Constant Round Authenticated Group Key Agreement via Distributed Computation *

Emmanuel Bresson[1] and Dario Catalano[2]

[1] Cryptology Department, CELAR, 35174 Bruz Cedex, France
Emmanuel.Bresson@polytechnique.org
[2] Dépt. d'informatique, École normale supérieure, 75230 Paris Cedex 05, France.
Dario.Catalano@ens.fr

Abstract. A group key agreement protocol allows a set of users, communicating over a public network, to agree on a private session key. Most of the schemes proposed so far require a linear number (with respect to the number of participants) of communication rounds to securely achieve this goal. In this paper we propose a new constant-round group key exchange protocol that provides efficiency and privacy under the Decisional Diffie-Hellman assumption. Our construction is practical, conceptually simple and it is obtained by taking advantage of the properties of the El-Gamal encryption scheme combined with standard secret sharing techniques.

1 Introduction

Group key agreement protocols allow several parties to come up with a common secret from which a session key can be derived. Hence, these protocols are likely to be used in numerous group-oriented scenarios, such as video conferencing, collaborative applications, secure replicated database, in order to achieve secure multicasting network layer among the parties. Typically, the parties hold some long-term keys that enable them to communicate privately from point to point, or to authenticate messages. Another setting considers password-based authentication, which we are not dealing with in this paper. The final goal of group key exchange protocols is to efficiently implement "secure" multicast channels. In order to specify what "efficiently" and "secure" mean, one may consider some desirable properties for a group key agreement protocol. *Efficiency*, while not measuring security, is to be considered as a crucial property when designing key agreement protocols and it is quantified as the number of communication rounds, as well as the space and computing resources required to agree on the final key. Indeed, limiting the number of rounds can be of prime importance in many real-life applications. Consider for example the case of a group where some (or all the) participants have a slow network connection. In such a situation the efficiency of the entire protocol can be severely degraded even if the "slow guys" constitute a very small minority of the group. Other scenarios where reducing the number of rounds is important are all those applications where many players are involved, or where many keys have to be exchanged.

* Extended abstract. A full version of this paper can be found at www.di.ens.fr/~catalano.

Combining efficiency and security is not a trivial task. One of the most basic security property that is required to a group key agreement protocol is the so-called *contributory*: all the parties are ensured to properly contribute to the final secret value and to its distribution — in other words, no party should be able to impose the value of the session key which should be uniformly distributed over the session key space. On top of that a key agreement scheme should guarantee some *privacy* property for final session key, i.e. that no eavesdropper should be able to gain information (at least in some computational sense) about the key after having seen the messages exchanged between the parties being involved in the protocol. In general, however, a group key agreement should preserve privacy even in the case on which the network is under control of some malicious adversary that may try to modify any message sent among the players. Thus the main features we would like to find in a Group Key Agreement scheme are security and efficiency, in the presence of an *active* adversary. The typical approach to the problem [1,5,8,9] requires some data to go through the complete set of parties, which by sequentially adding some private contribution, "build" the actual key in a linear number of rounds of communication. The main problem with this approach is, of course, that it may lead to very slow protocols. To improve on communication complexity the natural solution is to try to devise a scheme that allows for simultaneous sending of contributions.

So, at the very end, the basic problem of group key agreement can be simplified as follows: we want a set of players to agree on some random value that they will later, privately, reconstruct to use as shared key. Written in this way the problem seems to be quite similar to the standard multi-party computation goal where a group of players wants to compute the output of a public function when the input is shared among the participants. There is a crucial difference however: in the multi-party computation setting the output of the function is, in general, kept *shared* and may be publicly reconstructed if some conditions are met. In the group key agreement setting, on the other hand, we want the players to be able to *privately* reconstruct the secret. In other words, the goal of the key agreement is to establish a random value that at the end of the protocol should be disclosed to the players only. In this paper we basically combine standard secret sharing techniques [30] with the use of El-Gamal cryptosystem [20] to make this goal possible.

Related work – Some formal models for studying security of the session key were initiated by Bellare and Rogaway [5,6] and further refined by Blake-Wilson *et al.* [7,8]. Another formal model is based on the multi-party simulatability technique and was initiated by Bellare, Canetti and Krawczyk [2], and refined by Shoup [31]. Some classical examples of group key agreement protocols dealing with privacy are the generalizations of the original Diffie-Hellman paper [16], whose first proposals can be traced back to Ingemarsson *et al.* [23]. Some more sophisticated schemes [10,32] rely on the so-called *group Diffie-Hellman assumptions*, for which some reductions can be found in [11], while others are based on more heuristic, quite non-standard[1] assumptions [18]. Let us also mention some proposed schemes that are based on elliptic curve cryptography: Joux [24] proposed a single round method for 3-party Diffie-Hellman key agreement

[1] These assumptions make use of "multiple-decker" exponents, and are not easily related to DH.

using pairings. However, a generalization based on multi-linear forms is still an open problem [19].

As we said, a major issue of such protocols consists in efficiency, and this is especially true when considering large groups or dynamic peer group key agreement. Some protocols offering provable security have been recently analyzed by Bresson *et al.* [9,10]; they are essentially derived from an article by Steiner *et al.* [32]. However they require a linear number of communication rounds. In [12], Burmester and Desmedt proposed a very efficient, elegant protocol that needs only two rounds (three rounds when considering the confirmation step). The main advantage of constant round protocols is that the impact of "slow guys" is reduced, in the sense that 1 or several slow connections have essentially the same impact on efficiency. Burmester and Desmedt provide a security proof that reduces the privacy (one-wayness) of the session key to the (computational) Diffie-Hellman problem. However no proof of security (in the stronger sense of semantic security [22]) is provided in the original paper. Only recently, Katz and Yung [25] proposed a more general framework that provides a formal proof of security for this protocol, based on the DDH assumption. An interesting contribution of their paper is a scalable compiler that transforms any group key exchange protocol secure against a passive adversary into one that is secure against an active adversary, controlling the network.

In 1999, Li and Pieprzyk [26] proposed a key agreement protocol based on secret sharing techniques. They use the well-known polynomial secret sharing *à la* Shamir [30] to reconstruct a session key. While their work leads to a constant-round protocol and may appear quite similar to ours, it is actually less efficient. First of all they adopt an $(n+1)$-out-of-$2n$ sharing scheme and need to resort to secure channel to guarantee secrecy. In our case, on the other hand, we can use an n-out-of-n secret sharing scheme and no additional assumption is required. Furthermore in [26] to recover the secret the parties are required to perform Lagrange interpolation on the exponents. We emphasize that working in the exponents implies a relatively inefficient scheme, requiring $O(3n)$ exponentiations per player.

Our contributions – In this work, we propose a constant round key exchange protocol, based on secret sharing techniques, and using an asynchronous network. Our scheme is very efficient in terms of communication between the players (only two rounds of communications — plus a confirmation additional round — are required) and provides security (even with respect to parallel executions) under the well known Decisional Diffie-Hellman assumption. As noted above, only very few schemes proposed so far offer both authentication and privacy under standard assumptions [10, 25]. We emphasize that our solution achieves comparable bandwidth (in terms of the number of total bit per player exchanged) with respect to all previously proposed schemes. Also, if preprocessing is possible our protocol requires only - roughly - 2 exponentiations per player. Moreover we believe that our proposal allows for a more general approach to the problem. Indeed almost all previously suggested solutions are somehow generalizations of the basic Diffie-Hellman key exchange protocol [16], and thus are inherently related to the underlying (computational or decisional) assumptions. Finding alternative to existing solutions is not only a common practice in cryptography but a line of research of fundamental importance in practice. In this sense, in our case, the reduction to the

decisional Diffie-Hellman assumption comes *solely* from the fact that we are adopting the El Gamal cryptosystem as underlying encryption primitive (to take advantage, in terms of efficiency, of its nice properties). However, we stress here, that up to some loss in efficiency, it remains possible to successfully implement our protocol using a different semantically secure cryptosystem, relying on alternative intractability assumptions. One could even imagine a scheme in which the data are encrypted point-to-point using a symmetric encryption scheme (the drawback being there the number of secret keys).

2 The model

Players and network – We consider a network of n players $P_1, \ldots, P_n$, that are connected by point-to-point channels. We assume that each channel can be authenticated by the use of an underlying secure signature scheme. Thus, as already said, we consider an existing PKI and do not deal with password-authentication. We are based on an asynchronous network, in which messages can be delivered in arbitrary order. Moreover, and unless explicitly mentioned, we assume that each player can send several messages at the same time (multi-send property); this does not imply that all receivers will get the same message (broadcast). By saying that player A sends a message privately to B we intend A sending an encrypted message (with respect to B's public key) to B.

Adversary – The network is likely to be faulty, that is, not reliable because of attacks. To take into account such attacks, including those by "malicious" adversaries, we consider an active adversary $\mathcal{A}$ that has full control over the network. In particular we model this adversary as able to read, delete, and modify any message sent on the network. We stress that, as in previously proposed schemes, $\mathcal{A}$ does not have *any* control on the players themselves, and in particular, can not read their private memory[2].

Rushing attacks – We will assume that no player significantly deviates from the protocol, however we enable some players (but not all of them) to choose their contribution to the key according to some arbitrarily biased distribution (however we assume the adversary *does not* have any knowledge of such bias). Note that this allows for some player to adopt a *rushing* behavior by which he waits to receive the messages of the remaining parties (in a given round of communication) before sending his own. We stress, however, that this does not mean that rushing players do not follow the intructions nor that they follow instructions in a different order; it just means they choose their nonces non-uniformly, and if possible after the others. Moreover, we will assume that at least one player is *completely honest*.

Security notions – Our goal is to provide protocols allowing a pool of players to jointly agree on a common secret session key, in a presence of a malicious adversary (which includes the framework of a faulty network). We consider the following security notions, most of them are defined in [1].

[2] More precisely, $\mathcal{A}$ cannot access the storage of the session key; when considering forward-secrecy, one may consider that $\mathcal{A}$ *partially* corrupts the private memory of a player, and gets the long-term key.

Completeness means that, if the adversary is completely passive, the protocol terminates with each player holding a session key, which is the same for all of them.

Privacy means that whatever the adversary does, it cannot gain any information about the session key, if such a key is set (that is, if the protocol does not abort). In particular, it means that nobody outside of the group is able to compute the session key (*implicit authentication*).

Contributory means that each player is ensured to contribute equally to the final value of the key, and in particular, nobody can bias the distribution of the key.

Confirmation property encompasses the fact that a given player can be ensured a message has been delivered to other players. However, note that the receiver is not ensured that its confirmation has been delivered to the sender (unless using a confirmation again, which leads to infinite recursion). Such a network model thus needs to use time-out methods to abort a protocol if needed. Confirmations are used to achieve *explicit authentication*, by which every player has proof the group holds the same key.

Notations – Let ℓ be a security parameter. In the following we denote with $\mathbb{N}$ the set of natural integers and with $\mathbb{R}^+$ the set of positive real numbers. We say that a function $\mathsf{negl} : \mathbb{N} \to \mathbb{R}^+$ is *negligible* if for every polynomial $\rho(\ell)$ there exists a $\ell_0 \in \mathbb{N}$ s.t. for all $\ell > \ell_0$, $\mathsf{negl}(\ell) \leq 1/\rho(\ell)$. For $a, b \in \mathbb{N}$ we write $a|b$ if a divides b. If A is a set, then $a \leftarrow A$ indicates the process of selecting a at random and uniformly over A (which in particular assumes that A can be sampled efficiently).

2.1 The formal model

Players – We consider multiple, potentially parallel executions of the protocol; each player involved in the group has thus many *instances*, also called *oracles* running parallel sessions. The instances are seen as processes running on a given machine: some data (long-term key, public parameters) are shared, some data are specific to a process (eg, the session key). We assume that all signed messages are headed with sessions IDs, which uniquely identify each parallel run. We denote by sk_i^t the session key computed by player P_i in session whose ID is t. We consider a group of players whose membership is fixed, ie, there is no "Join" or "Remove" manipulations.

Adversarial capabilities – The adversary $\mathcal{A}$ is formalized through several *queries* describing possible interactions with oracles. Following [9], we define four types of queries: the Send-query is used to send arbitrary messages to an oracle; the Reveal-query is used to deal with known-key attacks, by revealing to $\mathcal{A}$ the session sk_i^t key hold by an oracle; the Corrupt-query leaks the long-term data L_i and allows to consider forward-secrecy; finally the Test-query is used to model the semantic security: it returns either the session key, or a random string, $\mathcal{A}$ being to guess which of the two cases.

Necessary conditions – A few straightforward conditions must be satisfied in order to properly use this model. These conditions are used in [1,9] to define the *Freshness* of a session key. First, a Reveal-query makes sense only if an oracle has already *accepted* a session key. Second a Test-query can be asked only once in the entire attack. Third, a Test-query must be asked before *any* Corrupt-query (asked to the Test-ed oracle or not). Four, a Test-query cannot be asked on a session for which some oracles have

accepted and have been Reveal-ed. The last three requirements ensure that the Test makes sense, that is, the session key is not "obviously" known by the adversary through basic means.

Definitions – We say that $\mathcal{A}$ *disrupts* an (instance of) player if it does not honestly relay the messages sent by this oracle (i.e., $\mathcal{A}$ is an active adversary generating faults on the network), but is still unable to access this player's internal data. When dealing with forward-secrecy, we say that $\mathcal{A}$ *corrupts* a player if it can get his long-term key. We denote L_i the long-term key used by player P_i.

We say that a Group Key Agreement Protocol, is secure if for any adversary $\mathcal{A}$ controlling the network the following four conditions are met.

Completeness: If $\mathcal{A}$ does not disrupt any oracle in a session t, then at the end of the protocol, there exists sk^t (which is efficiently computable) such that for all $i \in \{1, \ldots, n\}$ we have $sk_i^t = sk^t$.

Contributory (uniformity): If $\mathcal{A}$ does not disrupt any oracle in a session, then sk is uniformly distributed in the key space $\mathcal{K}$.

Privacy: We formalize this property as follows. Let $\mathcal{B}$ be a challenger facing the adversary and that runs the protocol, controlling all the players involved in the key exchange protocol being attacked. Let κ_1 be the corresponding session key computed by the members. On a Test-query, $\mathcal{B}$ chooses a random string κ_0 in the key space $\mathcal{K}$. Then it gives to the adversary either κ_0 or κ_1 (with equal probability). When terminating its attack, $\mathcal{A}$ should output a "guess" for the hidden bit b. We say that the protocol establishes a *private* session key sk if there exists a negligible function negl such that for sufficiently large ℓ, we have:

$$\mathsf{Adv}(\mathcal{A}) = 2\left(\Pr\left[\mathcal{A}(\mathcal{V}, \kappa_b) = b \,\middle|\, \begin{array}{c} \kappa_0 \leftarrow \mathcal{K};\quad \kappa_1 = sk \\ b \leftarrow \{0,1\} \end{array}\right] - \frac{1}{2}\right) = \mathsf{negl}(\ell)$$

Assumption 1 (DDH Assumption) *Let p and q two primes such that $|q| = \ell$ and $q|p-1$ and g an element of order q in $\mathbb{Z}_p^\star$. Let $Q = \langle g \rangle$. There exists a negligible function* negl *such that for sufficiently large ℓ, for any probabilistic polynomial-time distinguisher Δ, we have:*

$$\mathsf{Adv}^{\mathsf{ddh}}(\Delta) = \left|\Pr_{x,y}[\Delta(g, g^x, g^y, g^{xy}) = 1] - \Pr_{x,y,z}[\Delta(g, g^x, g^y, g^z) = 1]\right| = \mathsf{negl}(\ell)$$

Informally this assumption states that given the two elements $X = g^x \bmod p$ and $Y = g^y \bmod p$ the value $Z = g^{xy} \bmod p$ is indistinguishable from a random one in Q (see [22] for a definition of computational indistinguishability).

3 The Proposed Scheme

We start with an informal description of our protocol. The goal here is to highlight the main ideas underlying our construction without going too much into technical details.

Overview of the protocol – We will assume that each player P_i holds a pair of matching private/public key (x_i, h_i), where $h_i = g^{x_i} \bmod p$. We denote by $C_{i,j}(m, \alpha)$

an El-Gamal encryption of a message m under key h_j, using random α. Intuitively $C_{i,j}$ can be seen as an encrypted message sent from player P_i to player P_j.

The proposed protocol goes as follows. Every player P_i uniformly chooses a random value a_i as his own contribution to the key exchange protocol and a randomizer pad r_i. P_i proceeds by encrypting a_i, under the public key of every remaining player, and sends the ciphertext $C_{i,j}$ to player P_j. Moreover P_i randomly chooses a polynomial $f_i(z)$ of degree $n-1$ in $\mathbb{Z}_q^\star$ such that $f_i(0) = r_i$ and sends to player P_j the value $f_i(j)$. Once this first stage is over every player P_j sums the received $f_i(j)$ and multiplies the received ciphertexts (that is, corresponding to all indices but its own). Let us call C_j the resulting product and let δ_j be the plaintext corresponding to C_j. Note that, because of the homomorphic properties of the El-Gamal encryption scheme, the quantity $\delta_j \cdot a_j \bmod p$, is exactly $a = a_1 \cdots a_n \bmod p$, and, of course, it is the same for all the players involved in the protocol. Similarly the quantity $f(i)$, obtained by summing up all the $f_j(i)$'s, will be a share of a unique polynomial $f(z)$ such that $f(0) = r$ where $r = r_1 + \ldots + r_n \bmod q$. So to conclude the protocol the parties compute r, by interpolating $f(z)$ over $\mathbb{Z}_q^\star$ and set their session key $sk = a \cdot g^r$.

Dealing with rushing scenarios – One may wonder why we need to distribute encryptions of a_i, shares of r_i and then define the session key as $sk = \prod_{i=1}^n a_i \cdot g^r \bmod p$ rather than simply distribute encryptions of a_i and set the final key as $sk = \prod_{i=1}^n a_i \bmod p$. As a matter of fact, this second solution may be possible in a non-rushing scenario where all the parties are assumed to maintain a *completely* honest behavior, and using *un-biaised* pseudo-random generators. In our case, as sketched in section 2 a player may decide to choose his contribution *after* having received all those of the remaining parties. Thus he could, arbitrarily, set the value for the final key. In order to avoid such a situation, in our protocol we distinguish two stages: during the first one every player sends encryptions of a_i and waits for all the other guys to do the same. Then, once he has received all the shares he proceeds to the second stage by disclosing his $f(i)$. Such such a two round separation has the effect of forcing the players to choose their r_i without having any clue (in a strong information-theoretic sense) about the r_i's chosen by the remaining players. In this way the produced key is uniformly distributed if at least one of the players chooses his contributions uniformly and at random (and we stress that in our model we assume that at least one player mantains a fully honest behavior).

In practice, we implement this idea by assuming the second round starts when each player has received *all* the $n-1$ contributions from the remaining parties. The underlying intuition is that if a player has received $n-1$ ciphertexts, then he can safely start the second round, because he is ensured that every other party has already chosen his own share. Interestingly this approach allows for some modularity in error detection. Indeed, if at least one player aborted after round one (for instance, because he did not correctly receive all the expected ciphertexts) such a situation can be efficiently detected in round two as follows. If after round one some party aborted (i.e. quit) the protocol then the remaining players cannot reconstruct the polynomial $f(\cdot)$ — simply because not enough shares are available — and the protocol can be immediately aborted. On the other hand the fact of receiving all the expected shares in round two, can be seen as a "so far, so good" guarantee: if a player sends his share

Authenticated Group Key Agreement Protocol

Public Parameters: Two primes p, q such that $q|p-1$. A subgroup $Q = \langle g \rangle$ of order q. An hash function $\mathcal{H}$ modeled as a random oracle, and $\mathcal{ID}$ be the current session ID.
Public inputs: The players'public keys h_i, for $i = 1, \ldots, n$.
Private input (for player i): A value x_i such that $h_i = g^{x_i} \bmod p$.
In a preprocessing stage player P_i runs a signature generation algorithm `SigGen` to obtain a couple of matching signing and verification keys (SK_i, VK_i).

First Round – Each player P_i does the following:

1. Choose $a_i \leftarrow Q$
2. Choose $r_i, b_{i,1}, \ldots, b_{i,n-1} \leftarrow \mathbb{Z}_q$.
3. Define $f_i(z) = r_i + b_{i,1}z + \ldots + b_{i,n-1}z^{n-1} \bmod q$
4. For each $j = 1 \ldots n$ $(j \neq i)$
   Choose $k \leftarrow \mathbb{Z}_q$ and set $C_{i,j} = (A_{i,j}, B_{i,j}) = (g^k \bmod p, h_j^k a_i \bmod p)$.
5. Send to player P_j the values $C_{i,j}$, $f_i(j)$ and $\sigma_{i,j} = \mathtt{Sign}_{SK_i}(C_{i,j}||f_i(j)||\mathcal{ID})$.

Second Round – Once having received all the values above each player P_i does the following: (if P_i receives less than $n-1$ triplets $(C_{j,i}, f_j(i), \sigma_{j,i})$ he aborts the protocol)

1. Check the authentication (signature) of all received values. If the check fails the player aborts the protocol.
2. Multiply the received ciphertexts: let $A_i = \prod_{j \neq i} A_{j,i} \bmod p$ and $B_i = a_i \cdot \prod_{j \neq i} B_{j,i} \bmod p$.
3. Decrypt the result to define the value $a_{(i)} = B_i / A_i^{x_i}$.
4. Compute
$$f_i = f_i(i) + \sum_{j \neq i} f_j(i) \bmod q$$
as his share of a $(n-1)$-degree polynomial $f(z)$ whose free term we indicate with r.
5. Send to other players the values f_i and $\omega_i = \mathtt{Sign}_{SK_i}(f_i||\mathcal{ID})$.

Third Round –

1. The players interpolate $f(z)$ and retrieve r.
2. Player P_i defines its session seed as
$$sk_{(i)} = a_{(i)} \cdot g^r \bmod p.$$

Confirmation Step: Compute $s_i = \mathcal{H}(sk_{(i)}||\mathcal{ID})$ and broadcast this value together with its signature $\gamma_i = \mathtt{Sign}_{SK_i}(s_i||\mathcal{ID})$.
If the n broadcasted values are all the same, set the final key as
$$sk = \mathcal{H}(sk_{(i)})$$

Fig. 1. Pseudo-code description for the Group Key Agreement Protocol

of the polynomial, it means that he must have been happy with what he has received so far.

Disclosing the polynomial's shares – We notice that in the first round (step 5), a player can safely send his "own" shares $f_i(j)$ (the shares for his private polynomial f_i), without encrypting them, since this does not reveal any information at all about his randomizer $r_i = f_i(0)$: in fact the value $f_i(i)$ is *never* disclosed at any time. Moreover, and for the same reason, it is important to note that the entire transcript of the first round does not reveal anything about the "global" shares $f(i)$ neither. More precisely, recall that the "global" share for player P_i (i.e., his share of $f(\cdot)$) is defined as $f(i) = \sum_{j=1}^{n} f_j(i)$, but only the values $f_{j \neq i}(i)$ are disclosed. Thus, until the second round, step 5, all "global" shares $f(i)$ are still *information-theoretically* hidden to the adversary, and each player P_i knows exactly $f(i)$, that is, no more at all about other $f(j)$'s. During the second round, the shares will be disclosed (keep in mind that is done once the contributions a_i's have been received by each player).

Key confirmatory – There remains one final problem to discuss in our protocol, because of which we yet need a confirmation step at the very end of round two. Actually, to be sure that all parties correctly recover the session key, we use the following technique, known as *key confirmatory*, that allows each player to check that the remaining participants have computed the key. This additional confirmation step makes use of an asynchronous broadcast, which means that we need to assume that the network is equipped with such primitive. Using a broadcast, either everybody received the confirmation messages, or nobody did and the protocol aborts.

We can obtain the key confirmatory property by having each player computing and broadcasting an additional value to other players. Every player sends a "receipt" which is computed from the session key, thus playing the role of an authenticator. The technique is described with more details in [9] and requires the assumption of random oracle[3] in order for the authenticator not to leak any information about the session key. In particular, such an authenticator, should not be computed directly from the final session key, but rather from an intermediate common secret (otherwise, an eavesdropper would be able to gain some partial information about sk — for instance the hash of sk — and to distinguish it from a random string).

4 Security of the scheme

In this section we prove the following security theorem.

Theorem 1. *The protocol presented in figure 1 is a secure Authenticated Group Key Agreement protocol, achieving completeness, contributory and privacy (under the DDH assumption).*

Completeness – Obvious by inspection.

[3] Actually the random oracle is considered for efficiency reasons only and it is not necessary for the [9] technique to work. In particular the random oracle can be replaced by a *pseudo-random* function [21].

Privacy – We consider a simulator $\mathcal{S}$ that *emulates* the protocol to the adversary in such way that the simulation is indistinguishable from the real attack. Formally the simulator goes as follows. It receives as input a triplet (X, Y, Z), for which it has to decide whether it is a Diffie-Hellman triplet or not. Let $\mathcal{Q}$ be the total number of interactions the adversary is likely to make. $\mathcal{S}$ starts by choosing at random an integer q_0 in $[1, \mathcal{Q}]$, hoping the q_0-th interaction will be the attacked session. Then it chooses uniformly and at random an index i_0 in $[1, n]$. After that, it initializes the protocol by choosing n random exponents ξ_1 through ξ_n. It sets the public key $h_i = Y g^{\xi_i} \bmod p$ for every player P_i. Finally it gives g and all the h_i's to the adversary $\mathcal{A}$, as the public input of the protocol.

To take into account the rushing scenarios, we consider two different pseudo-random generators $\mathcal{R}$ and $\mathcal{R}^\star$, assuming the latter is biased. In particular, $\mathcal{R}^\star$, when called by a player, takes as input all previous data used by this player. We denote, to formalize our simulation, by $\mathcal{R}_j$ the pseudo-random generator used by P_j, and we set $\mathcal{R}_{i_0} = \mathcal{R}$ and $\mathcal{R}_j = \mathcal{R}^\star$ for all $j \neq i_0$.

Then (for each parallel sessions), the simulator $\mathcal{S}$ simulates the players in the first round as follows. On receiving a $\mathsf{Send}(U_j, \mathtt{start})$-query, the simulator chooses a secret contribution a_j and n coefficients $r_j, (b_{j,k})_{1 \leq k \leq n-1}$, using the pseudo-random generator $\mathcal{R}_j$. The ciphertexts in step 4 are computed straightforwardly, except if $j = i_0$ and $q = q_0$. In that later case, the simulator chooses (uniformly) $n-1$ random values ρ_j (for $j = 1, \ldots, n$ but $j \neq i_0$) and computes $A_{i_0,j} = X g^{\rho_j}$ and $B_{i_0,j} = Z Y^{\rho_j} X^{\xi_j} g^{\rho_j \xi_j} a_{i_0}$ as an encryption of a_{i_0}. The query is answered with the $n-1$ (signed) flows to be sent to others.

The second round starts (for player U_j) after having received $n-1$ queries, from $n-1$ other players (within a given concurrent session). Before that, the simulator just stores the received flows. The simulator checks the authenticity of the received flows, then defines $a_{(j)}$ as the product of all a_i. Note, in particular, $\mathcal{S}$ does not perform the multiplication of ciphertexts (step 2), nor the decryption (step 3), since it does not know the private key $x_j = \log_g h_j$. Steps 4 and 5 of round 2 are performed straightforwardly, and the query is finally answered by f_j, together with its signature.

Round 3 is simulated as in the real protocol. The confirmation step is processed straightforwardly. After the third Round, a Reveal-query is answered straightforwardly, except if asked to U_{i_0} within the q_0-session. In that case $\mathcal{S}$ aborts.

If the Test-query does not occur within the q_0-th session, the simulator aborts. This happens with probability at most $(\mathcal{Q}-1)/\mathcal{Q}$. Otherwise, it is processed as follows. Let $\kappa = a \cdot g^r = \prod_{i=1}^n a_i \cdot g^{\sum_{i=1}^n r_i}$. When the Test-query occurs, the simulator flips a private coin β and set $\kappa_0 \leftarrow \mathcal{K}, \kappa_1 = \kappa$, where $\mathcal{K}$ is the session key space (and in our case $\mathcal{K} = Q$). Then it gives κ_β to the adversary. The interaction might continue then; at the end of the attack, the adversary answers with a bit b', that the simulator relays back as its own guess. The theorem then follows immediately from the following two claims.

Claim 1 *If the simulator's input is a Diffie-Hellman triplet (that is $b = 1$) the adversary's view is perfectly indistinguishable from the real protocol.*

It is easy to see that, in this case, the simulation is perfectly identical to the real protocol with player P_i using private contribution a_i, and thus the value κ is actually

the session key sk. This means that an infinitely powerful adversary, which would be able to recover all plaintexts, would necessarily lead to $sk = \kappa$. Indeed, the secret key of player P_j is implicitly $y + \xi_j$, where $y = \log_g Y$. And any ciphertext $C_{i_0,j}$ is an honest encryption of a_{i_0}, using randomness $x + \rho_j$, where $x = \log_g X$. Of course, any other $C_{i,j}$ is an encryption of a_i under public key h_j.

Then we have ($\mathcal{A}_\mathcal{V}$ denotes the adversary together with its view):

$$\begin{aligned}
\Pr\left[\mathcal{A}_\mathcal{V}(\kappa_\beta) = 1 | \beta = 1 \wedge b = 1\right] &= \Pr\left[\mathcal{A}_\mathcal{V}(\kappa_\beta) = 1 \,\middle|\, \beta = 1 \wedge \kappa_1 = sk\right] \\
&= \Pr\left[\mathcal{A}_\mathcal{V}(\kappa_1) = 1 \,\middle|\, \kappa_1 = sk\right] \qquad (1) \\
\Pr[\mathcal{A}_\mathcal{V}(\kappa_\beta) = 1 | \beta = 0 \wedge b = 1] &= \Pr\left[\mathcal{A}_\mathcal{V}(\kappa_\beta) = 1 \,\middle|\, \beta = 0 \wedge \kappa_0 \leftarrow \mathcal{K}\right] \\
&= \Pr\left[\mathcal{A}_\mathcal{V}(\kappa_0) = 1 \,\middle|\, \kappa_0 \leftarrow \mathcal{K}\right] \qquad (2)
\end{aligned}$$

Then using the fact that $\Pr[\beta = 1] = \Pr[\beta = 0] = 1/2$, we have:

$$\Pr\left[\mathcal{A}_\mathcal{V}(\kappa_\beta) = 1 | b = 1\right] = \frac{1}{2}\Pr\left[\mathcal{A}_\mathcal{V}(\kappa_1) = 1 | \kappa_1 = sk\right] + \frac{1}{2}\Pr\left[\mathcal{A}_\mathcal{V}(\kappa_0) = 1 | \kappa_0 \leftarrow \mathcal{K}\right]$$

Claim 2 *If the simulator's input is a random triplet (that is $b = 0$) the adversary's view is independent from a_{i_0}.*

In such a case, all the values are correctly computed, except that the ciphertexts $C_{i_0,j}$ encrypt random values. More precisely, the value computationally hidden in $C_{i_0,j}$ under public key $h_j = Yg^{\xi_j}$ is (implicitly):

$$\hat{a}_{i_0,j} = \frac{B_{i_0,j}}{(A_{i_0,j})^{y+\xi_j}} = \frac{ZY^{\rho_j}X^{\xi_j}g^{\rho_j\xi_j}a_{i_0}}{(Xg^{\rho_j})^{y+\xi_j}} = \frac{g^{z+y\rho_j+x\xi_j+\rho_j\xi_j}a_{i_0}}{(g^{x+\rho_j})^{y+\xi_j}} = g^{z-xy}a_{i_0}$$

where $z = \log_g Z$. Note that this value does not depend from the index j of the receiver. This is due to the fact we use the *additive* random self-reducibility property of the Diffie-Hellman problem.

Consequently, the plaintext that an infinitely powerful adversary would recover by decrypting all the ciphertexts is (for any j): $\hat{a}_{(j)} = g^{z-xy}\prod_k a_k = g^{z-xy}a_{(j)}$; thus, the adversary learns no information at all about $a_{(j)}$ when eavesdropping the messages. According to adversary's view, the session key sk associated to this simulated execution of the protocol is thus

$$\hat{a} \cdot g^{\sum_{j=1}^{n} r_j} = g^{z-xy}a \cdot g^r.$$

On the other hand, the simulation makes all players setting their key to $a_{(j)}$. Then, the value "recovered" (according to the simulation) by every player P_i, including P_{i_0}, is $\kappa_1 = ag^r$; moreover a_{i_0} and, thus κ_1, is uniformly distributed over Q, exactly as κ_0 is. Consequently, the value of β is information-theoretically hidden to $\mathcal{A}$.

$$\Pr\left[\mathcal{A}_\mathcal{V}(\kappa_\beta) = 1 | b = 0\right] = \frac{1}{2}\Pr\left[\mathcal{A}_\mathcal{V}(\kappa_1) = 1 | \kappa_1 \leftarrow \mathcal{K}\right] + \frac{1}{2}\Pr\left[\mathcal{A}_\mathcal{V}(\kappa_0) = 1 | \kappa_0 \leftarrow \mathcal{K}\right]$$

By subtraction, we get:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{ddh}}(\mathcal{S}) &= \Pr\left[b'=1|b=1\right] - \Pr\left[b'=1|b=0\right] \\ &= \Pr\left[\mathcal{A}_{\mathcal{V}}(\kappa_\beta)=1|b=1\right] - \Pr\left[\mathcal{A}_{\mathcal{V}}(\kappa_\beta)=1|b=0\right] \\ &= \frac{1}{2}\left(\Pr\left[\mathcal{A}_{\mathcal{V}}(\kappa_1)=1\middle|\kappa_1=sk\right] - \Pr\left[\mathcal{A}_{\mathcal{V}}(\kappa_1)=1\middle|\kappa_1\leftarrow\mathcal{K}\right]\right) + \frac{1}{2}\times 0 \\ &= \frac{\mathsf{Adv}(\mathcal{A})}{2}\end{aligned}$$

Assuming the DDH assumption, this quantity is a negligible amount.

In fact, we have conditioned by the fact the Test-query has been correctly guessed, so we must divide by $1/\mathcal{Q}$.

Contributory – Contributory trivially follows from the fact that every player is forced to choose his share a_i having no information at all (in an information theoretic sense!) about the actual value of the randomizer r.

5 Comments, Optimizations and Variants

Efficiency of the protocol – Our protocol is very efficient both in terms of bandwidth and in term of number of rounds required. The number of bits sent by each player is bounded by $3|p|n$ plus $n+2$ times the size of the employed signature scheme (used for the authentication). The protocol requires 2 rounds of communication, one asynchronous broadcast for the confirmation step and roughly $2n$ exponentiations per player (plus the cost of computing the signatures). If precomputations are possible (in a context where, for example, the participants public keys are all known in advance), all the exponentiations in Round 1 can be done off-line and the number of total exponentiations (per player) reduces to 2 (plus the cost of the signatures, and the cost of multiplying the received ciphertexts of course). To our knowledge, in this case (and for this specific aspect) our scheme is one of the most efficient group key agreement solutions known. Moreover, being a constant round protocol, it has the property that the number of "slow guys" is not a major efficiency issue. Indeed, a n round protocol is like a token ring network: a player does its work then passes the token to the next one; hence the delays induced by slow parties go cumulating. In our case, everybody works in parallel so we have the same delay whatever the number of slow guys is (more precisely, the delay is essentially that of the slowest guy).

Considering forward-secrecy – The forward-secrecy property [10, 17, 29] encompasses that the privacy (semantic security) of the session key is not compromised even in case of a further leakage of the long-term El-Gamal key. In other words, if the adversary learns a private key x_i at some time, then the knowledge of x_i, as well as the view of previous session key establishments, does not help him to get information about these previously established session keys. We state informally that our protocol provides forward-secrecy if at most one private key, say x_1, is revealed. Indeed, if an adversary $\mathcal{A}$ knows x_1, it can decrypt all ciphertexts sent to P_1, thus learning all contributions $a_2, \ldots, a_n$. However, P_1's contribution, namely a_1, is never encrypted under $h_1 = g^{x_1}$, and then, remains (computationally) hidden to $\mathcal{A}$, and so does the session

key. In order to cover larger scenarios, we have to consider forward-secure public-key encryption schemes [14].

Resistance to known-key attacks – A key exchange protocol is said to be resistant to known-key attacks [32] if the exposure of a session key gives no advantage to an adversary for breaking the privacy of future session keys. This property takes some importance in dynamic groups, in which future session keys are computed from private data among which is the current session key. Our protocol trivially provides resistance to such attacks, since all values are one-time used and picked ("fresh") at the beginning of a key exchange.

A General Solution – Up to some loss in efficiency it is possible to generalize our construction in order to obtain a constant round authenticated group key agreement scheme provably secure under the sole assumption that trapdoor functions exist (indeed, this assumption ensures that a semantically secure encryption scheme and a secure signature scheme exist). Details will appear in the final version of this paper.

6 Conclusions

In this paper we presented a new protocol that achieves strong properties of efficiency and security under standard assumptions. The protocol is efficient both in communication rounds and in bandwidth: the number of communication rounds is constant, and the bandwidth is comparable with that of previously proposed schemes. Our scheme is provably secure under the Decisional Diffie-Hellman assumption, and enjoys several additional properties such as forward-secrecy or an increased efficiency when preprocessing is allowed. An intriguing, still open, research problem is to establish a secure key agreement scheme that provides some kind of "resistance" with respect to active adversaries (i.e. for example a protocol that allows to the non corrupted players to eliminate the bad guys and to agree on a key.

Acknowledgments. We wish to thank Jacques Stern for valuable comments on a early version of this paper. We thank David Pointcheval for a number of helpful discussions we had.

References

1. G. Ateniese, M. Steiner and G. Tsudik. New multi-party authentication services and key agreement protocols. *IEEE Selected Areas in Communications*, **18**(4): 628–639, 2000.
2. M. Bellare, R. Canetti and H. Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols. In *STOC '98*, pp. 419–428, 1998.
3. M. Bellare, D. Pointcheval and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *Eurocrypt '00*, *LNCS* 1807, pp. 139–155. Springer, 2000.
4. M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *ACM CCS '93*, pp. 62–73. 1993.
5. M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Crypto '93*, *LNCS* 773, pp. 232–249. Springer, 1993.
6. M. Bellare and P. Rogaway. Provably secure session key distribution: The three party case. In *STOC '95*, pp. 57–66, 1995.

7. S. Blake-Wilson, D. Johnson and A. Menezes. Key agreement protocols and their security analysis. In *Conf. on Cryptography and Coding*, *LNCS* 1355, pp. 30–45. Springer, 1997.
8. S. Blake-Wilson and A. Menezes. Authenticated Diffie-Hellman key agreement protocols. In *SAC '98*, *LNCS* 1556, pp. 339–361. Springer, 1998.
9. E. Bresson, O. Chevassut and D. Pointcheval. Provably authenticated group Diffie-Hellman key exchange – the dynamic case. In *Asiacrypt '01*, *LNCS*, pp. 290–309.
10. E. Bresson, O. Chevassut and D. Pointcheval. Dynamic group Diffie-Hellman key exchange under standard assumptions. In *Eurocrypt '02*, *LNCS* 2332, pp. 321–336.
11. E. Bresson, O. Chevassut and D. Pointcheval. The group Diffie-Hellman problems. In *SAC '02*, LNCS 2595, pp. 325–338. Springer, 2002.
12. M. Burmester and Y. G. Desmedt. A secure and efficient conference key distribution system. In *Eurocrypt '94*, *LNCS* 950, pp. 275–286. Springer, 1995.
13. C. Cachin, K. Kursawe, F. Petzold, and V. Shoup. Secure and efficient asynchronous broadcast protocols. In *Crypto '01*, *LNCS* 2139, pp. 524–541. Springer, 2001.
14. R. Canetti, S. Halevi and J. Katz. A forward-secure public-key encryption scheme. In *Eurocrypt '2003*, *LNCS* 2656, pp. 255–271, Springer, 2003.
15. R. Cramer V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Crypto '98*, *LNCS* 1462, pp. 13–25. Springer.
16. W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, vol. 22(6): pp. 644–654, November 1976.
17. W. Diffie, P. van Oorschot and W. Wiener. Authentication and authenticated key exchange. In *Designs, Codes and Cryptography*, vol. 2(2), pp. 107–125, June 1992.
18. W. Diffie, D. Steer, L. Strawczynski and M. Wiener. A secure audio teleconference system. In *Crypto '88*, *LNCS* 403, pp. 520–528. Springer, 1988.
19. R. Dupont and A. Enge. Practical non-interactive key distribution based on pairings. Cryptology eprint archive, `http://eprint.iacr.org`, May 2002.
20. T. ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. *IEEE IT*-31(4):469–472, 1985.
21. O.Goldreich, S. Goldwasser and S. Micali. How to Construct Random Functions. *JACM* **33**(4):792-807, 1986.
22. S. Goldwasser and S. Micali. Probabilistic Encryption. *JCSS*, 28(2):270–299, April 1984.
23. I. Ingemarsson, D. T. Tang and C. K. Wong. A conference key distribution system. *IEEE Transactions on Information Theory*, vol. IT-28(5): pp. 714–720, September 1982.
24. A. Joux. A one-round protocol for tripartite Diffie-Hellman. In *Proc. of ANTS-4 Conference*, vol. 1838 of *LNCS*, pp. 385–394, Springer-Verlag, 2000.
25. J. Katz and M. Yung. A fully scalable Protocol for Authenticated Group Key Exchange. In *Crypto '03*, to appear.
26. C.-H. Li and J. Pieprzyk. Conference key agreement from secret sharing. In *Proc. of ACISP '99*, vol. 1587 of *LNCS*, pages 64–76. Springer-Verlag, 1999.
27. D. Malkhi, M. Merrit and O. Rodeh. Secure Reliable Multicast Protocols in a WAN. In *Proc. of International Conference on Distributed Computing Systems*, pp. 87–94, 1997.
28. D. Malkhi and M. Reiter. A High-Throughput Secure Reliable Multicast Protocol. *J. of Computer Security*, vol. 5, pp. 113-127, 1997.
29. A. J. Menezes, P. V. Oorschot and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997. `http://cacr.math.uwaterloo.ca/hac/`.
30. A. Shamir. How to share a secret. In *CACM*, **22**:612–613. November 1979.
31. V. Shoup. On formal models for secure key exchange. Tech. Rep. RZ 3120, IBM Zürich Research Lab, November 1999.
32. M. Steiner, G. Tsudik and M. Waidner. Key agreement in dynamic peer group. *IEEE Transactions on Parallel and Distributed Systems*, vol. 11(8): pp. 769–780, August 2000.