(g_0^{\mu_i})^{y_i/x_j} = g_i^{y_i/x_j}$, where $\mu_i = \nu_1 \cdots \nu_i \bmod q$, $g_i = g_0^{\mu_i}$ and $y_i = x_1 \cdots x_i \bmod q$. Therefore, the i-th element α_i of the down-flow is $g_n^{y_n/x_i}$ which with the knowledge of x_i leads to the common value $K = \alpha_i^{x_i} = g_n^{y_n}$.

4.3 Dictionary Attacks

In EKE, we have to be careful of the content in the ciphertext since any redundancy in the concatenation of the plaintexts in the flows of the protocol could be used by the adversary to mount a dictionary attack. The adversary could decrypt flows using all the passwords in the dictionary and look for this redundancy.

Namely, the trivial conversion wherein one substitutes in a group Diffie-Hellman protocol [10] the signature scheme by a symmetric encryption scheme is easily seen insecure, while it works in the two-party case. This conversion indeed produces a protocol in which all the computations are performed with $\nu_i = 1$, for all i; therefore the last element of each plaintext in Fl_i also belongs to the plaintext in Fl_{i+1}.

5 Security Result

In this section we assert that under reasonable and well-defined intractability assumptions the protocol EKE securely distributes session keys. We deal with the AKE goal only and thus do not consider forward-secrecy here. However, concurrent executions are possible.

Theorem 1. *Let P be the EKE protocol,* **SK** *be the session-key space and* Password *be a finite dictionary of size N. Let $\mathcal{A}$ be an adversary against the AKE security of P within a time bound T, after q_s interactions with the parties, q_h hash-queries, and q_e encryption/decryption queries. Then we have:*

$$\mathsf{Adv}_P^{\mathsf{ake}}(\mathcal{A}) \leq \frac{2q_s}{N} + 2q_s \mathsf{Adv}_{\mathbb{G}}^{\mathsf{mddh}_n}(T') + 2q_h \mathsf{Succ}_{\mathbb{G}}^{\mathsf{tgcdh}_n}(T') + \frac{2Q^2}{q^2}$$

where $T' \leq T + nQ\tau_{\mathbb{G}}$, $Q = 3q_s + q_e$ and $\tau_{\mathbb{G}}$ is the computational time for an exponentiation in $\mathbb{G}$. (Recall that q is the order of $\mathbb{G}$).

This theorem shows that EKE 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 the adversary's capability to enumerate passwords off-line is only limited by its computational power.

Of course, the security results only holds provided that the adversary does not solve either the trigon group computational problem TG-CDH or the multi-decisional Diffie-Hellman problem M-DDH. But these terms can be made negligible by appropriate choice of parameters for the group $\mathbb{G}$.

6 Proof of Security

In this section we show that the protocol EKE achieves security against dictionary attacks as claimed by Theorem 1. We first introduce the notations we will use and then prove that the best the adversary can do is to essentially eliminate one password from the dictionary per initiated session (maybe concurrently). Here, we present a proof that does not yet deal with forward-secrecy. All the lemmas are proven in the full version of this paper [8].

6.1 Operator Ψ

As illustrated in Figure 2, each player U_i generates a new basis when processing an up-flow by raising the values it received to the power of its random blinding exponent ν_i. But let us notice that given a $\mathsf{TG\text{-}CDH}$ instance of size n, with the $\mathsf{TG\text{-}CDH}$-solution, one can easily derivate the trigon whose lines are the flows sent during an honest execution of EKE, by raising the lines to the power of random and independent exponents.

We denote by θ a n-tuple of elements in $\mathbb{Z}_q^\star$, by θ_i the i-th component of θ, and by $[g]_n$ the n-tuple $(g, \ldots, g)$. For any line L of length $i+1$, the operator Ψ takes as input a n-tuple θ, a random exponent ν and applies a (multiplicative) self-reduction of the line L as follows:

- **[Using** θ**]** first, one raises the first i elements of L to the power of Θ_i/θ_1, $\ldots$, Θ_i/θ_i respectively, and the last element to the power of Θ_i, where $\Theta_i = \theta_1 \cdots \theta_i$;
- **[Change of basis]** then, one raises all the elements of the tuple to the power ν.

For example, from a line $L = (g_1, \ldots, g_{i+1})$, with any tuple θ and ν, one gets

$$\Psi(L, \theta, \nu) = (g_1^{\nu\Theta_i/\theta_1}, \ldots, g_i^{\nu\Theta_i/\theta_i}, g_{i+1}^{\nu\Theta_i}),$$

where $\Theta_i = \theta_1 \cdots \theta_i$. A line L of form $\{g^{y_i/x_1}, \ldots, g^{y_i/x_i}, g^{y_i}\}$ where $y_i = x_1 \cdots x_i$ is thus represented as follows:

$$L = \Psi([g]_{i+1}, (x_1, \ldots, x_i, 1, \ldots, 1), 1) \in \bar{\mathbb{G}}^{i+1}.$$

The following lemmas exhibit some useful results about the operators Φ and Ψ:

Lemma 2 (Equality of distributions). *Let $g \in \bar{\mathbb{G}}$ and $L = \{g^{\alpha_0}, \ldots, g^{\alpha_i}\} \in \bar{\mathbb{G}}^{i+1}$. The following two distributions are perfectly indistinguishable:*

$$\left\{\Psi\big(L, \theta, \nu\big)\right\}_{(\theta,\nu)\in(\mathbb{Z}_q^\star)^{n+1}} \text{ and } \left\{g^{r_0}, \ldots, g^{r_i}\right\}_{(r_0,\ldots,r_i)\in(\mathbb{Z}_q^\star)^{i+1}}$$

Lemma 3 (Commutativity and composition). *Let* $x, \nu, \nu' \in \mathbb{Z}_q^\star$ *and* $\theta, \theta' \in (\mathbb{Z}_q^\star)^n$. *For any line* $L \in \bar{\mathbb{G}}^i$, *we have (where* $\theta\theta'$ *is the component-wise multiplication of the vectors* θ *and* θ'*):*

$$\begin{aligned} \Psi\big(\Phi(L,x,\nu),\theta,\nu'\big) &= \Phi\big(\Psi(L,\theta,\nu'),x\theta_i,\nu\big); \\ \Psi\big(\Psi(L,\theta,\nu),\theta',\nu'\big) &= \Psi(L,\theta\theta',\nu\nu'); \\ \Psi\big(\Phi(L,x,\nu),[1]_n,\nu'\big) &= \Phi(L,x,\nu\nu'); \\ \text{if } (\forall j < i, \theta_j = \theta'_j), \Psi(L,\theta,\nu) &= \Psi(L,\theta',\nu). \end{aligned}$$

6.2 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 $\mathbf{G}_6$. We let b and b' be defined as in Section 2.1 and refer to S_i as the event $b = b'$ in game $\mathbf{G}_i$. We also define the event $\mathsf{Encrypt}_i$ as the event that a flow has been encrypted, but not decrypted *first* (see below), by the adversary under *pw* (with any symmetric encryption scheme $\mathcal{E}$ or $\mathcal{E}'$). We use the following lemma within our sequence of games [20]:

Lemma 4. *Let* E, E' *and* F, F' *be some events defined on a probability space. Let us assume that* $\Pr[F] = \Pr[F'] = \varepsilon$ *and* $\Pr[E \wedge \neg F] = \Pr[E' \wedge \neg F']$. *Then,* $\big|\Pr[E] - \Pr[E']\big| \leq \varepsilon$.

Game $\mathbf{G_0}$: This is the real attack $\mathbf{Game}^{\mathsf{ake}}(\mathcal{A}, P)$ in which several oracles are available to the adversary: a hash oracle, the encryption/decryption oracles, and all the instances of players (in order to cover concurrent executions).

> **Rule 1**: The instances of players process each Send-query with a pair of random exponents (x_i, ν_i), using the operators Φ and Φ'.

Thus, the instances of players can easily answer to the Reveal-query and the Test-query. The $\mathsf{Execute}$-query is proceeded similarly. By definition, $\Pr[\mathsf{S}_0] = (\mathsf{Adv}_P^{\mathsf{ake}}(\mathcal{A}) + 1)/2$.

Game $\mathbf{G_1}$: We simulate the hash and the encryption/decryption oracles as in $\mathbf{G}_0$ by maintaining five lists: a hash list (Λ_H), encryption lists (Λ_E, Λ'_E) and decryption lists (Λ_D, Λ'_D). The lists are initially empty. We denote by q_H the size of Λ_H, and by q_E the number of encryption-decryption relations: i.e. q_E is the size of $\Lambda_E \cup \Lambda'_E$. The queries are answered as follows:

- *Hash-query*: For a query q such that a record (q, r) appears in Λ_H, the answer is r. Otherwise r is chosen at random from $\{0,1\}^{\ell_2}$ and the record (q, r) is added to Λ_H. We have $\mathcal{H}(q) = r$.
- *Encryption-query*: For an encryption query (k, X) to $\mathcal{E}$ (resp. $\mathcal{E}'$) such that a record $(*, k, X, Y)$ appears in Λ_E (resp., Λ'_E) the answer is Y. Otherwise Y is a random ciphertext of length $|X|$. The record (δ, k, X, Y) is then added to encryption list Λ_E (resp. Λ'_E). $\delta \in \{0,1\}$ is a bit indicating the originator of the query: if the query comes from the simulator then $\delta = 0$ else the query comes from the adversary $\delta = 1$.

– *Decryption-query*: For a decryption query (k, Y) to $\mathcal{D}$ (resp. $\mathcal{D}'$) such that a record $(*, k, X, Y)$ appears in Λ_E (resp. Λ'_E), the answer is X. Otherwise X is generated by the following rule:

  Rule 2: X is a random tuple $\{g^{r_i}\}_{1 \leq i \leq |Y|}$ where $r_1, \ldots, r_{|Y|}$ are randomly drawn in $\mathbb{Z}_q^\star$.

  The record (k, Y, X) is then added to the decryption list Λ_D (resp. Λ'_D) while the record $(0, k, X, Y)$ is added to the encryption list Λ_E (resp. Λ'_E).

We notice that a decryption-query adds a record to both the encryption and the decryption lists, while an encryption-query adds a record to the encryption list only. In both cases, a later encryption or decryption query on the same elements does not add any record to any list. Hence, a record (k, Y, X) appears in the decryption list if and only if the decryption query (k, Y) has been asked *first* (i.e., before the corresponding encryption query).

With this definition, $\mathsf{Encrypt}_i$ is defined in game $\mathbf{G}_i$ as the event that there exists a record $(1, pw, X, Y)$ in an encryption list, such that Y has been submitted in a Send-query. Note that this implies that the corresponding record (pw, Y, X) does not appear in any decryption list.

¿From the above simulation we easily see that the games $\mathbf{G}_1$ and $\mathbf{G}_0$ are perfectly indistinguishable, unless the permutation property of the block ciphers does not hold. One could have avoided collisions in the above simulation but this is at most $q_E^2/2(q-1)^2$ since the smallest set for the encryption functions is $|\bar{\mathbb{G}}^2| = (q-1)^2$:

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

Game $\mathbf{G_2}$: We delete the executions wherein the adversary may have guessed the password. More formally, we delete the executions wherein event $\mathsf{Encrypt}_1$ occurs: i.e. a $\mathsf{Send}(\Pi, Y)$-query is asked and a record of the form $(1, pw, *, Y)$ appears in an encryption list. In these executions, we stop setting b' at random:

$$|\Pr[\mathsf{S}_2] - \Pr[\mathsf{S}_1]| \leq \Pr[\mathsf{Encrypt}_1]. \tag{2}$$

Game $\mathbf{G_3}$: We simulate the instances of players from a tuple $(x_1, \ldots, x_n)$. This tuple allows us to compute an instance $\mathcal{D} = \mathcal{D}(x_1, \ldots, x_n)$ of the $\mathsf{TG\text{-}CDH}_n$ with its solution $g^{x_1 \cdots x_n}$. We use $\mathcal{D}$ to construct using blinding exponents $\nu_1, \ldots, \nu_n$ the triangular structure illustrated in Figure 3 where all the bases are randomized. The lines of this structure will be used in this game to answer to the Send-queries. The lines of this triangular structure are denoted and constructed as

$$L_i = \begin{cases} \{g\} & \text{if } i = 0, \\ \Phi(L_{i-1}, x_i, \nu_i) & \text{if } 1 \leq i \leq n, \end{cases}$$

where the $n+1$-th element κ of L_n is $\kappa = g_n^{x_1 \cdots x_n} = \left(g^{x_1 \cdots x_n}\right)^{\nu_1 \cdots \nu_n}$.

We now show how to use these lines to simulate the instances of players. We first maintain a list Λ_Ψ that keeps track of the exponents θ used to blind a line L_i: $L = \Psi(L_i, \theta, \nu)$. This list contains records of the form (i, θ, ν, L) and is initially set to $\{(0, [1]_n, 1, \{g\})\}$. Then, we answer to a $\mathsf{Send}(\Pi_i^t, \mathsf{Fl})$-query as follows:

– Π_i^t is waiting for an up-flow: if the length of Fl is different from i, then we do not do anything. Otherwise we do perform the following two steps.
  1. if $i = 1$ then one sets $L = \{g\}$, else one invokes the decryption oracle to get $L = \mathcal{D}_{pw}(\mathsf{Fl})$.
  2. one computes line L' according to **Rule 1.1** and encrypts *some* elements of L'. If $i < n$, then the whole line L' is encrypted into $\mathcal{E}_{pw}(L')$. Otherwise, if $i = n$, then only the n first elements of L', we refer to them as L'', are encrypted using $\mathcal{E}'_{pw}$. Finally, Π_i^t waits for the down-flow.

     Rule 1.1: We first chooses two random exponents $(\rho_i^t, \mu_i^t) \in (\mathbb{Z}_q^\star)^2$. If $(i-1, \theta, \nu, L) \in \Lambda_\Psi$ for some θ, ν, then we compute $L' = \Psi(L_i, \theta', \mu_i^t \nu)$ where θ' is defined to θ except that $\theta'_i = \rho_i^t$, and we update the list Λ_Ψ. Otherwise one applies the **Rule 1.1'** presented below.

     Rule 1.1': One still uses **Rule 1**, but with $(x_i \rho_i^t, \nu_i \mu_i^t)$ instead of (x_i, ν_i).

     The random ρ_i^t is different each time one answers this flow (either by **Rule 1.1** or **Rule 1.1'**.) Indeed, the same flow Fl may be sent several times by the adversary in different and concurrent executions.

  By induction, one easily show that any line L either comes from **Rule 2** or, under $\neg\mathsf{Encrypt}$, from a previous **Rule 1.1**.

– Π_i^t is waiting for a down-flow: if the length of Fl is different from n, then we do not do anything. Otherwise, we invoke the decryption oracle $\mathcal{D}'_{pw}(\mathsf{Fl})$ to obtain L'' or more specifically to obtain the i-th element α_i^t in L'':

  Rule 3: For any Π_i^t, K_i^t is set to be $(\alpha_i^t)^{x_i \rho_i^t}$, where α_i^t is the i-th element in the down-flow L'' received by Π_i^t.

We have now to show that the Φ relation between the lines L_{i-1} and L_i is "preserved" by the Ψ transformation. The following lemma shows it: two lines L_{i-1}, L_i of the triangular structure already related by the operator Φ are still related by Φ after having respectively been transformed into L, L' by the operator Ψ.

Lemma 5. $L' = \Phi(L, x_i \rho_i^t, \mu_i^t \nu_i)$.

By Lemma 5, our simulation simply makes the player choose as private exponent $x_i \rho_i^t$ and as blinding exponent $\nu_i \mu_i^t$ (both values are uniformly distributed because μ_i^t and ρ_i^t are). From the value K_i^t, we can then easily compute the session key $sk_{\Pi_i^t}$ to be $\mathcal{H}(\mathcal{U} \| \mathsf{Fl}_n \| K_i^t)$.

It follows that games $\mathbf{G}_2$ and $\mathbf{G}_3$ are perfectly indistinguishable:

$$\Pr[\mathsf{S}_3] = \Pr[\mathsf{S}_2]. \tag{3}$$

Game $\mathbf{G_4}$: We now modify the way the decryption queries are simulated by modifying the **Rule 2**, in order to embed the instance $\mathcal{D}$ in the answers output by the decryption oracle, so that an attack may help us to solve it.

> **Rule 2.1**: One chooses a random blinding exponent $\nu \in \mathbb{Z}_q^\star$ and random exponents θ in $(\mathbb{Z}_q^\star)^n$. If Y is a query to $\mathcal{D}_{pw}$, then X is set to $\Psi(L_i, \theta, \nu)$, where $i = |Y| - 1$. If Y is a query to $\mathcal{D}'_{pw}$, and $|Y| = n$, then X' is set to $\Psi(L_n, \theta, \nu)$, but X is set to the n first elements of X'. In both cases, the list Λ_Ψ is updated.

From Lemma 2, with random θ and ν, all the answers X are perfectly random in $\bar{\mathbb{G}}^{|Y|}$:

$$\Pr[\mathsf{S}_4] = \Pr[\mathsf{S}_3]. \tag{4}$$

Before going further on, let us claim the following lemma. It shows that from now on, **Rule 1.1'** will not be used anymore.

Lemma 6. *If one assumes $\neg\mathsf{Encrypt}_4$ in game $\mathbf{G}_4$, any plaintext L included in a flow received via a Send-query is recorded in Λ_Ψ (possibly with one more element if L has been decrypted by $\mathcal{D}'_{pw}$, and thus corresponds to a down-flow).*

Game $\mathbf{G_5}$: In the above game $\mathbf{G}_4$, one can remark that knowledge of the x_i's is not needed, one could only be given an instance $\mathcal{D} = \mathcal{D}(x_1, \ldots, x_n)$ of the $\mathsf{TG\text{-}CDH}_n$ with its solution $g^{x_1 \cdots x_n}$ only.

But while the x_i's are not needed to construct the triangular structure $\{L_0, \ldots, L_n\}$, the x_i's are needed to compute K_i^t. This value is in turn used to compute the session key $sk_{\Pi_i^t}$ and therefore needed to answer to the Reveal-query and Test-query. Thus, if we want to avoid the use of the x_i's, we need to find another way to compute K_i^t.

Fortunately, it is possible: according to Lemma 6, if $\mathsf{Encrypt}_4$ did not occur then the upflow $L' = \Phi(L, x_i\rho_i^t, \nu_i\mu_i^t)$ has been generated by the **Rule 1.1** with as input of $L = \Psi(L_{i-1}, \theta, \nu)$. Let us recall that L'' corresponds to the n first elements of the tuple $\Psi(L_n, \theta', \nu')$, $\kappa = g^{x_1 \cdots x_n}$ is the last component of L_n and Θ' is equal to the product $\theta'_1 \cdots \theta'_n$. We can now compute K_i^t by modifying **Rule 3** to be:

> **Rule 3.1**: $K_i^t = \kappa^{\nu'\Theta' \cdot \rho_i^t / \theta'_i}$. And then, the session keys can been computed without the x_i's.

Lemma 7. ***Rule 3*** *and* ***Rule 3.1*** *lead to the same result.*

By Lemma 7, the games $\mathbf{G}_4$ and $\mathbf{G}_5$ are perfectly indistinguishable, as soon as $\mathsf{Encrypt}_5$ does not occur.

$$\Pr[\mathsf{S}_5] = \Pr[\mathsf{S}_4]. \tag{5}$$

Game $\mathbf{G_6}$. Finally, we are just given an instance $\mathcal{D} = \mathcal{D}(x_1, \ldots, x_n)$ of the $\mathsf{TG\text{-}CDH}_n$ *without* its solution $g^{x_1 \cdots x_n}$. Then, if the adversary helps us to get some K_i^t, we have solved the $\mathsf{TG\text{-}CDH}_n$ problem (and we are done, see the Lemma 8 below).

However, since we do not know κ, we can no longer compute K_i^t and can not therefore answer to Reveal-oracles (and the Test-query). We simply simulate the Reveal-oracles (even for a Test-query), by answering a random value, without asking the hash oracle $\mathcal{H}$.

Let us denote by AskH the event that $\mathcal{A}$ makes a hash-query of the form $(\mathcal{U} \| \mathsf{Fl}_n \| K_i^t)$, where Fl_n is the down-flow received by any Π_i^t. Unless neither AskH occurs (nor $\mathsf{Encrypt}_5$) games $\mathbf{G}_5$ and $\mathbf{G}_6$ are perfectly indistinguishable:

$$\Pr[\mathsf{S}_6 \,|\, \neg\mathsf{AskH}] = \Pr[\mathsf{S}_5 \,|\, \neg\mathsf{AskH}]. \tag{6}$$

The probability of event AskH is upper-bounded in the following lemma. Let us recall that q_h denotes the number of hash-queries asked by the adversary.

Lemma 8. $\Pr[\mathsf{AskH}] \leq q_h \mathsf{Succ}_{\mathbb{G}}^{\mathsf{tgcdh}_n}(T + (q_s + q_E)n\tau_{\mathbb{G}})$.

In this game, answers to the Reveal-queries, and thus to the Test-query, are purely random. Then, it is straightforward to see that

$$\Pr[\mathsf{S}_6] = \frac{1}{2}. \tag{7}$$

From Lemma 4 and Equations (6), (7), we get:

$$\left| \Pr[\mathsf{S}_5] - \frac{1}{2} \right| = \left| \Pr[\mathsf{S}_5] - \Pr[\mathsf{S}_6] \right| \leq \Pr[\mathsf{AskH}].$$

Finally, from Equations (1), (2), (3), (4) and (5), and Lemmas 4, 8, we get:

$$\left| \Pr[\mathsf{S}_0] - \frac{1}{2} \right| \leq \Pr[\mathsf{Encrypt}_1] + \frac{q_E^2}{q^2} + q_h \mathsf{Succ}_{\mathbb{G}}^{\mathsf{tgcdh}_n}(T + (q_s + q_E)n\tau_{\mathbb{G}}). \tag{8}$$

6.3 Probability of Event $\mathsf{Encrypt}_1$

The security against dictionary attacks is measured by the probability that the event $\mathsf{Encrypt}_1$ occurs. To evaluate this probability, we define a game wherein the view of the adversary is perfectly independent from the password pw, in the information theoretical sense. First, let us note that the games $\mathbf{G}_2$, $\mathbf{G}_3$, $\mathbf{G}_4$ and $\mathbf{G}_5$ are perfectly indistinguishable, and thus

$$\Pr[\mathsf{Encrypt}_5] = \Pr[\mathsf{Encrypt}_1]. \tag{9}$$

We define an auxiliary game $\mathbf{G}_6''$ similar to game $\mathbf{G}_5$ except that we answer differently to a $\mathsf{Send}(\Pi_i^t, \mathsf{Fl})$-query when instance Π_i^t is waiting for an up-flow. In this game $\mathbf{G}_6''$, we in fact re-define all coefficients used by the random self-reducibility and entirely re-blind line L_i:

Rule 1.2: Whatever appears in Λ_Ψ so far, we choose a random exponent $\nu \in \mathbb{Z}_q^\star$, a full vector $\theta \in (\mathbb{Z}_q^\star)^n$ and compute $L' = \Psi(L_i, \theta, \nu)$. We then update Λ_Ψ.

By Lemma 2, the plaintext L' is indistinguishable from a random plaintext in $\bar{\mathbb{G}}^{i+1}$ and, therefore, the simulation is completely independent from the password pw. So we have

$$\Pr[\mathsf{Encrypt}_6''] = q_s/N. \tag{10}$$

Moreover the only difference between games $\mathbf{G}_6''$ and $\mathbf{G}_5$ is in the way the Send-queries are answered. On input of a line $L = (a_1, \ldots, a_{i-1}, a_i)$, the **Rule 1.1** generates line $L' = (a_1^{\nu x}, \ldots, a_{i-1}^{\nu x}, a_i^{\nu}, a_i^{\nu x})$ while the **Rule 1.2** generates $L'' = (g^{r_0}, \ldots, g^{r_i})$. Using the classical hybrid technique we can obtain:

$$\left| \Pr[\mathsf{Encrypt}_6''] - \Pr[\mathsf{Encrypt}_5] \right| \leq q_s \, \mathsf{Adv}_{\mathbb{G}}^{\mathsf{mddh}_n}(T + (q_s + q_E) n \tau_{\mathbb{G}}). \tag{11}$$

Finally, Equations (9), (10) and (11) lead to

$$\Pr[\mathsf{Encrypt}_1] \leq \frac{q_s}{N} + q_s \mathsf{Adv}_{\mathbb{G}}^{\mathsf{mddh}_n}(T + (q_s + q_E) n \tau_{\mathbb{G}}).$$

Note that q_E is the size of $\Lambda_E \cup \Lambda_E'$ and it is thus equal to q_e plus the number of queries asked by our simulation (at most two per Send-query): $q_E \leq q_e + 2q_s$. This note, combined with Equation (8), concludes the proof. □

7 Forward-Secrecy

The above proof does 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 after this one):

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

Then we define a new flavor of freshness, saying that an oracle Π_i^t is **Fresh** (or holds a **Fresh** key sk) if the following conditions hold. First, no $\mathsf{Corrupt}$-query has been made by the adversary since the beginning of the game. Second, Π_i^t has computed a session key and neither Π_i^t nor its partners have been asked for a Reveal-query. The partnering notion is more formally defined in the full version of this paper [8].

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 $\mathsf{Adv}_P^{\mathsf{akefs}}(\mathcal{A})$ the advantage an adversary can get on a fresh key, with the ability to make a $\mathsf{Corrupt}$-query.

Theorem 2. *Let P be the EKE protocol,* **SK** *be the session-key space and* $\mathsf{Password}$ *be a finite dictionary of size N. Let $\mathcal{A}$ be an adversary against the AKE security of P within a time bound T, after q_s interactions with the parties,*

q_h hash-queries, and q_e encryption/decryption queries. Then, there exists $k \leq n$ such that:

$$\mathsf{Adv}_P^{\mathsf{akefs}}(\mathcal{A}) \leq \frac{2q_s}{N} + 2q_s \mathsf{Adv}_{\mathbb{G}}^{\mathsf{mddh}_n}(T') + 2n^2 q_s^n q_h \mathsf{Succ}_{\mathbb{G}}^{\mathsf{tgcdh}_k}(T') + \frac{2Q^2}{q^2}.$$

where $T' \leq T + Qn\tau_{\mathbb{G}}$, $Q = 5q_s + q_e$ and $\tau_{\mathbb{G}}$ is the time of computation required for an exponentiation in $\mathbb{G}$. (Recall that q is the order of $\mathbb{G}$).

Proof. The proof of this theorem is given in the full version of this paper [8].

8 Mutual Authentication

The well-known approach for turning an Authenticated Key Exchange (AKE) protocol into a protocol that provides mutual authentication (MA) is to use the shared session key to construct a simple "authenticator" for the other parties. In [10], we already described the transformation, and justified its security in the random-oracle model. The first analysis has been done before in the two-party case in [2].

9 Conclusion

This paper provides the first formal treatment of the authenticated group Diffie-Hellman key exchange problem that encompasses dictionary attacks. Addressed in this paper are two security goals of the group Diffie-Hellman key exchange: the authenticated key exchange and the mutual authentication. For each we present a definition, a protocol and a security proof in both the random oracle model and the ideal-cipher model that the protocol meets its goals. Furthermore, we consider forward-secrecy, even if the reduction is not very efficient. Reducing the ideal-cipher model assumption and improving the reduction for the forward-secrecy are still open problems.

Acknowledgments

The authors thank the anonymous referees for their useful comments. 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-49479. Disclaimer available at `http://www-library.lbl.gov/disclaimer`.

References

1. N. Asokan and P. Ginzboorg. Key Agreement in Ad-hoc Networks. *Computer Communications*, 23(18):1627–1637, 2000.

2. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated Key Exchange Secure Against Dictionary Attacks. In B. Preneel, editor, *Proc. of Eurocrypt '00*, LNCS 1807, pages 139–155. Springer-Verlag, 2000.
3. S. M. Bellovin and M. Merrit. 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. J. Black and P. Rogaway. Ciphers with Arbitrary Finite Domains. In *Proc. of the RSA Cryptographer's Track (RSA CT '02)*, LNCS 2271, pages 114–130. Springer-Verlag, 2002.
5. Bluetooth. Specification of the Bluetooth System, December 1999. Available at `http://www.bluetooth.com/developer/specification/specification.asp`.
6. D. Boneh. The Decision Diffie-Hellman Problem. In *Third Algorithmic Number Theory Symposium*, LNCS 1423, pages 48–63. Springer-Verlag, 1998.
7. V. Boyko, P. MacKenzie, and S. Patel. Provably Secure Password-Authenticated Key Exchange Using Diffie-Hellman. In B. Preneel, editor, *Proc. of Eurocrypt '01*, LNCS 1807, pages 156–171. Springer-Verlag, 2000.
8. E. Bresson, O. Chevassut, and D. Pointcheval. Group Diffie-Hellman Key Exchange Secure against Dictionary Attacks. In Y. Zheng, editor, *Proc. of Asiacrypt '2002*. Springer, December 2002. Full Version – `http://www.di.ens.fr/users/pointche`.
9. E. Bresson, O. Chevassut, and D. Pointcheval. The Group Diffie-Hellman Problems. In H. Heys and K. Nyberg, editors, *Proc. of SAC '2002*, LNCS. Springer-Verlag, August 2002.
10. E. Bresson, O. Chevassut, D. Pointcheval, and J. J. Quisquater. Provably Authenticated Group Diffie-Hellman Key Exchange. In *Proc. of 8th ACM Conference on Computer and Communications Security*, pages 255–264, November 2001.
11. W. Diffie and M. Hellman. New Directions In Cryptography. In *IEEE Transactions on Information Theory*, volume IT-22(6), pages 644–654, November 1976.
12. O. Goldreich and Y. Lindell. Session-Key Generation using Human Passwords Only. In J. Kilian, editor, *Proc. of Crypto '01*, LNCS 2139, pages 408–432. Springer-Verlag, August 2001.
13. M. Jakobsson and S. Wetzel. Security Weaknesses in Bluetooth. In *Proc. of the RSA Cryptographer's Track (RSA CT '01)*, LNCS 2020, pages 176–191. RSA Data Security, Springer-Verlag, 2001.
14. J. Katz, R. Ostrovsky, and M. Yung. Efficient Password-Authenticated Key Exchange using Human-Memorable Passwords. In *Proc. of Eurocrypt '01*, LNCS 2045, pages 475–494. Springer-Verlag, May 2001.
15. P. MacKenzie. More Efficient Password Authenticated Key Exchange. In D. Naccache, editor, *RSA Conference '01*, LNCS 2020, pages 361–377. Springer-Verlag, 2001.
16. M. Naor and O. Reingold. Number-Theoretic Constructions of Efficient Pseudo-Random Functions. In *Proc. of 38th FOCS*, pages 458–467. IEEE, 1997.
17. NIST. AES, December 2000. Available at `http://www.nist.gov/aes`.
18. K. Obraczka, G. Tsudik, and K. Viswanath. Pushing the Limits of Multicast in Ad Hoc Networks. In *International Conference on Distributed Computing Systems*, April 2001.
19. C. E. Perkins. *Ad Hoc Networking*. Addison Wesley, 2001.
20. V. Shoup. OAEP Reconsidered. In J. Kilian, editor, *Proc. of Cryto' 01*, LNCS 2139, pages 239–259. Springer-Verlag, 2001.
21. L. Zhou and Z. J. Haas. Securing Ad Hoc Networks. *IEEE Network Magazine*, 13(6), 1999.