On the Computational Security of a Distributed Key Distribution Scheme

Vanesa Daza[1], Javier Herranz[2] and Germán Sáez[3]

***Abstract*— In a distributed key distribution scheme, a set of servers help a set of users in a group to securely obtain a common key. Security means that an adversary who corrupts some servers and some users has no information about the key of a non-corrupted group.**

In this work we formalize the security analysis of one of such schemes [11], which was not considered in the original proposal. We prove the scheme secure in the random oracle model, assuming that the Decisional Diffie-Hellman problem is hard to solve. We also detail a possible modification of that scheme and the one in [24], which allows to prove the security of the schemes without assuming that a specific hash function behaves as a random oracle. As usual, this improvement in the security of the schemes is at the cost of an efficiency loss.

***Index Terms*— Cryptography, key distribution, secret sharing schemes, provable security.**

I. Introduction

There are many situations where a group (or *conference*) of users need a common secret key, in order to use it for some purpose: for secure communication by using symmetric encryption techniques; to allow the access to some restricted resource, such as forums in the Internet or multi-player computer games; in general, to identify the members of the conference in front of themselves or in front of external entities.

There are some different ways in which these users can securely obtain this common key, depending on the resources available to them. For example, in group key exchange protocols [7] the users broadcast partial values that allow all of them, after a certain number of rounds of communication, to securely compute the same group key. Current schemes [5], [22] work with only three or four rounds of communication.

Group key distribution schemes [12], [23] are also managed by the users of the group themselves, but are more suitable for dynamic situations, where users can join or leave the group at any period of time, which is decided by the own group members. This dynamism capability makes these schemes slightly different to group key exchange protocols, which must be re-initialized every time that the composition of the group changes.

A drawback of these two approaches is that there must be some kind of synchronization among the users, because they all must execute the protocol at the same time. In some scenarios this can be a problem, and it is desirable that every user could obtain the conference key whenever he wants.

To achieve this property, Needham and Schroeder [25] proposed to consider a key distribution center which computes and stores the conference keys, and sends them to the users when they contact him. However, this solution has some drawbacks: the center is a bottleneck of the system, a possible point of failure and a clear target for attacks, because he knows all the keys of the system.

To overcome these problems, Naor, Pinkas and Reingold [24] introduced the notion of *distributed key distribution schemes*: the task of the key distribution center is shared among a set of servers, who independently send some information to the users which allows them to compute the conference key. In a nutshell, the protocol is divided into three different phases. In the *initialization phase*, servers jointly generate some shared secret information. Later, in the *key request phase*, a user asks some servers for a conference key; servers check that this user actually belongs to the conference and, if so, each of them uses his share of the secret information to compute a partial conference key. In the *key delivery phase*, the user combines the received information to obtain the final conference key. A necessary property is that all the users in the same conference must obtain the same key, independently of the set of servers that they contact.

a) Related work: Distributed key distribution schemes have been considered in both information-theoretic [4], [10], [24] and computational [11], [13], [24] frameworks. The first proposal of distributed key distribution schemes in a computational framework was given in the original paper [24], as a natural application of the concept of threshold pseudo-random functions. Although the specific security of distributed key distribution schemes was not considered there, they proved that their threshold pseudo-random functions satisfy some properties (in the random oracle model, and under the Decisional Diffie-Hellman Assumption), which directly imply that the resulting distributed key distribution schemes are secure in the formal model defined some years later. The proposal in [24] pursues to increase server availability and decrease network communication; it does not take very much into consideration the computational costs for servers and users.

In [11], an alternative scheme was proposed, with the goal of reducing the computational load on the users, which was quite heavy in the proposals of [24]. This improvement is at the price of increasing the computational and communication costs for the servers. The idea is that this different way of distributing the costs of the scheme can make more sense in practice, because servers are assumed to be powerful devices, whereas users can be mobile devices (PDAs, mobile phones,...) with low computational resources. In such situations, a user can broadcast a query for a key to a set of servers, and expects to receive the key quite directly, without the necessity of performing some extra operations to obtain it (other than decrypting). On the other hand, servers can be in some way paid for doing most of the work. However, the proposal in [11] did not include any formal security analysis.

The first explicit and formal security model for distributed

[1] Universitat Rovira i Virgili, Tarragona, Spain
[2] IIIA-CSIC, Bellaterra, Spain
[3] Universitat Politècnica de Catalunya, Barcelona, Spain

key distribution schemes in the computational framework was given in [13]. Roughly speaking, a distributed key distribution scheme is secure if an adversary who corrupts some subset of servers and some conferences of users has no information about the key of a different, and non-corrupted, conference. In the same work, a general construction of secure distributed key distribution schemes starting from distributed signature schemes was provided.

b) Our contribution: In this work, we formally prove that the scheme in [11] enjoys the security properties required for distributed key distribution schemes. The proof is by reduction to the hardness of the Decisional Diffie-Hellman (DDH) problem, in the random oracle model. We consider malicious adversaries, and so we add mechanisms to detect dishonest participants, such as verifiable secret sharing and zero-knowledge proofs of knowledge.

We also detail how to modify a part of the scheme in order to avoid the use of a specific hash function which is modelled as a random oracle in the security proofs. This modification was already mentioned (briefly) in [24], and is valid for the scheme proposed there, as well. As it usually happens, the improvement in the security of the schemes is at the cost of a reduction in their efficiency, because now the servers must maintain a table where each conference is assigned one value jointly chosen by them.

c) Organization of the paper: In Section II we describe the framework of distributed key distribution schemes: the protocols that define such schemes and the formal security requirements that they must satisfy. Then in Section III we recall ElGamal encryption and a protocol for the joint generation of a shared secret value, because they are essential tools of the scheme that we study. In Section IV we review the distributed key distribution scheme proposed in [11] and then we analyze it: we add the security analysis of the employed protocol to compute proofs of knowledge, which provides robustness to the scheme, and we prove that the whole scheme is secure in the random oracle model. We propose in Section V a slight modification to the schemes in [11], [24], which allows us to prove their security without the (sometimes too strong) assumption that a hash function behaves as a random oracle. Finally, we devote Section VI to the conclusions of this work.

II. Distributed Key Distribution Schemes

Naor, Pinkas and Reingold introduced the notion of distributed key distribution schemes in [24]. In this kind of schemes, we have a set $\mathcal{U} = \{U_1, \ldots, U_m\}$ of m users and a set $\mathcal{S} = \{S_1, \ldots, S_n\}$ of n servers. Users in the same *conference* $C \subset \mathcal{U}$ want to obtain a common secret key κ_C, for example in order to communicate securely with the other members of the conference, or to use it as an access key for some restricted resources. To obtain κ_C, users must interact with some servers in $\mathcal{S}$. The subsets of servers that are allowed to provide conference keys are those in the *access structure* $\Gamma \subset 2^{\mathcal{S}}$, which must be monotone increasing: if $A_1 \subset A_2$ and $A_1 \in \Gamma$, then $A_2 \in \Gamma$.

In more detail, a distributed key distribution schemes consists of the following phases:

- **Initialization Phase.** This phase involves only the servers, who interact with each other. The input is the access structure Γ, and the output contains some public information, and a partial secret information for each server $S_i \in \mathcal{S}$.
- **Key Request Phase.** In this phase, a user U_j contacts a subset of servers in order to obtain a conference key for some conference C to which he belongs. Servers send to U_j some value(s) which depends on their partial secret information, on the identity of U_j and on the conference C.
- **Key Delivery Phase.** Finally, the user U_j combines the received values and tries to compute the conference key κ_C for the conference C. If the contacted subset of servers belongs to the access structure Γ, then U_j is always able to compute the correct κ_C.

1) Communication Model: We assume the following communication model for the distributed key distribution scheme that we study in this paper. On the one hand, communication between servers is done through secret and authenticated channels, not necessarily synchronous. On the other hand, communication between servers and users requires authenticated channels, but does not require secret or synchronous channels.

A. Security of Distributed Key Distribution Schemes

The level of security required for distributed key distribution schemes depends on the scenario where the conference members use the generated key, and basically on the type of attacks the application must be protected against. A high enough (and standard) level of security is the equivalent of *semantic security* for encryption schemes [20]. Roughly speaking, an adversary should not be able to distinguish a conference key κ_C from a value taken at random from the set of possible keys, even if he corrupts some subset of servers and the protocol is executed for other conferences. The adversary can require the execution of the protocol for conferences of his choice, as many times as he wants; this corresponds to a dishonest user who starts many sessions of the same application, with different conferences.

1) The Adversarial Model: Obviously, the number of servers that an adversary can corrupt must be limited; otherwise, if he controls for example an authorized set of servers, in Γ, he can compute all the conference secret keys, and there is no security at all. Therefore, an adversary structure $\mathcal{A} \subset 2^{\mathcal{S}}$ must be defined, containing those subsets of dishonest servers that the system is able to tolerate. It must be monotone decreasing: if a subset B can be corrupted by $\mathcal{A}$, then any subset contained in B can also be corrupted by $\mathcal{A}$. A trivial condition to ensure security of the schemes in this scenario is that $\mathcal{A} \cap \Gamma = \emptyset$.

We deal with *static* but *malicious* adversaries. Static means that the set of corrupted servers is chosen at the beginning of the life of the system, and remains fixed from that moment on. Malicious means that the adversary can put together the private information of the corrupted parties, and can force them to deviate from the correct execution of the protocols. In this case, a standard requirement for the designed protocols is that of *robustness*: there must be mechanisms to detect the corrupted parties who do not follow the protocol correctly; once this is done, there must remain enough honest parties to successfully finish the execution of the protocol. A necessary condition to ensure robustness is $\mathcal{A}^c \subset \Gamma$, where $\mathcal{A}^c = \{\mathcal{P} - B \mid B \in \mathcal{A}\}$. Throughout the paper, since we want robustness, we will assume that this condition is satisfied by the structures considered in the protocols.

2) Formal Security Model for Distributed Key Distribution Schemes: The adversary against the distributed key distribution scheme will be denoted as $\mathcal{F}$. Such an adversary $\mathcal{F}$ is given the public parameters of the system: the set of servers $\mathcal{S} = \{S_1, \ldots, S_n\}$, the access and adversary structures $\Gamma \subset 2^{\mathcal{S}}$ and $\mathcal{A} \subset 2^{\mathcal{S}}$, and the set of users $\mathcal{U} = \{U_1, \ldots, U_m\}$.

Extending the security model introduced in [13], inspired by other works on key distribution (see [1], for example), we describe here a game $\mathcal{G}$, played by an adversary $\mathcal{F}$ against a distributed key distribution scheme. We consider schemes which fall inside the model explained in this work, with an initialization phase performed only by servers, a key request and computational phase where servers and users interact, and a key delivery phase where users compute the conference keys. The game $\mathcal{G}$, which captures the capabilities (requesting the execution of the protocol, step 3) and the goals (distinguishing a conference key from a random key, steps 4-6) of the adversary $\mathcal{F}$, works as follows.

1) The adversary chooses a subset of servers $B \in \mathcal{A}$ to be corrupted.
2) The initialization phase of the distributed key distribution scheme, which involves only servers, is executed. The adversary $\mathcal{F}$ obtains all the public information and the secret information of the corrupted servers $S_j \in B$.
3) The adversary $\mathcal{F}$ adaptively chooses Q_{conf} different conferences $C' \subset \mathcal{U}$. Key request and computational phase of the scheme, as well as the key delivery phase, are executed for each user of these conferences, as many times as $\mathcal{F}$ wants. As a result, $\mathcal{F}$ obtains all the information produced in these executions. One could think that the adversary has corrupted all members of such conferences C'. $\mathcal{F}$ obtains the private information generated by the corrupted servers, the information made public by all the servers, the information obtained by users in C' (including the resulting conference key $\kappa_{C'}$), etc.
4) The adversary chooses a conference C different from the conferences it has queried in the previous step. For this conference, the key request and computational phase of the scheme is executed. The adversary $\mathcal{F}$ obtains the private information of the corrupted servers and the information broadcast by all the servers, but it has not access to the secret information received by users in C.
   This step 4) can be executed in parallel to step 3), to allow concurrent attacks. Furthermore, more key conference queries can be made by the adversary after this step, for conferences $C' \neq C$, which are answered as in step 3).
5) A random bit $b^* \in \{0,1\}$ is chosen by a challenger. If $b^* = 1$, the adversary $\mathcal{F}$ is given the real key κ_C. If $b^* = 0$, the adversary is given an element taken uniformly at random from the set of possible conference keys.
6) The adversary $\mathcal{F}$ outputs a bit $b' \in \{0,1\}$, with $b' = 1$ if $\mathcal{F}$ thinks that the received element is actually κ_C, and $b' = 0$ otherwise. It wins the game if $b' = b^*$.

We define the *advantage* of such an adversary $\mathcal{F}$ in breaking the semantic security of the distributed key distribution scheme as

$$\varepsilon = \Pr[b' = b^*] - \frac{1}{2}.$$

Definition 1: An adversary $\mathcal{F}$ $(\mathcal{U}, \mathcal{S}, \Gamma, \mathcal{A}, T, \varepsilon, Q_{conf})$-breaks the distributed key distribution scheme if it plays game $\mathcal{G}$ in time at most T, and its advantage in breaking the semantic security of the scheme is at least ε.

Regarding the protocol of Naor et al. in [24], they actually propose *threshold pseudo-random functions* (TPRFs), which in particular can be easily transformed into distributed key distribution schemes. They define the security requirements for such functions, and prove that their proposal of TPRF satisfies these requirements, in the random oracle model, under the Decisional Diffie Hellman (DDH) Assumption. It is easy to see that this fact directly implies that the resulting distributed key distribution scheme is secure in this formal model.

For simplicity, we have decided to concentrate on the case of one single session for each conference. In the case of the studied scheme, one possibility to deal with different sessions s would be to include another input to the hash function H; that is, the conference key for C in the session s would be $\kappa_{C,s} = H(C,s)^\alpha$. Since in the security proof we will assume that H behaves as a random oracle, this solution would be secure. The same technique could be applied to the scheme in [24], in order to allow different sessions.

III. Building Blocks

In this section we review two cryptographic primitives that will appear in the explanation of the proposal of distributed key distribution scheme that we study in this work.

A. ElGamal Encryption

In [17], ElGamal proposed a public-key probabilistic encryption scheme. The public parameters of the scheme are two large primes p and q, such that $q|p-1$, and an element g verifying that the multiplicative subgroup $\mathbb{G} = \langle g \rangle$ of $\mathbb{Z}_p^*$ has order q (this mathematical framework will be common to all the protocols explained in this work).

Every user U generates both his public and private keys by choosing a random element $x \in \mathbb{Z}_q^*$ and computing $y = g^x \mod p$. The public key of user U is y and his private key is x.

If someone wants to encrypt a message $m \in \mathbb{G}$ for user U, he chooses a random element $\beta \in \mathbb{Z}_q^*$, and computes $r = g^\beta \mod p$ and $s = my^\beta \mod p$. The ciphertext of message m that is sent to user U is $c = (r,s)$.

When U wants to recover the original message m from the ciphertext $c = (r,s)$, he computes

$$m = sr^{-x} \mod p.$$

The semantic security of ElGamal cryptosystem is equivalent to the Decisional Diffie-Hellman Assumption [14], which states that the Decisional Diffie-Hellman (DDH) problem is hard to solve.

Definition 2: (Decisional Diffie-Hellman problem.) We say that an algorithm is a (ε, T)-solver of the DDH problem if it runs in time at most T and distinguishes with probability at least $1/2 + \varepsilon$ between the two probability distributions $\mathcal{D}_{DH}$ and $\mathcal{D}_{rand}$, where $\mathcal{D}_{DH} = (g^a, g^b, g^{ab})$, for uniformly and independently chosen values $a, b \in \mathbb{Z}_q$, and $\mathcal{D}_{rand} = (g^a, g^b, g^c)$, for uniformly and independently chosen values $a, b, c \in \mathbb{Z}_q$.

B. Secret Sharing Techniques

In this section we explain some basics on secret sharing schemes. These schemes can also be used by a set of players to jointly generate shares of a random secret value $\alpha \in \mathbb{Z}_q$ such that the value $y = g^\alpha \mod p$ is public.

In a *secret sharing scheme*, a dealer (usually denoted by D) distributes shares of a secret value among a set of players $\mathcal{P} = \{P_1, \ldots, P_n\}$ in such a way that only authorized subsets of players (those in the monotone increasing access structure $\Gamma \subset 2^{\mathcal{P}}$) can recover the secret value from their shares, whereas non-authorized subsets do not obtain any information about the secret.

Secret sharing schemes were introduced independently by Shamir [29] and Blakley [3] in 1979. Shamir considered *threshold access structures* $\Gamma = \{A \subset \mathcal{P} : |A| \geq t+1\}$ defined by a threshold $t+1$, and proposed a secret sharing scheme for these structures, based on polynomial interpolation. Other works have proposed schemes realizing more general access structures, such as *vector space secret sharing schemes* [6]. An access structure Γ is realizable by such a scheme, in a finite field $\mathbb{Z}_q$ for some prime q, if there exists a positive integer r and an assignment of vectors $\psi : \mathcal{P} \cup \{D\} \longrightarrow (\mathbb{Z}_q)^r$ (one for each participant) such that $A \in \Gamma$ if and only if $\psi(D)$ is a linear combination of the vectors in $\{\psi(P_i)\}_{P_i \in A}$. Here D denotes a special entity (real or not), outside the set $\mathcal{P}$. If a real dealer D wants to share a secret $k \in \mathbb{Z}_q$ among the players in $\mathcal{P}$, he chooses a random vector $\mathbf{v} \in (\mathbb{Z}_q)^r$ such that $\mathbf{v} \cdot \psi(D) = k$. The share of each player $P_i \in \mathcal{P}$ is $k_i = \mathbf{v} \cdot \psi(P_i)$. If $A \in \Gamma$, there exist values $\{\lambda_i^A\}_{P_i \in A}$ such that $\psi(D) = \sum_{P_i \in A} \lambda_i^A \psi(P_i)$. Then, it is easy to see that $k = \sum_{P_i \in A} \lambda_i^A k_i \bmod q$. Using simple linear algebra, one can also see that subsets out of Γ obtain no information at all about the secret value k.

Simmons, Jackson and Martin [30] introduced *linear secret sharing schemes*, which can be seen as a generalization of vector space secret sharing schemes where each player can be assigned more than one vector (and therefore, the length of each share can be larger than the secret). They proved that any access structure can be realized by a linear secret sharing scheme; but the scheme is in general quite inefficient, in the sense that the rate between the length of the secret and the length of the shares is very low.

From now on in our work, we will consider any possible access structure Γ, so we will know that there exists a linear secret sharing scheme realizing this structure. For simplicity, we will suppose that this scheme is a vector space one defined by a function ψ. All the results are valid in the more general case where each player is associated with more than one vector, as well.

1) Joint Generation of a Random Shared Secret Value: If the secret value to be shared is a random number from the corresponding finite field, the generation and distribution of shares in a secret sharing scheme can be performed by the players themselves, without any external party (like the dealer).

Now we explain the protocol for the joint generation of a random secret value α, shared among players in $\mathcal{P}$ according to the access structure Γ. The protocol is secure against the action of a malicious adversary who can corrupt a subset of players in the adversary structure $\mathcal{A}$. An obvious required condition to ensure the privacy of the final secret is $\Gamma \cap \mathcal{A} = \emptyset$. We assume that Γ can be realized by a vector space secret sharing scheme defined by a mapping $\psi : \mathcal{P} \cup \{D\} \rightarrow (\mathbb{Z}_q)^r$. Some other parameters are public, like p, q, and two elements g, h such that $\mathbb{G} = \langle g \rangle$ has order q and $h \in \mathbb{G}$. The protocol, which is a generalization of the threshold protocol in [19], consists of the following steps:

1) Each player $P_i \in \mathcal{P}$ chooses at random a value $k_i \in \mathbb{Z}_q$, and distributes it among all players in $\mathcal{P}$, with the following verifiable secret sharing technique (which is a generalization of [26]):
   (a) P_i chooses two random vectors $\mathbf{v_i} = (v_i^{(1)}, \ldots, v_i^{(r)})$ and $\mathbf{w_i} = (w_i^{(1)}, \ldots, w_i^{(r)})$, in $(\mathbb{Z}_q)^r$, such that $\mathbf{v_i} \cdot \psi(D) = k_i$. Then P_i sends to each player P_j in $\mathcal{P}$ the pair $(k_{ij}, k'_{ij}) = (\mathbf{v_i} \cdot \psi(j), \mathbf{w_i} \cdot \psi(j))$. He also makes public the commitments $V_{i\ell} = g^{v_i^{(\ell)}} h^{w_i^{(\ell)}}$, for $1 \leq \ell \leq r$.
   (b) Each player $P_j \in \mathcal{P}$ verifies the correctness of his share k_{ij} by checking that
$$g^{k_{ij}} h^{k'_{ij}} = \prod_{\ell=1}^{r} (V_{i\ell})^{\psi(P_j)^{(\ell)}}, \qquad (1)$$
   where $\psi(P_j)^{(\ell)}$ is the ℓ-th component of the vector $\psi(P_j)$. If this check fails, P_j makes public a complaint against P_i.
   (c) For any complaint that P_i receives from some player P_j, he must broadcast a pair of values (k_{ij}, k'_{ij}) satisfying Eq. 1.
   (d) Player P_i is marked as *disqualified* if he receives complaint from a set of players out of $\mathcal{A}$, or if he answers some complaint with values which do not satisfy Eq. 1.
2) Each player defines the set of non-disqualified players $Qual$. Since $\mathcal{A}^c \subset \Gamma$, we have that $Qual \in \Gamma$. Furthermore, for all players $P_i \in Qual$ that pass this phase, there are valid shares k_{ij} corresponding to players P_j that form an authorized subset.
3) The generated random secret will be $\alpha = \sum_{P_i \in Qual} k_i$. Each player $P_j \in \mathcal{P}$ computes his share α_j of the secret α as $\alpha_j = \sum_{P_i \in Qual} k_{ij}$.
4) Now the players want to compute the value $y = g^{\alpha} = \prod_{P_i \in Qual} g^{k_i} \in \mathbb{Z}_p^*$. They use Feldman's verifiable secret sharing scheme (see [15] for the original threshold version):
   (a) Each player $P_i \in Qual$ broadcasts $A_{i\ell} = g^{v_i^{(\ell)}}$, for $1 \leq \ell \leq r$.
   (b) Each player $P_j \in Qual$ verifies if
$$g^{k_{ij}} = \prod_{\ell=1}^{r} (A_{i\ell})^{\psi(P_j)^{(\ell)}} \qquad (2)$$
   If this check fails, player P_j complains by broadcasting the pair (k_{ij}, k'_{ij}) that satisfies Eq. 1 but do not satisfy Eq. 2.
   (c) If player P_i receives some valid complaint at step 4(b), the other players $P_j \in Qual$ run the reconstruction phase of Pedersen's scheme to recover k_i and in particular $A_i = g^{k_i}$. Otherwise, the value g^{k_i} is computed from the commitments $A_{i\ell}$ and the public vector $\psi(D)$ as
$$g^{k_i} = g^{\mathbf{v_i} \cdot \psi(D)} = \prod_{\ell=1}^{r} g^{v_i^{(\ell)} \psi(D)^{(\ell)}} = \prod_{m=1}^{t} (A_{i\ell})^{\psi(D)^{(\ell)}}.$$
   (d) Finally, the value $y = g^{\alpha}$ can be computed as
$$y = g^{\sum_{P_i \in Qual} k_i} = \prod_{P_i \in Qual} g^{k_i}.$$

Something similar happens with the values $D_j = g^{\alpha_j} = g^{\sum_{P_i \in Qual} k_{ij}}$, which can be publicly computed from the values $g^{k_{ij}}$ obtained in steps 4(b) and 4(c). We denote the output of this protocol with the expression:
$$(\alpha_1, \ldots, \alpha_n) \quad \stackrel{(\mathcal{P},\Gamma,\mathcal{A})}{\longleftrightarrow} \quad (\alpha, g^{\alpha}, \{D_j\}_{1 \leq j \leq n})$$

where α_i is the share of the secret α held by player P_i, and the values g^α and $\{D_j\}_{1\le j\le n}$ are publicly known.

It is not difficult to see that this protocol is *simulatable*, in the following sense: given a corruptible subset $B \in \mathcal{A}$ and a value $Y \in \mathbb{G}$, it is possible to simulate, in polynomial time, values which are indistinguishable from the ones that an adversary corrupting players in B would obtain from an execution of the protocol which outputs $g^\alpha = Y$. We denote all this simulated information by $Sim(B,Y)$. This fact ensures that the protocol is secure, because an adversary does not obtain any information from the execution of the protocol that he could not produce by himself.

The proof of this fact is omitted here by lack of originality, since it is basically a rewriting of the proof of simulatability of the protocol in [19], where threshold secret sharing is replaced with linear secret sharing. Note that this change does not add any new subtle issue to the proof in [19], which had been proposed to correct some flaws in previous well-known protocols for distributed key generation [18], [26].

IV. The Distributed Key Distribution Scheme of Daza, Herranz, Padró and Sáez

In this section we analyze the scheme proposed in [11]. The goal of this paper is to prove the security of this scheme, because [11] does not include any security analysis. After reviewing and modifying the protocols of the scheme, we first prove that the employed proof of knowledge is secure. Then, in Section IV-C, we will prove that the whole distributed key distribution scheme is secure.

In the original proposal of Naor et al. [24], servers must send the partial information to users throughout private channels. In practice, this can be done for example by using a suitable encryption scheme. Then, the user must check the correctness of the received values and combine some correct values to obtain the final key; this step involves some modular exponentiations which can be costly to the user, if he does not have enough computational resources.

The goal of the scheme by Daza et al. [11] is to reduce the computational effort required to the users. The idea is the following: since the information received by users must be encrypted anyway, an encryption scheme such as ElGamal cryptosystem can be used. In general, any multiplicatively *homomorphic* encryption scheme $E_{PK}(m)$ satisfying $E_{PK}(m_1 \cdot m_2) = E_{PK}(m_1) \cdot E_{PK}(m_2)$ could be used. The partial encryptions computed by the servers can be combined by the servers themselves (with one extra round of communication) in order to produce an encryption of the final conference key. This value is sent to the user, who is the only one who can decrypt it and obtain the key. In this way, the number of operations that users must perform is significantly reduced, at the cost of increasing the computational and communication effort required to servers. Note that this may make sense in real scenarios where users have low computational resources, and servers are paid in some way for doing this work.

A. The Scheme

Let $\mathcal{U} = \{U_1,\dots,U_m\}$ and $\mathcal{S} = \{S_1,\dots,S_n\}$ be the sets of users and servers. Let $\Gamma, \mathcal{A} \subset 2^{\mathcal{S}}$ be the access and adversary structures, satisfying $\mathcal{A}\cap\Gamma = \emptyset$ and $\mathcal{A}^c \subset \Gamma$. Again, we assume that the access structure Γ can be realized by a vector space secret sharing scheme. That is, there exist a positive integer r and a function $\psi : \mathcal{S}\cup\{D\} \longrightarrow (\mathbb{Z}_q)^r$ such that $A \in \Gamma$ if and only if $\psi(D) \in \langle\psi(S_i)\rangle_{S_i\in A}$.

We denote by $\Omega = \Omega(\Gamma,\mathcal{A}) = \{R \subset \mathcal{P} \mid R - B \in \Gamma, \text{ for all } B \in \mathcal{A}\} \subset \Gamma$ the monotone increasing family of *robust subsets*. Since Γ and $\mathcal{A}$ satisfy $\mathcal{A}^c \subset \Gamma$, we know that $\mathcal{P} \in \Omega$ and so the family Ω is not empty. In the scheme, a user who wants to know a conference key must contact with the servers of some robust subset $R \in \Omega$.

Let H be a hash function (collision and pre-image resistant) that inputs a conference $C \subset \mathcal{U}$ and outputs an element in $\mathbb{G} = \langle g\rangle \subset \mathbb{Z}_p^*$. In the security proof of the scheme, this hash function is assumed to behave as a random oracle (following a usual methodology introduced in [2]).

Each user $U \in \mathcal{U}$ has an ElGamal secret key $x \in \mathbb{Z}_q^*$ and the matching public key is $y = g^x$. The parameters $(p, q, \mathbb{G} = \langle g\rangle)$ are the same for all the servers and users in the system.

Initialization Phase

In order to compute their secret shares, servers in $\mathcal{S}$ jointly execute the protocol

$$(\alpha_1,\dots,\alpha_n) \overset{(\mathcal{S},\Gamma,\mathcal{A})}{\longleftrightarrow} \left(\alpha, g^\alpha, \{D_j\}_{1\le j\le n}\right)$$

where $\alpha, \alpha_i \in \mathbb{Z}_q$ are random, as explained in Section III-B.1.

Note that in the proposal published in [11], a different (and simpler) protocol for the joint generation of a random secret is used. However, when analyzing security, one realizes that the simpler protocol is not simulatable and therefore the security of the distributed key distribution scheme, as it is in [11], cannot be proved. However, the protocol explained in Section III-B.1 is simulatable, and the whole scheme can be proved secure, as we will see in Section IV-C.

Key Request and Computational Phase

A user U in a conference $C \subset \mathcal{U}$ asks for the conference key κ_C to a robust subset of servers $R \in \Omega$.

1) Each server $S_i \in R$ applies the hash function H to the conference C, obtaining $h_C = H(C) \in \mathbb{G}$. The conference key will be $\kappa_C = h_C^\alpha$.
2) Then server $S_i \in R$ encrypts the value $h_C^{\alpha_i}$ using the ElGamal public key y of user U. That is:
   - Server S_i chooses a random element $\beta_i \in \mathbb{Z}_q^*$.
   - He computes $r_i = g^{\beta_i}$ and $s_i = h_C^{\alpha_i} y^{\beta_i}$.
   - Server S_i broadcasts the ciphertext $c_i = (r_i, s_i)$.
3) Server S_i computes a non-interactive and zero-knowledge proof of the knowledge of values α_i and β_i which make consistent the ciphertext c_i:

$$Proof_i = PK\{ (\alpha_i,\beta_i) : D_i = g^{\alpha_i} \wedge r_i = g^{\beta_i} \wedge s_i = h_C^{\alpha_i} y^{\beta_i} \}.$$

   More details about the design and security of this proof of knowledge, which makes use of a hash function $\hat{H} : \{0,1\}^* \to \mathbb{Z}_q$, are given in Section IV-B.
4) Each server $S_i \in R$ verifies the proofs published by the rest of servers, until he obtains correct partial ciphertexts from an authorized subset $A \in \Gamma$. Notice that such a subset in Γ always exists, because $R \in \Omega$, and therefore $R - B \in \Gamma$ for any possible subset B of incorrect partial ciphertexts computed by corrupted servers.
   Since $A \in \Gamma$, we know that there exist values $\{\lambda_j^A\}_{S_j\in A}$ such that $\alpha = \sum_{S_j\in A} \lambda_j^A \alpha_j$.

5) Using homomorphic properties of ElGamal encryption, server S_i can compute an encryption (r,s) of the conference key $\kappa_C = h_C^\alpha$ from the values c_j broadcast by servers $S_j \in A$, as follows:

$$r = \prod_{S_j \in A} r_j^{\lambda_j^A} = g^{\sum_{S_j \in A} \lambda_j^A \beta_j}$$

$$s = \prod_{S_j \in A} s_j^{\lambda_j^A} = h_C^{\sum_{S_j \in A} \lambda_j^A \alpha_j} \; y^{\sum_{S_j \in A} \lambda_j^A \beta_j} = h_C^\alpha \, y^{\sum_{S_j \in A} \lambda_j^A \beta_j}$$

Since the elements $\{\beta_j\}_{S_j \in A}$ are random, the element $\sum_{S_j \in A} \lambda_j^A \beta_j$ is also random, and so (r,s) is a valid ElGamal encryption of the message h_C^α. We also note that the resulting ciphertext (r,s) does not depend on the considered authorized subset $A \in \Gamma$.

Key Delivery Phase

Each server in R sends to user U the ciphertext $c = (r,s)$ that he has computed. User U selects a value that has been sent by all the servers of some subset out of $\mathcal{A}$ (this means that there exists at least one honest server in this subset, and so the corresponding ciphertext must be the correct one). The user decrypts it and obtains the conference key $\kappa_C = h_C^\alpha$. Note that, as desired, the obtained key does not depend on the set of servers who have participated in the execution of the protocol.

B. *The Proof of Knowledge*

In the protocol described above, each server S_i must compute the proof

$$Proof_i = PK\{(\alpha_i,\beta_i): D_i = g^{\alpha_i} \wedge r_i = g^{\beta_i} \wedge s_i = h_C^{\alpha_i} \, y^{\beta_i}\}.$$

This proof can be computed using a standard strategy [8], [16], [28] that transforms an interactive proof of knowledge into a non-interactive one, by using a hash function $\hat{H}$ which is modelled, in the security analysis, as a random oracle.

For some known values $A, B, C, g_1, g_2, g_3 \in \mathbb{G}$, the proof

$$PK\{(\alpha,\beta): \; A = g_1^\alpha \; \wedge \; B = g_1^\beta \; \wedge \; C = g_2^\alpha g_3^\beta\}$$

can be computed as follows: let $\hat{H} : \{0,1\}^* \to \mathbb{Z}_q$ be a hash function. The prover does the following:

1) Choose uniformly and at random $u, v \in \mathbb{Z}_q^*$.
2) Compute $T_1 = g_1^u$, $T_2 = g_1^v$ and $T_3 = g_2^u g_3^v$
3) Compute $h = \hat{H}(A,B,C,g_1,g_2,g_3,T_1,T_2,T_3)$
4) Compute $w_1 = u - \alpha h \mod q$ and $w_2 = v - \beta h \mod q$.
5) The proof of knowledge is the tuple (h, w_1, w_2).

The verifier of the proof must check if

$$h = \hat{H}\left(A,B,C,g_1,g_2,g_3, \; g_1^{w_1}A^h, \; g_1^{w_2}B^h, \; g_2^{w_1}g_3^{w_2}C^h\right).$$

If so, then he accepts the proof; otherwise he rejects the proof.

A zero-knowledge proof of knowledge must satisfy three properties: *correctness*, which means that the verifier always accepts the proof if the prover knows the secret values; *soundness*, which means that, if a prover does not know the secrets, then the probability that he computes a proof that makes the verifier accept is negligible; and finally, *zero-knowledge* means that the only information that the verifier obtains from the execution of the protocol is if the prover knows the stated secret values or not.

Correctness of the protocol explained above is trivially fulfilled. We next prove that the protocol also achieves the other two properties, in the random oracle model [2], where a hash function is seen as a totally random function.

Soundness

We prove that an algorithm $\mathcal{B}$ which has probability ε to compute a valid proof knows the values α and β with probability $\varepsilon^2/5$.

In effect, the randomness of the algorithm $\mathcal{B}$ consists of the randomness of the values output by the (random) hash function $\hat{H}$ and the own randomness of $\mathcal{B}$. If a execution of $\mathcal{B}$ produces a tuple (h, w_1, w_2), then we denote as X the randomness employed by $\mathcal{B}$ just before asking to the random oracle the value $\hat{H}\left(A,B,C,g_1,g_2,g_3, \; g_1^{w_1}A^h, \; g_1^{w_2}B^h, \; g_2^{w_1}g_3^{w_2}C^h\right)$. We denote as Y the rest of randomness employed by $\mathcal{B}$ (including the value output by the random oracle in the above-mentioned query).

We denote as $W \subset X \times Y$ the set of random choices (or executions of $\mathcal{B}$) that lead to a valid proof of knowledge. By hypothesis, we have that $\Pr[W] = \varepsilon$. We apply to this situation the well-known splitting lemma (see for example [27]), also known as heavy-row lemma.

Then there exists $V \subset W$ such that, for all $(x,y) \in V$, we have

$$\Pr_{y'}[(x,y') \in W] \geq \frac{\varepsilon}{2} \quad \text{and} \quad \Pr[V \mid W] \geq \frac{1}{2}.$$

When we execute $\mathcal{B}$ with randomness (x,y), we will obtain $(x,y) \in V$ with probability at least $\varepsilon/2$. In this case, we execute $\mathcal{B}$ again with fixed x and random y'. With probability at least $\varepsilon/2$, the new execution (x,y') belongs to W; that is, it also leads to a valid proof of knowledge.

Since $\hat{H}$ and $\hat{H}'$ are random functions, the probability that $h = \hat{H}(A,B,C, \; g_1,g_2,g_3,g_1^u,g_1^v,g_2^u g_3^v) = \hat{H}'(A,B,C,g_1,g_2,g_3,g_1^u,g_1^v,g_2^u g_3^v) = h'$ is $1/q$. Since $q > 5$, we have in particular that the probability that $h \neq h'$ is $1 - 1/q > 4/5$.

Summing up, $\mathcal{B}$ would obtain with probability at least $\varepsilon/2 \cdot \varepsilon/2 \cdot (q - 1/q) > \varepsilon^2/5$ two valid proofs of knowledge (h, w_1, w_2) corresponding to the randomness (x,y) and (h', w_1', w_2') corresponding to the randomness (x, y'), such that $h \neq h'$.

Since the randomness x is equal until the crucial query to the random oracle $\hat{H}$, we get:

- $g_1^{w_1}A^h = g_1^{w_1'}A^{h'}$, and
- $g_1^{w_2}B^h = g_1^{w_2'}B^{h'}$.

The first statement implies that $w_1 + \alpha h = w_1' + \alpha h' \mod q$, whereas the second statement implies $w_2 + \beta h = w_2' + \beta h' \mod q$. Therefore, the algorithm $\mathcal{B}$ can compute

$$\alpha = \frac{w_1 - w_1'}{h' - h} \mod q \quad \text{and} \quad \beta = \frac{w_2 - w_2'}{h' - h} \mod q.$$

Zero knowledge

We show that a verifier does not obtain any new information from the execution of the protocol, other than the proof itself. In order to prove this, we show that the verifier himself could have computed a valid proof of knowledge which is indistinguishable from a real one computed by a honest prover, in the random oracle model.

In effect, the verifier can choose at random $w_1, w_2, h \in \mathbb{Z}_q$. Then, he can impose that

$$\hat{H}\left(A,B,C,g_1,g_2,g_3, \; g_1^{w_1}A^h, \; g_1^{w_2}B^h, \; g_2^{w_1}g_3^{w_2}C^h\right) = h.$$

Since the hash function $\hat{H}$ is assumed to behave as a totally random function (the proof is valid in the random oracle model), then this step is consistent and produces a valid proof of knowledge, which is indistinguishable from a real one computed by a honest prover who knows α and β. This is true if the value imposed to the hash function $\hat{H}$ in the simulation does not produce a collision with previously assigned values.

C. Security Analysis

In this section we prove that the scheme of Daza et al., explained in Section IV-A, is secure with respect to the formal model explained in Section II-A.2, as long as the Decisional Diffie-Hellman problem (see Definition 2) is hard to solve. As it happens with the scheme of Naor et al. [24], the proof is valid in the random oracle model for the hash functions H and $\hat{H}$. In Section V, we will explain how it is possible to erase this assumption for the hash function H (this modification is valid for both schemes). This will be at the cost of a loss in efficiency.

Theorem 1: Assume that the distributed key distribution scheme of Daza et al. can be $(\mathcal{U}, \mathcal{S}, \Gamma, \mathcal{A}, T, \varepsilon, Q_{conf})$-broken with Q_H queries to the random oracle H and $Q_{\hat{H}}$ queries to the random oracle $\hat{H}$, satisfying $Q_{conf} < q/(2n)$ and $Q_{\hat{H}} < q^2/2$.

Then the Decisional Diffie-Hellman problem can be (ε', T')-solved, where $\varepsilon' \geq \varepsilon/4$ and $T' \leq T + T_f Q_H + (7+n^2) T_f Q_{conf}$, where T_f is the time to perform a modular exponentiation.

Proof: Let $\mathcal{D} = (g^a, g^b, g^c)$ be an instance of the Decisional Diffie-Hellman problem; the goal is to decide if $\mathcal{D} = \mathcal{D}_{DH}$ or $\mathcal{D} = \mathcal{D}_{rand}$. To do it, we will run the hypothetic adversary $\mathcal{F}$ against the scheme of Daza et al.. We initialize the scheme with the same public parameters that appear in the given instance of the DDH problem: the primes p and q, and the element g. We send to $\mathcal{F}$ the information about the sets and the structures: $\mathcal{S}$, Γ, $\mathcal{A}$, $\mathcal{U}$ and $\mathcal{C}$.

Since we are assuming that the hash function $H : \{0,1\}^* \rightarrow \mathbb{G}$ behaves as a random function, we maintain a table TAB_H as follows: when $\mathcal{F}$ asks for the value $H(C)$ for some conference $C \subset \mathcal{U}$, we look for the input C in the table TAB_H. If it is already in the table, we return to $\mathcal{F}$ the corresponding output. If not, we choose at random $\rho_C \in \mathbb{Z}_q$ and define $h_C = H(C) = (g^a)^{\rho_C}$. We return this value to $\mathcal{F}$ and store the new relation in TAB_H. The condition $h_{C_1} \neq h_{C_2}$ must be satisfied for any two different conferences $C_1 \neq C_2$.

In an analogous way, we manage a table $TAB_{\hat{H}}$ for the values of the random hash function $\hat{H}$ that is used in the proofs of knowledge.

The adversary $\mathcal{F}$ plays game $\mathcal{G}$ against the scheme, and we must simulate all the information that $\mathcal{F}$ obtains in the different steps of the game. Note that the simulation that we propose below will be consistent only in the case where $\mathcal{D} = \mathcal{D}_{DH}$.

1) $\mathcal{F}$ chooses a subset $B \in \mathcal{A}$ of corrupted servers. Without loss of generality (adding if necessary some players to the original subset B), we can assume that B is a maximal non-authorized subset; that is, for any server $S_i \notin B$, it is fulfilled that $B \cup \{S_i\} \in \Gamma$.
2) The initialization phase of the scheme must be executed, which involves only the servers. The adversary obtains the public information and the private information of the corrupted servers. We proceed as follows: we compute $Sim(B, g^b)$, which is all the information that the adversary would obtain in an execution of the joint protocol for the generation of a shared secret value α such that $g^\alpha = g^b$ (described in Section III-B.1). This information $Sim(B, g^b)$ consists (basically) of the values $\{\alpha'_j\}_{S_j \in B}$ and $\{g^{\alpha'_i}\}_{S_i \in \mathcal{P}}$.
3) In step 3 of the game, $\mathcal{F}$ chooses different conferences $C' \subset \mathcal{U}$. For each user $U \in C'$, with ElGamal public key y, the adversary $\mathcal{F}$ must receive all the information that U and servers in B would see in an execution of the protocol where U obtains an encryption of the conference key $\kappa_{C'} = h_{C'}^\alpha = g^{a\rho_{C'} b}$. We allow the adversary $\mathcal{F}$ to know the secret key of user U and thus to obtain the final conference key $\kappa_{C'}$.
We are implicitly assuming that $g^c = g^{ab}$. So first we compute an ElGamal encryption (under U's public key, y) of the assumed final key $\kappa_{C'} = g^{c\rho_{C'}}$: we choose at random $\beta \in \mathbb{Z}_q^*$ and compute
$$r = g^\beta \text{ and } s = \left(g^c\right)^{\rho_{C'}} y^\beta \ .$$
Then we compute the partial ElGamal encryptions, starting with servers $S_j \in B$; for them, we know the shares α'_j, so we choose at random $\beta_j \in \mathbb{Z}_q^*$ and compute
$$r_j = g^{\beta_j} \text{ and } s_j = \left(g^{a\rho_{C'}}\right)^{\alpha'_j} y^{\beta_j} \ .$$
We can easily compute $Proof_j$ for these servers S_j, as well, since we know the values α'_j and β_j.
Finally, for servers $S_i \notin B$, we know that $B \cup \{S_i\} \in \Gamma$. This implies the existence of values λ_0 and $\{\lambda_j\}_{S_j \in B}$ such that $\psi(S_i) = \lambda_0 \psi(D) + \sum_{S_j \in B} \lambda_j \psi(S_j)$. Then we can compute the values
$$r_i = r^{\lambda_0} \prod_{S_j \in B} r_j^{\lambda_j} \quad \text{and} \quad s_i = s^{\lambda_0} \prod_{S_j \in B} s_j^{\lambda_j} \ ,$$
which form a valid ElGamal encryption of the value $h_{C'}^{\alpha'_i} = (g^{a\rho_{C'}})^{\alpha'_i}$ under public key y.
To simulate $Proof_i$ for servers $S_i \notin B$, we proceed as we described in Section IV-B. We choose at random $w_1, w_2, h \in \mathbb{Z}_q$ and define
$$\hat{H}\left(D_i, r_i, s_i, g, h_{C'}, y,\ g^{w_1} D_i^h,\ g^{w_2} r_i^h,\ h_{C'}^{w_1} y^{w_2} s_i^h\right) \ = \ h.$$
This relation is stored in the table $TAB_{\hat{H}}$. Note that there cannot be collisions among different simulations of proofs of knowledge, because there is only one simulation for the values $D_i, h_{C'}, y$ corresponding to server S_i, conference C' and user U with public key y. However, there can be other kind of collisions, if some input of $\hat{H}$ obtained in a simulation has been already asked to the random oracle by $\mathcal{F}$. If we detect such a collision, we abort the game and output a random answer to the given instance of the DDH problem. Since we simulate at most nQ_{conf} proofs of knowledge and $\mathcal{F}$ is allowed to make at most $Q_{\hat{H}}$ queries to the random oracle $\hat{H}$, the probability μ that such a collision occurs is
$$\mu \leq \frac{nQ_{conf}Q_{\hat{H}}}{q^3}.$$
Since $Q_{conf} < q/(2n)$ and $Q_{\hat{H}} < q^2/2$, the probability that no collision occurs is $1 - \mu > 1/2$.
4) The adversary $\mathcal{F}$ chooses a conference C different from the previous conferences C'. We can simulate the information that $\mathcal{F}$ would obtain from the execution of the protocol

for C, exactly in the same way as in the previous step. The difference is that now we do not allow $\mathcal{F}$ to obtain the ElGamal decryption, which is, supposedly, the final conference key κ_C.

Note that this step 4) can eventually occur in parallel to previous step 3); the simulation can be done exactly in the same way. As well, new key conference queries, as those in step 3), could be made by the adversary now, for conferences C' satisfying $C' \neq C$. They are answered exactly as explained in step 3).

5) In step 5, we send to $\mathcal{F}$ the value $(g^c)^{\rho_C}$. Note that, if $g^c = g^{ab}$, this value is exactly κ_C.
6) The adversary $\mathcal{F}$ outputs a bit b'.

We answer to the given instance $\mathcal{D}$ of the DDH problem as follows: if $\mathcal{F}$ outputs 1, we state that $\mathcal{D}$ follows the Diffie-Hellman distribution, or in other words that $c = ab \mod q$. If $\mathcal{F}$ outputs 0, we state that $c \neq ab \mod q$.

Let us compute the probability ε' that we solve the DDH problem correctly. If some collision happens in the management of the table $TAB_{\hat{H}}$, then we succeed with probability $1/2$. If no collisions occur, then we can distinguish two cases. On the one hand, if $\mathcal{D} = \mathcal{D}_{DH}$, which happens with probability $1/2$ since $\mathcal{D}$ is taken at random between $\mathcal{D}_{DH}$ and $\mathcal{D}_{rand}$, all the values that $\mathcal{F}$ receives during the game are consistent, and in step 6 he receives the correct value of κ_C (that is, $b^* = 1$). In this case, by hypothesis, $\mathcal{F}$ will output the correct answer $b' = b^* = 1$ with probability at least $\varepsilon + 1/2$. And with this probability we will answer correctly that $\mathcal{D}$ follows the Diffie-Hellman distribution. On the other hand, if $\mathcal{D} = \mathcal{D}_{rand}$ (which also happens with probability $1/2$), all the information that $\mathcal{F}$ receives during the game is completely independent of the bit b^* which defines the game, so $\mathcal{F}$ will output the correct $b' = 0$ with probability $1/2$.

Summing up, considering all the possible cases, we obtain that the probability of correctly solving the instance of the DDH problem is at least

$$\mu \cdot \frac{1}{2} + (1-\mu) \cdot \left(\frac{1}{2} \cdot \frac{2\varepsilon + 1}{2} + \frac{1}{2} \cdot \frac{1}{2} \right) = \frac{1}{2} + \frac{\varepsilon(1-\mu)}{2} > \frac{1}{2} + \frac{\varepsilon}{4} .$$

Therefore, we get $\varepsilon' \geq \varepsilon/4$. With respect to the execution time T' of our algorithm for solving the DDH problem, if we consider as significant only the time T_f to perform modular exponentiations, then T' is the same as the execution time T of $\mathcal{F}$, plus the time of answering the Q_H queries to the hash (random) oracle H, plus the time of simulating the information that $\mathcal{F}$ obtains in game $\mathcal{G}$ (ElGamal partial encryptions, proofs of knowledge, etc.). Summing up, we have $T' \leq T + T_f Q_H + (7 + n^2) T_f Q_{conf}$.

■

V. A Modification to Avoid the Random Oracle Model

The security of the distributed key distribution schemes in [11], [24] can be proved in the random oracle model. This is a usual strategy in cryptography, although the assumption of the total randomness of a hash function is unrealistic. Even if a proof in the random oracle model is always better than nothing, efforts are constantly made by cryptographers to provide protocols whose security does not rely on this assumption.

In their work [24], Naor et al. briefly mention a possible modification in the way in which the values h_C are computed, in order to avoid the use of hash function H and therefore the assumption of the random oracle model for this function in the security proof. We detail here this modification, which is valid also for the scheme of Daza et al. that we study in this paper. However, note that the random oracle model will be still necessary for the function $\hat{H}$ if we consider the zero-knowledge proofs of knowledge to achieve robustness against malicious adversaries (in both schemes).

The idea is that servers will jointly compute a value $h_C \in \mathbb{G} = \langle g \rangle$ for each conference C and store this relation in a table. In some sense, the management of this table by the servers plays the role of a random oracle model. Since the values h_C can be random, servers can compute them off-line, in the initialization phase. Later, when a new conference key is requested, they publicly assign to this conference C one of the pre-computed values h_C, and store the relation in the table. When all the users in the conference C have obtained the key κ_C, they erase this relation from the table.

Note that this solution may be suitable only for scenarios where the expected number of conferences is not too big, since the size of the table that must be managed by the servers is linear on the number of expected conferences.

There are different possibilities for this joint generation of the value h_C. A first (naive) solution consists in each server $S_i \in \mathcal{S}$ generating at random a value $\mu_{C,i} \in \mathbb{Z}_q$ and then broadcasting the value $h_{C,i} = g^{\mu_{C,i}}$. The final value to be stored in the table is

$$h_C = \prod_{S_i \in \mathcal{S}} h_{C,i} = g^{\sum_{S_i \in \mathcal{S}} \mu_{C,i}} .$$

However, this solution requires simultaneous broadcast channels [21] for all the servers in $\mathcal{S}$, to ensure that all the values $h_{C,i}$ are broadcast at the same time, with independence. If not, a server S_j corrupted by the adversary could wait and choose its value $h_{C,j}$ in order to result in a desired h_C. Later, he can do the same for a different conference C' and produce, for example, $h_{C'} = h_C^2$. In this case, if the adversary could have access to the key $\kappa_C = h_C^\alpha$, then he would be able to compute the conference key $\kappa_{C'} = h_{C'}^\alpha = \kappa_C^2$, breaking thus the security of the scheme.

Such an assumption about simultaneous broadcast channels may be very strong in some scenarios. For this reason, an alternative (and less efficient) solution is that servers run, for each value h_C to be computed, the protocol described in Section III-B.1. With this last modification, the security of the two distributed key distribution schemes [11], [24] can be proved in a similar way. Since the protocol in Section III-B.1 is simulatable, the information that the adversary obtains in the generation of the values h_C can be perfectly simulated in the security proof. The relation between the times T' and T, in Theorem 1, will now include the time of simulating Q_{conf} times the execution of the aforementioned protocol. Note that this use of verifiable secret sharing techniques (i.e. protocol in Section III-B.1) to achieve “mutually independent channels” is not new, see the original paper [9] on verifiable secret sharing.

VI. Conclusions

In this work we have analyzed the security of an existing proposal of distributed key distribution scheme [11], which had not been studied at all. This proposal results specially suitable in real scenarios where users have low computational resources, and servers are even paid in some way for doing most of the work. We have considered a formal security model for this kind of schemes. We have proved that this specific scheme achieves the desired level of security, in the random oracle model, as long as the Decisional Diffie-Hellman problem is hard to solve. To do so, we have first replaced the protocol for the joint generation of a random value in [11] with a more secure one, and we have also analyzed the security of a zero-knowledge protocol which provides robustness to the scheme.

We also discuss a modification of the scheme in [11], which is valid also for the scheme in [24], that replaces the use of a hash function with a table that must be managed by the servers. On the one hand, this results in less efficient distributed key distribution schemes, maybe only suitable for scenarios where the expected number of conferences is small; but on the other hand, this modification allows us to avoid the strong assumption that this hash function behaves as a random oracle.

Acknowledgment

The authors would like to thank the anonymous referees of IEEE Transactions on Computers for their helpful comments about this work. The work of the two first authors was partially supported by Spanish *Ministry of Education and Science*, under projects TSI2007-65406-C03-02 ("eAEGIS") and CONSOLIDER CSD2007-00004 ("ARES"). The work of the first author was also supported by the Government of Catalonia under grant 2005-SGR-00446 and by the UNESCO *Chair in Data Privacy*. The work of the second author was also supported by the Government of Catalonia under grant 2005-SGR-00093. Finally, the work of the last author was partially supported by Spanish *Ministry of Education and Science*, under project TSI2006-02731.

References

[1] M. Bellare, D. Pointcheval and P. Rogaway, Authenticated key exchange secure against dictionary attacks, *Proceedings of Eurocrypt'00*, Lecture Notes in Computer Science **1807**, Springer-Verlag (2000): pp. 139–155.
[2] M. Bellare and P. Rogaway, Random oracles are practical: a paradigm for designing efficient protocols, *Proceedings of the 1st ACM Conference on Computer and Communications Security* (1993): pp. 62–73.
[3] G.R. Blakley, Safeguarding cryptographic keys, *Proceedings of the National Computer Conference, American Federation of Information*, Processing Societies Proceedings, **48** (1979): pp. 313–317.
[4] C. Blundo, P. D'Arco, V. Daza and C. Padró, Bounds and constructions for unconditionally secure distributed key distribution schemes for general access structures, *Theoretical Computer Science*, **320** (2004): pp. 269–291.
[5] E. Bresson and D. Catalano, Constant round authenticated group key agreement via distributed computation, *Proceedings of PKC'04*, Lecture Notes in Computer Science **2947**, Springer-Verlag (2004): pp. 115–129.
[6] E.F. Brickell, Some ideal secret sharing schemes, *Journal of Combinatorial Mathematics and Combinatorial Computing*, **9** (1989): pp. 105–113.
[7] M. Burmester, Y.G. Desmedt, A secure and efficient conference key distribution system, *Proceedings of Eurocrypt'94*, Lecture Notes in Computer Science **950**, Springer-Verlag (1994): pp. 275–286.
[8] J. Camenisch, Group signature schemes and payment systems based on the discrete logarithm problem, PhD thesis, ETH Zurich. *Diss. ETH No. 12520* (1998).
[9] B. Chor, S. Goldwasser, S. Micali and B. Awerbuch, Verifiable secret sharing and achieving simultaneity in the presence of faults, *Proceedings of FOCS'85*, IEEE Computer Society Press (1985): pp. 383–395.
[10] P. D'Arco and D.R. Stinson, On unconditionally secure robust distributed key distribution centers, *Proceedings of Asiacrypt'02*, Lecture Notes in Computer Science **2501**, Springer-Verlag (2002): pp. 346–363.
[11] V. Daza, J. Herranz, C. Padró and G. Sáez, A distributed and computationally secure key distribution scheme, *Proceedings of ISC'02*, Lecture Notes in Computer Science **2433**, Springer-Verlag (2002): pp. 342–356.
[12] V. Daza, J. Herranz and G. Sáez, Constructing general dynamic group key distribution schemes with decentralized user join, *Proceedings of ACISP'03*, Lecture Notes in Computer Science **2727**, Springer-Verlag (2003): pp. 464–475.
[13] V. Daza, J. Herranz and G. Sáez, Protocols useful on the Internet from distributed signature schemes, *International Journal of Information Security*, Vol. **3** (2), Springer Verlag (2004): pp. 61–69.
[14] W. Diffie and M.E. Hellman, New directions in cryptography, *IEEE Transactions on Information Theory*, Vol. **22 (6)** (1976): pp. 644–654.
[15] P. Feldman, A practical scheme for non-interactive verifiable secret sharing, *Proceedings of FOCS'87*, IEEE Computer Society Press (1987): pp. 427-437.
[16] A. Fiat and A. Shamir, How to prove yourself: practical solutions of identification and signature problems, *Proceedings of Crypto'86*, Lecture Notes in Computer Science **263**, Springer-Verlag (1986): pp. 186–194.
[17] T. ElGamal, A public key cryptosystem and a signature scheme based on discrete logarithms, *IEEE Transactions on Information Theory* **31** (1985): pp. 469–472.
[18] R. Gennaro, S. Jarecki, H. Krawczyk and T. Rabin, Robust threshold DSS signatures, *Proceedings of Eurocrypt'96*, Lecture Notes in Computer Science **1070**, Springer-Verlag (1996): pp. 354–371.
[19] R. Gennaro, S. Jarecki, H. Krawczyk and T. Rabin, Secure distributed key generation for discrete-log based cryptosystems, *Proceedings of Eurocrypt'99*, Lecture Notes in Computer Science **1592**, Springer-Verlag (1999): pp. 295–310.
[20] S. Goldwasser and S. Micali, Probabilistic encryption, *Journal of Computer and System Sciences*, **28** (1984): pp. 270–299.
[21] A. Hevia and D. Micciancio, Simultaneous broadcast revisited, *Proceedings of PODC'05*, ACM Press (2005): pp. 324–333.
[22] J. Katz and M. Yung, Scalable protocols for authenticated group key exchange, *Proceedings of Crypto'03*, Lecture Notes in Computer Science **2729**, Springer-Verlag (2003): pp. 110–125.
[23] H. Kurnio, R. Safavi-Naini and H. Wang, A group key distribution scheme with decentralised user join, *Proceedings of SCN'02*, Lecture Notes in Computer Science **2576**, Springer-Verlag (2002): pp. 146–163.
[24] M. Naor, B. Pinkas and O. Reingold, Distributed pseudo-random functions and KDCs, *Proceedings of Eurocrypt'99*, Lecture Notes in Computer Science **1592**, Springer-Verlag (1999): pp. 327–346.
[25] R.M. Needham and M.D. Schroeder, Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, **21**(1978): pp. 993–999.
[26] T.P. Pedersen, Non-interactive and information-theoretic secure verifiable secret sharing, *Proceedings of Crypto'91*, Lecture Notes in Computer Science **576**, Springer-Verlag (1991): pp. 129–140.
[27] D. Pointcheval and J. Stern, Security arguments for digital signatures and blind signatures, *Journal of Cryptology*, **13 (3)**, Springer-Verlag (2000): pp. 361–396.
[28] C.P. Schnorr, Efficient signature generation by smart cards, *Journal of Cryptology*, **4**, Springer-Verlag (1991): pp. 161–174.
[29] A. Shamir, How to share a secret, *Communications of the ACM*, **22** (1979): pp. 612–613.
[30] G.J. Simmons, W. Jackson and K. Martin, The geometry of secret sharing schemes, *Bulletin of the ICA* **1** (1991): pp. 71–88.

Vanesa Daza graduated in Mathematics in University of Barcelona (UB, Barcelona, Spain) in 1999 and received her Ph.D. in Mathematics in Technical University of Catalonia (UPC, Barcelona, Spain) in 2004. Afterwards, she joined a crypto-based security company as a Senior Researcher. Currently she holds a lecturer position in the CRISES research group in the Rovira i Virgili University (URV, Tarragona, Spain). Her research interests are mainly related with distributed cryptography, including secret sharing and multiparty computation.

Javier Herranz graduated in Mathematics in 2000 and obtained his Ph.D. in Applied Mathematics in 2005, in the Technical University of Catalonia (UPC, Barcelona, Spain). After that, he spent 9 months in the École Polytechnique (France) and 9 months in the Centrum voor Wiskunde en Informatica (CWI, The Netherlands), as a post-doctoral researcher, granted with an ERCIM fellowship (from May 2005 to October 2006). Currently, he is a post-doctoral researcher in IIIA-CSIC (Bellaterra, Spain). His research interests are mostly related to cryptography, especially to digital signatures and distributed cryptographic protocols.

Germán Sáez is associate professor in the Dept. of Applied Mathematics IV at the Technical University of Catalonia (UPC, Barcelona, Spain). He graduated in Mathematics (UB, Barcelona, Spain) in 1987 and received his Ph.D. in Mathematics (UPC) in 1998. He has been the general co-chair of the conference Eurocrypt'08. He belongs to the Research Group on Mathematics Applied to Cryptography (MAK), at UPC. His research interests include secret sharing schemes and digital signatures in distributed scenarios.