On Unconditionally Secure Robust Distributed Key Distribution Centers

Paolo D'Arco[1] and Douglas R. Stinson[2]

[1] Department of Combinatorics and Optimization
University of Waterloo, N2L 3G1, Waterloo Ontario, Canada
pdarco@cacr.math.uwaterloo.ca
[2] School of Computer Science
University of Waterloo, N2L 3G1, Waterloo Ontario, Canada
dstinson@cacr.math.uwaterloo.ca

Abstract. A Key Distribution Center enables secure communications among groups of users in a network by providing common keys that can be used with a symmetric encryption algorithm to encrypt and decrypt messages the users wish to send to each other. A Distributed Key Distribution Center is a set of servers of a network that *jointly* realize a Key Distribution Center. In this paper we propose an unconditionally secure scheme to set up a *robust* Distributed Key Distribution Center. Such a distributed center keeps working even if some minority of the servers malfunction or misbehave under the control of a *mobile* adversary. Our scheme for a distributed key distribution center is constructed using unconditionally secure proactive verifiable secret sharing schemes. We review the unconditionally secure verifiable secret sharing scheme described by Stinson and Wei, discuss a problem with the proactive version of that scheme, and present a modified version which is proactively secure.

1 Introduction

A group of users of a network, referred to as a "conference", in order to securely communicate over public channels, could decide to use symmetric encryption algorithms, e.g., RC6 or AES. These algorithms are fast and presumed to be secure. But to apply this strategy, they need a common key with which to encrypt and to decrypt the messages they wish to send to each other. This basic problem is well-known in the literature and it is called the *Key Establishment Problem.*

A common solution to the Key Establishment Problem is to use a Key Distribution Center (KDC, for short), in which a server is responsible for the distribution and management of the secret keys. The idea is the following: Each user shares a common key with the center. When he wants to securely communicate with a subset of other users, he sends a request for a conference key. The center checks for membership of the user in that conference, and generates and distributes the conference key in encrypted form to each member of the group. Needham and Schroeder [13] initiated this approach, implemented most notably in the Kerberos System [14]. Kerberos was formally defined and studied in [1], where it is referred to as the *three-party model*.

The scheme implemented by the Key Distribution Center is called a *Key Distribution Scheme* (KDS, for short). The scheme is said to be *unconditionally secure* if it is secure independent of the computational resources of the adversary. Several kinds of Key Distribution Schemes have been considered in the literature: Key Pre-Distribution Schemes (KPS, for short), Key Agreement Schemes (KAS, for short) and Broadcast Encryption Schemes (BES, for short) among others. The reader can consult [20] for a survey on unconditionally secure schemes, [19, 11] for a general and detailed description of a variety of protocols for the Key Establishment Problem and related issues, and [3] for a simple introduction.

Our attention in this paper focuses on a model which remedies some potential weaknesses introduced by using a *single KDC*. Indeed, the main drawback of a single KDC is that it must be *trusted*. Potentially, it could eavesdrop on all the communications. Moreover, the center can be a "bottleneck" for the performance of the network and, if it crashes, secure communication cannot be supported anymore. Last but not least, even if the KDC is honest and everything works fine, the KDC still represents an attractive target to the adversary because the overall system security is lost if the KDC is compromised.

In order to solve the above problems, a new approach to key distribution was introduced in [12]. A Distributed Key Distribution Center (DKDC, for short) is a set of n servers of a network that *jointly realize* the function of a Key Distribution Center. A user who needs to participate in a conference sends a key-request to a subset of his own choosing of the n servers, and the contacted servers answer with some information enabling the user to compute the conference key. In such a model, a single server by itself does not know the secret keys, since they are *shared* between the n servers. Moreover, if some server crashes, secure communication can still be supported by the other servers and, since each user can contact a different subset of servers, the slow-down factor for the performance of the applications introduced by a single KDC can be improved.

In subsequent papers [6, 2, 4], the notion of DKDC has been studied from an information theoretic point of view. Therein, the authors showed that the protocol proposed in [12], based on bivariate polynomials, is optimal with respect to the amount of information needed to set up and manage the system.

In this paper we show how to set up a Robust DKDC. Namely, we describe a protocol where each server can *verify* that the information it stores and uses to answer the user's key-request messages is consistent with the information stored by the other servers; at the same time, the users are *guaranteed* that they can compute the same key for a given conference in which they belong to. Moreover, time is divided in *periods*, and at the beginning of each period the servers are involved in an update procedure that "refreshes" the private information they store while the conference keys they provide stay the same. This property is referred to as *proactive security*. Notice that, in [12], a simple solution was outlined, which could have been applied to the basic polynomial construction they proposed in order to achieve the above properties. However, as we show here, that solution does not work.

The design of our DKDC is based on unconditionally secure proactive verifiable secret sharing. In Section 5 we show that some existing schemes [21, 15] contain flaws. Then, we describe two techniques to modify these schemes in order to realize the proactive security property. We point out that the same ideas can be used to provide the proactive security property even to the verifiable secret sharing schemes given in [8].

2 The Model

Let $\mathcal{U} = \{U_1, \ldots, U_m\}$ be a set of m users and let $\mathcal{S} = \{S_1, \ldots, S_n\}$ be a set of n servers. Each user has *secure channels* connecting him or her to *all* the servers. Each pair of servers is connected by a *secure channel* and all of them share a *broadcast channel* and a *global clock* (i.e., the system is synchronous). Servers can be good (i.e., they honestly execute the protocol) or bad (i.e., they are controlled by an adversary and can deviate from the protocol in arbitrary ways) but a majority of good servers is always present across the system. Let $\mathcal{C}$ be the set of *conferences*, i.e., the set of groups of users which want to securely communicate, and let $\mathcal{G}$ be the set of *tolerated coalitions*, i.e., the set of coalitions of users who can try to break the scheme in some way. For example, $\mathcal{C}$ could be the set of all subsets of users of size p while $\mathcal{G}$ could be the set of all subsets of users of size q. A *verifiable distributed key distribution scheme* is divided in three phases: an *initialization phase*, which involves only the servers; a *key-request phase*, in which users ask servers for keys; and a *key-computation phase*, in which users construct keys from the messages they received from the servers who were contacted during the key-request phase.

Initialization phase. We assume that the initialization phase is performed by a *joint computation* of all the servers. Each of them, using a *private source* of randomness, r_i, generates some messages that it securely sends to the others. More precisely, for $i = 1, \ldots, n$, S_i sends to S_j some message $\gamma_{i,j}$, for each $j = 1, \ldots, n$. At the end of the distribution phase, for $i = 1, \ldots, n$, each server S_i *verifies* the information received, sends messages along the broadcast channel and, eventually, *computes and stores* some secret information $a_i = f(\gamma_{1,i}, \ldots, \gamma_{n,i})$, where f is a publicly known function. Moreover, each server constructs a list $\mathcal{L}$ of the good servers present across the network at the end of this phase (the lists held by the good servers will all contain the same identifiers).

Key-request phase. Let $C_h \in \mathcal{C}$ be a conference. Each user U_j in C_h contacts a subset of a certain size of good servers belonging to $\mathcal{L}$, requesting a key for the conference C_h. We denote this key by κ_h. Each good server S_i, contacted by user U_j, checks[3] for membership of U_j in C_h; if $U_j \in C_h$, then S_i computes

[3] We do not consider the underlying authentication mechanism involved in a key request phase.

a value $y_{i,j}^h = F(a_i, j, h)$, where F is a publicly known function. Otherwise, S_i sets $y_{i,j}^h = \perp$ (a special value which conveys no information about κ_h). Finally, S_i sends the value $y_{i,j}^h$ to U_j. Note that a bad server can either refuse to reply or it may send some incorrect value.

Key-computation phase. Having received the values from the servers, each user U_j in C_h computes κ_h from a certain majority of the values received.

Roughly speaking, a Verifiable DKDC must satisfy the following properties:

- **Correct and Verifiable Initialization Phase.** When the initialization phase successfully terminates, any good server S_i must be able to identify the subset of good servers and to compute his private information a_i.
- **Consistent Key Computation.** Each user in a conference $C_h \subseteq \mathcal{U}$ must be able to compute *the same* conference key, after interacting with a subset of good servers of a certain size.
- **Conference Key Security.** A conference key must be secure against attacks performed by coalitions of bad servers, coalitions of users, and hybrid coalitions of a certain size consisting of servers and users.

Let b be an upper bound on the number of bad servers during any phase of the protocol, and let $t > b$ denote a sufficient number of parties to reconstruct a possible conference key. In a more precise way, we state the following definition:

Definition 1. *Let b, t and n be integers such that $n \geq t$ and $t > b$. Let $\mathcal{U} = \{U_1, \ldots, U_m\}$ be a set of m users, and let $\mathcal{S} = \{S_1, \ldots, S_n\}$ be a set of n servers. Finally, let $\mathcal{C} \subseteq 2^{\mathcal{U}}$ be the set of conferences and let $\mathcal{G} \subseteq 2^{\mathcal{U}}$ be the set of tolerated coalitions. A verifiable $(b, t, n, \mathcal{C}, \mathcal{G})$-Distributed Key Distribution Scheme (for short, $(b, t, n, \mathcal{C}, \mathcal{G})$-VDKDS) is a three-phase protocol consisting of an Initialization Phase, a Key Request Phase, and a Key Computation Phase, which enables each user in $C_h \in \mathcal{C}$ to compute a common key κ_h by interacting with at least $n - b$ servers of the network. More precisely, the following properties are satisfied:*

1. *After the initialization phase, each good server computes his private information and verifies its consistency with the information received and stored by the other good servers. At least $n - b$ servers successfully complete this phase and each of them construct the same (public) list $\mathcal{L}$ containing the identities of the good servers.*
2. *Each user in $C_h \in \mathcal{C}$ can compute the common key κ_h by contacting the servers in $\mathcal{L}$. At least $|\mathcal{L}| - b > t$ of the $|\mathcal{L}|$ servers give good answers, from which the user reconstructs the key. Any t good answers are sufficient to reconstruct the key.*
3. *Each conference key is completely secure against coalitions of users $G \in \mathcal{G}$; coalitions of servers of size less than b; and joint coalitions of at most b servers and users in a subset $G \in \mathcal{G}$.*

Basically, in the above model, we assume that at most b servers can misbehave during the initialization phase of the system, and during the key request phase. Moreover, these two subsets of bad servers can be different: in other words, we assume that the adversary is *mobile*. Notice that a crash of some of the servers in the above model can be seen as a simple type of misbehavior.

3 A Verifiable Secret Sharing Scheme

The main component of our $(b, t, n, \mathcal{C}, \mathcal{G})$-VDKDS is a Verifiable Secret Sharing Scheme (VSS, for short). Loosely speaking, in a VSS Scheme, a Dealer shares a secret among a set of participants in such a way that each participant can verify if the shares he gets, from the dealer during the distribution phase and from the other participants during the recovering phase, are consistent with the secret. VSS schemes were introduced in [5].

In this section we describe the VSS we are going to use. It is a slightly modified version of the scheme proposed by Stinson and Wei in [21], whose round complexity has been improved by applying the technique recently described in [8]. Due to the use of a symmetric polynomial, the scheme of [21], enhanced with the ideas of [8], is a bit more efficient than the scheme described in [8] with the same parameters (i.e., when $b < \frac{n}{4}$). Notice that, in the following construction, the dealer, after sending messages during the initialization phase, becomes inactive. In fact, as we will argue later, he can be completely substituted by a joint computation performed by the servers of the system.

First of all, we recall the definition of a VSS.

Definition 2. *Let $\mathcal{D}$ be a dealer and let $P_1, \ldots, P_n$ be n participants connected by secure channels and having access to a broadcast channel. Moreover, let $\mathcal{A}$ be an adversary that can corrupt up to b of the participants (including the dealer). Assume that π is a protocol consisting of two phases,* Share *and* Reconstruct*, and let S be a set of possible secret values. At the beginning of* Share*, the dealer inputs a secret $s \in S$. At the end of* Share *each participant P_i outputs a boolean value ver_i. At the end of* Reconstruct *each participant outputs a value in S. The protocol π is an (n, t, b, S)* Unconditionally Secure Verifiable Secret Sharing Scheme *if the following properties are satisfied:*

1. *If a good player P_i outputs $ver_i = 0$ at the end of* Share*, then every good player outputs $ver_i = 0$.*
2. *If the dealer is good, then $ver_i = 1$ for every good P_i.*
3. *If at least $n - b$ players P_i output $ver_i = 1$ at the end of* Share*, then there exists an $s' \in S$ such that the event that all good P_i output s' at the end of* Reconstruct *is fixed at the end of* Share *and $s' = s$ if the dealer is good.*
4. *If $|S| = q$, s is chosen randomly from S and the dealer is good, then any coalition of at most $t - 1$ participants cannot guess, at the end of* Share*, the value s with probability greater than $\frac{1}{q}$.*

The scheme we are going to use works as follows: let t, b be two integers such that $n \geq t+3b$ and $t > b$. Let $S = GF(q)$ be a finite field and let ω be a primitive element in $GF(q)$. All computations are done in the field $GF(q)$.

Share.

- When $\mathcal{D}$ wants to share a secret value $s \in S$, he chooses a random symmetric polynomial
$$f(x,y) = \sum_{i=0}^{t-1}\sum_{j=0}^{t-1} a_{ij}x^i y^j,$$
where $a_{00} = s$ and $a_{ij} = a_{ji}$ for all i, j. Then, for each k, $\mathcal{D}$ sends $h_k(x) = f(x, \omega^k)$ to P_k through a secure channel. At the same time, for each i, P_i generates and sends to every P_k a random value $r_{ik} \in GF(q)$ through a secure channel.
- After receiving $h_k(x)$ from $\mathcal{D}$ and $r_{1k}, \ldots, r_{nk}$ from the other participants, each P_k broadcasts the value $h_k(\omega^\ell) + r_{k\ell} + r_{\ell k}$, for each $\ell \neq k$.
- Each P_i computes the maximum subset $G \subseteq \{1, \ldots, n\}$ such that any ordered pair $(\ell, k) \in G \times G$ is *consistent*, i.e. such that $h_k(\omega^\ell) + r_{k\ell} + r_{\ell k} = h_\ell(\omega^k) + r_{\ell k} + r_{k\ell}$. If $|G| \geq n - b$, then P_i outputs $ver_i = 1$. Otherwise, P_i outputs $ver_i = 0$.

Reconstruct

- Each P_i sends $h_i(0)$ to each P_k, where $i \in G$, the set of good participants after *Share*.
- After receiving the $h_i(0)$'s, P_k computes a polynomial $f_k(0, y)$ such that $f_k(0, \omega^i) = h_i(0)$ for at least $n - 2b$ of the data he has received. This operation can be done efficiently, for example, either using the methods described in [18], or using error correction techniques for Reed-Solomon Codes [10].
- P_k computes and outputs $s' = f_k(0, 0)$.

The security of the protocol can be shown along the same line of Theorem 2 in [21]. Our only change in the protocol is to *Share* where, instead of using the secure channels for the check of consistency of the shares the dealer $\mathcal{D}$ distributes, we use random one-time pads and the broadcast channel as in [8]. With this trick we save one round of communication, compared to [21].

4 A Verifiable Distributed Key Distribution Scheme

Using the VSS described in the previous section, we describe a simplified version of a Verifiable DKDS. We assume that a Dealer $\mathcal{D}$ initializes the system but, as we will show later, this assumption can be easily removed. Our scheme provides ℓ-wise independent conference keys, i.e., the ℓ-th conference key is uniformly distributed over the set of possible values, even if an adversary already knows $\ell - 1$ previous conference keys. It works as follows:

Set up Phase.

- Let ℓ_G be the maximum number of conference keys that a group G can compute. Assume that $\ell > max_{G \in \mathcal{G}} \ell_G$. The dealer chooses a *random* polynomial $K(x) = \sum_{z=0}^{\ell-1} k_z x^z$. The conference key for C_s is defined by $\kappa_s = K(s)$.
- Then, for each coefficient k_z of $K(x)$ the dealer runs ℓ independent copies Σ_z of the VSS described before, where the secret that Σ_z distributes among the servers is k_z.
- Each server S_i stores the ℓ univariate polynomials $h_i^{k_0}(x), \ldots, h_i^{k_{\ell-1}}(x)$ sent by the dealer during the executions of the Share Phase of the Σ_z's, and publishes the list of good servers he has found.

In a VSS, the reconstruction of the secret is done by the participants (i.e., the servers in our setting) while in a DKDS each user of a given conference contacts the servers, receives some information and computes the common key by applying a public function to the values received. A straightforward "solution" to this different scenario could be that each server S_i sends, according to the VSS scheme, the values of his polynomials evaluated in zero, i.e., $h_i^{k_0}(0), \ldots, h_i^{k_{\ell-1}}(0)$, to the users. But this is insecure, because, in this case, the user reconstructs *all* the keys! Thus, we need a different approach. Basically, the values sent by the servers must enable them to compute a single key, namely, the one the user is asking for.

Key Request and Key Computation Phases.

- User $U_j \in C_s$ asks a subset of good servers of size at least $n - b$ for the key κ_s.
- Each server S_i computes

$$\overline{h_i}(0) = h_i^{k_0}(0) + h_i^{k_1}(0)s + \cdots + h_i^{k_{\ell-1}}(0)s^{\ell-1},$$

  and sends $\overline{h_i}(0)$ to the user.
- The user interpolates a polynomial $\overline{h}(x)$ such that $\overline{h}(\omega^i) = \overline{h_i}(0)$ for at least $n - 2b$ of the values received. Then, he recovers $\kappa_s = \overline{h}(0)$.

Correctness. The correctness of the construction can be shown as follows: according to the VSS scheme described in the previous section, each coefficient of $K(x)$ can be recovered by applying the Lagrange formula. More precisely, assuming that the first t servers are good servers, we have

$$k_j = \sum_{i=1}^{t} h_i^{k_j}(0) b_i, \qquad \text{where} \qquad b_i = \prod_{k \neq i} \frac{\omega^k}{\omega^k - \omega^i}.$$

Notice that,

$$
\begin{aligned}
K(s) &= k_0 + k_1 s + \cdots + k_{\ell-1} s^{\ell-1} \\
&= \sum_{i=1}^{t} h_i^{k_0}(0) b_i + (\sum_{i=1}^{t} h_i^{k_1}(0) b_i) s + \cdots + (\sum_{i=1}^{t} h_i^{k_{\ell-1}}(0) b_i) s^{\ell-1} \\
&= \sum_{i=1}^{t} (h_i^{k_0}(0) + h_i^{k_1}(0) s + \cdots + h_i^{k_{\ell-1}}(0) s^{\ell-1}) b_i = \sum_{i=1}^{t} \overline{h_i}(0) b_i = \overline{h}(0) = \kappa_s.
\end{aligned}
$$

In general, since the user does not know *a priori* which servers send correct values, he needs to interpolate a polynomial $\overline{h}(x)$ which agrees with at least $n - 2b$ of the values received, which can be done efficiently by applying the techniques given in [18] or in [10], exactly as in the VSS. Finally, he recovers the common key by evaluating $\overline{h}(x)$ at $x = 0$.

A One-Time Scheme (Toy Example). In order to give to the reader a concrete idea of the protocol, let us consider the following example: let $q = 7$, $\omega = 3$, $n = 5$, $t = 2$ and $b = 1$. The dealer defines the keys as points belonging to $K(x) = 3 + 5x (\text{mod} 7)$ and, to share the coefficients 3 and 5, he chooses two symmetric bivariate polynomials, say

$$f^1(x, y) = 3 + 5x + 5y + 3xy \quad \text{and} \quad f^2(x, y) = 5 + 4x + 4y + 4xy.$$

Therefore server S_i, whose public identity is defined by ω^i, gets two polynomials $h_i^1(x) = f^1(x, \omega^i)$ and $h_i^2(x) = f^2(x, \omega^i)$. More precisely, the polynomial distributed are listed in the following table

Server	identifier	$h_i^1(x)$	$h_i^2(x)$
S_1	$\omega^1 = 3$	4	$3 + 2x$
S_2	$\omega^2 = 2$	$6 + 4x$	$6 + 5x$
S_3	$\omega^3 = 6$	$5 + 2x$	1
S_4	$\omega^4 = 4$	$2 + 3x$	$4x$
S_5	$\omega^5 = 5$	$6x$	$4 + 3x$

The value of the conference key $\kappa_3 = 3 + 5 \times 3 \bmod 7 = 4$. Assume that servers S_1 and S_2, belonging to the list $\mathcal{L}$ of good servers, send to a user in C_3 correct values in order to enable him to recover κ_3. More precisely, the user gets from S_1 the value $4 + 3 \times 3 = 6$ and the value $6 + 6 \times 3 = 3$ from S_2. Using the public identifiers of S_1 and S_2, the user sets up the two pairs of values $(3, 6), (2, 3)$, and by applying the Lagrange Formula, he interpolates the polynomial $P(x) = 3 \times x - 3 \bmod 7$. It is easy to see that $P(0) = 4$, and hence the user recovers κ_3. Moreover, assuming that S_5 was bad in the set up phase, the user gets from the other "supposed to be good" servers S_3 and S_4 the values $2 + 0 \times 3 = 2$ and $0 + 4 \times 3 = 5$ (if they are honest). These values belong to the polynomial interpolated. Notice that, assuming that S_1 and S_2 send correct values (i.e., are honest) and since at most one server (i.e., $b = 1$) can send an incorrect value during the key request phase, at least one of the values send by S_3 and S_4 must agree with $P(x)$.

Security. The security of the protocol can be shown by considering the following possible cases:

- *Coalition of users.* As long as a group $G \in \mathcal{G}$ does not recover more than ℓ conference keys and, more precisely, does not obtain information from the servers for more than ℓ conference keys, the group cannot compute any information about another conference key in an information theoretic sense. This property easily follows from the assumption that the conference keys are values of a polynomial of degree $\ell - 1$ (i.e., they are ℓ-wise independent). By the ℓ-wise independence, it is easy to see that a coalition holding $\ell - 1$ pairs (s, κ_s), for any choice of an ℓ-th pair $(s', \kappa_{s'})$ can interpolate a *different polynomial* of degree $\ell - 1$. Hence, the ℓ-th key is unconditionally secure.
- *Coalition of servers.* By the property of the VSS, any coalition of b servers, even putting together all the information received during the set up phase, cannot compute any information about any conference key, because each coefficient of the polynomial determining the keys is shared in the VSS by a t-degree polynomial, where $t > b$. Moreover, users reconstruct the conference keys even if at most b servers are bad during the initialization phase and at most b (possibly different) servers send incorrect information to the users during the key-request phase. Hence, in this case, the security follows from the security of the VSS (see, e.g., Theorem 2 in [21]).
- *Coalitions of users and servers.* The worst scenario we have to consider is when b servers collude with a group of users $G \in \mathcal{G}$ who has run the protocol many times, recovering a bunch of conference keys. For example, assuming that the bad servers are $S_1, \ldots, S_b$, the information the coalition possesses is given by the partial polynomials $h_1^{k_0}(x), \ldots, h_1^{k_{\ell-1}}(x), \ldots, h_b^{k_0}(x), \ldots, h_b^{k_{\ell-1}}(x)$ plus the values received by the users during the previous executions of the protocol in order to retrieve some conference keys. As the previous cases have shown, the two types of information by themselves are useless in order to find out information about a new key. However, it is not difficult to see that even the *joint* knowledge of this information does not help, since the coalition does not have "enough points" to interpolate a new key. Actually, any new key can still assume any possible value, for each choice of the values that should be provided by a group of at least $t - b$ other servers (i.e., perfectly secure).

Remark. The Dealer $\mathcal{D}$ can be easily removed from the above protocol since it can be removed from the VSS scheme, as shown in [21]: each participant, during *Share*, chooses a different secret value and executes the protocol. The *real* shared value is given by the *sum* of the values chosen by the good participants. Along the same line, each server of the system, during the initialization phase of the VDKDS can act as the dealer, choosing a different polynomial. In this case *the keys are points of the polynomial obtained by summing up the polynomials chosen* *by the good servers*. The presence of the dealer has been used only to simplify the description of the protocol. Moreover notice that the assumption that there are at most b bad servers in this scenario implies that the Share Phase of the VSS protocol is *always successfully* completed by the good servers.

5 Proactivity

The concept of *proactive security* was introduced in [17] and applied to the secret sharing setting in [9]. Basically the idea is that, if the information stored by the servers in order to share a given secret *stays the same* for all the lifetime of the system, then an adversary can eventually break into a sufficient number of servers to learn or destroy the secret. On the other hand, if time is divided into *periods*, and at the beginning of each period the information stored by the servers in a given time period changes (while the shared secret stays the same), then the adversary probably does not have enough time to break into the necessary number of servers. Moreover, the information he learns during period p is useless during period $p+i$, for $i = 1, 2 \ldots,$. So, he has to start a new attack from scratch during each time period.

The design of a Proactive VDKDS easily follows once we have a Proactive VSS. Therefore, in the following subsections we address the construction of unconditionally secure proactive VSS. Notice that, in [12], a simple solution to set up a Proactive VDKDS was given, but as we show in Appendix A, it does not work.

5.1 An Unconditionally Secure Proactive VSS for $b \leq \frac{n}{4} - 1$ bad servers

The first unconditionally secure proactive VSS was proposed by Stinson and Wei in [21], where proactivity is added to the basic VSS described before. A generalization of that scheme has subsequently been given in [15]. We start by analyzing a weakness of the scheme given in [21], and we show how it can be used to attack the proactive security property. Then, we show a variation of the scheme that solves the problem. Moreover, we describe another technique that can be used to add proactive security to both VSSs given in [21] and [8] for the case in which the number of bad servers is $b \leq \frac{n}{4} - 1$.

Let $t > b + 1$. We assume that time is divided in periods $p = 1, 2, ...,$. Each good server, at the beginning of the new period, performs the steps given in the table of the next page to renew the shares [21]. Unfortunately, the symmetry of the polynomial $r^\ell(x, y)$ can be used by bad servers to break the scheme. Indeed, during step 2, server P_ℓ broadcasts the polynomial $h_0^\ell(x) = r^\ell(x, 0) = r^\ell(0, y)$. Hence, any server can compute the values $h_0^\ell(\omega^k) = r^\ell(0, \omega^k) = h_k^\ell(0)$ for $k = 1, \ldots, n$. Then, in step 6, each good player P_m updates his own share $h_m(x)$ by adding the $\sum_{k \in \mathcal{L}} h_m^k(x)$. At this point notice that, according to the VSS, the only part of the share $h_m(x)$ used to reconstruct the secret is $h_m(0)$, the first coefficient of the polynomial, which is updated by $\sum_{k \in \mathcal{L}} h_m^k(0)$.

Renewal

1. Each server P_ℓ selects a random symmetric polynomial

$$r^\ell(x,y) = \sum_{i=0}^{t-1}\sum_{j=0}^{t-1} r_{i,j}x^i y^j,$$

where $r_{00} = 0$ and $r_{ij} = r_{ji}$ for all i, j.
2. P_ℓ sends $h_k^\ell(x) = r^\ell(x, \omega^k)$ to P_k for $k = 1, 2, \ldots, n$ by a secure channel, and broadcasts $h_0^\ell(x) = r^\ell(x, 0)$.
3. P_k checks whether $h_0^\ell(0) = 0$ and $h_k^\ell(0) = h_0^\ell(\omega^k)$. If the conditions are satisfied, then P_k computes and sends to P_m the value $h_k^\ell(\omega^m)$. Otherwise P_k broadcasts an accusation of P_ℓ.
4. P_m checks whether $h_m^\ell(\omega^k) = h_k^\ell(\omega^m)$ for all values of ℓ not accused by $n - b$ servers of the system. If the equation is not true for more than b values of k, then P_m broadcasts an accusation of P_ℓ.
5. If P_ℓ is accused by at most b servers, then he can defend himself as follows: For those P_i he is accused by, P_ℓ broadcasts $h_i^\ell(x)$. Then, server P_k checks whether $h_i^\ell(\omega^k) = h_k^\ell(\omega^i)$ and broadcasts "yes" or "no". If there are at least $n - b - 2$ servers broadcasting yes, then P_ℓ is not a bad server.
6. P_m updates the list of good servers $\mathcal{L}$ (i.e., all the values ℓ for which P_ℓ is accused by at least $b+1$ servers, or found bad in the previous step are not in $\mathcal{L}$). Then, P_m updates its shares as

$$h_m(x) \leftarrow h_m(x) + h_m^k(x)$$

for all $k \in \mathcal{L}$.

But this sum can be computed by everybody using the public broadcasts in step 2. The consequence is that if a *passive adversary* breaks into server P_m during period p, he can still use the share $h_m(0)$ during periods $p+i$ because he can compute all the updates for this coefficient performed between period p and period $p+i$. More precisely, if the adversary learns the shares $h_1(0), \ldots, h_b(0)$ held by $S_1, \ldots, S_b$ during period p, and he learns $h_{b+1}(0), \ldots, h_{b+s}(0)$ held by $S_{b+1}, \ldots, S_{b+s}$ during period $p+1$ (the adversary is mobile), then, he can compute the new shares held by $S_1, \ldots, S_b$ during period $p+1$ from $h_1(0), \ldots, h_b(0)$ and the broadcasts of period $p+1$, and if $b + s \geq t$ he can recover the secret. Hence, the proactive security property is lost because the renewal scheme does not render useless the shares learnt during the previous period. Exactly the same strategy can be applied to break the Renewal procedure given in [15], which is a generalization of the one given in [21].

Basically, the problem in the above procedure is due to the *broadcast* in Step 2 of $h_0^\ell(x)$, needed to verify that the update does not destroy the secret, and the symmetry of $r^\ell(x,y)$. We propose two solutions. The first one changes the structure of the renewal phase in order to avoid the broadcast. The second keeps the same structure as before, but removes the symmetry property of $r^\ell(x,y)$. Let us describe the first approach: We would like to refresh the shares still by summing up "new shares" derived from a random symmetric polynomial $r(x,y) = \sum_{i=0}^{t-1}\sum_{j=0}^{t-1} r_{i,j}x^iy^j$ whose known coefficient is $r_{0,0} = 0$. Indeed, this property guarantees that the secret stays the same. However, some server P_ℓ can be bad and can choose a polynomial $r^\ell(x,y)$ where $r_{0,0} \neq 0$. In order to prevent this problem, avoiding the broadcast, we generate $r(x,y)$ as $r(x,y) = (x+y)r^*(x,y)$, where $r^*(x,y) = \sum_{i=0}^{t-2}\sum_{j=0}^{t-2} r^*_{i,j}x^iy^j$ is given by the sum of the partial choices $r^\ell(x,y) = \sum_{i=0}^{t-2}\sum_{j=0}^{t-2} r^\ell_{i,j}x^iy^j$ of the good players P_ℓ, and the term $(x+y)$ is introduced by each server through a private computation. In this way, the condition $r_{0,0} = 0$ is surely satisfied and the polynomial remains symmetric. From a technical point of view, the degree of $r(x,y)$ must be $t-1$, in order to enable the reconstruction of the secret. Hence, due to the generation rule for $r(x,y)$, every $r^\ell(x,y)$ must have degree $t-2$.

Renewal

1. Each server P_ℓ selects a random symmetric polynomial
$$r^\ell(x,y) = \sum_{i=0}^{t-2}\sum_{j=0}^{t-2} r_{i,j}x^iy^j,$$
where $r_{ij} = r_{ji}$ for all i,j.
2. P_ℓ sends $h^\ell_k(x) = r^\ell(x,\omega^k)$ to P_k for $k = 1,2,\ldots,n$ by a secure channel.
3. After receiving $h^\ell_k(x)$, P_k computes and sends the value $h^\ell_k(\omega^m)$ to P_m, for $m = 1,\ldots,n$, by a secure channel.
4. P_m checks whether $h^\ell_m(\omega^k) = h^\ell_k(\omega^m)$ for $k = 1,\ldots,n$. If the equation is not true for more than b values of k, then P_m broadcasts an accusation of P_ℓ.
5. If P_ℓ is accused by at most b servers, then he can defend himself as follows: For those P_i he is accused by, P_ℓ broadcasts $h^\ell_i(x)$. Then, server P_k checks whether $h^\ell_i(\omega^k) = h^\ell_k(\omega^i)$ and broadcasts "yes" or "no". If, for every broadcasted $h^\ell_i(x)$, there are at least $n-b-2$ servers broadcasting yes, then P_ℓ is not a bad server. In this case, if P_i has an $h^\ell_i(x)$ different from the one that P_ℓ has broadcasted, then he stores the broadcasted one.
6. P_m updates the list $\mathcal{L}$ of good servers (i.e., the servers found bad in the previous step are not in $\mathcal{L}$) and updates his share as
$$h_m(x) \leftarrow h_m(x) + (x+\omega^m)h^*_m(x)$$
where $h^*_m(x) = \sum_{\ell\in\mathcal{L}} h^\ell_m(x)$.

Notice that the above procedure is a slightly revised version of the one we initially proposed [7]: it incorporates the observations and the work done by Nikov et al. [16] on our preprint [7]. See [16] for details.

Security (Sketch). The security of the above protocol can be shown by proving that the secret stays the same and the update of the shares cannot be computed by a coalition of bad servers. Concerning the first property, notice that the secret s is shared by the VSS by means of $h_1(x), \ldots, h_n(x)$. More precisely, it is the first coefficient of the polynomial $h(x, 0)$. Since during *Renewal* each server P_m computes a new share as $h_m(x) \leftarrow h_m(x) + (x + \omega^m)h_m^*(x)$, implicitly the secret becomes the first coefficient of the new polynomial $h(x, 0) + xh^*(x, 0)$, where $xh^*(x, 0)$ is zero when evaluated at $x = 0$. Hence, the secret stays the same.

About the security of the update, notice that if the adversary controls b servers, say $S_1, \ldots, S_b$, he can compute at most $b < t-1$ points $h_m^*(\omega^1), \ldots, h_m^*(\omega^b)$, which give no information about the polynomial $h_m^*(x)$ used by P_m to update his share for any $m \notin \{1, \ldots, b\}$. Moreover, due to the random choices performed at each executions of *Renewal*, it is not difficult to check that the adversary cannot use the information learnt in period p during period $p+1$ or in any other period. Finally notice that, during step 4, a good server P_ℓ, in order to defend himself, broadcasts at most b polynomials $h_k^\ell(x)$, corresponding to the P_k he is accused by. Assuming that $t > b + 1$, the polynomials broadcasted give no information about $r^\ell(x, y)$. This implies again that an adversary can gain no information about $h_m^\ell(x)$, for every P_m not belonging to the coalition of corrupted servers.

During each time period, the servers need to check if some of them have been corrupted by the adversary. Indeed, those servers should be rebooted[4] in order to recover a correct functionality. The following procedures enable the detection of corrupted servers and the recovering of good shares, once the corrupted servers have been rebooted [21].

Detection

1. P_ℓ computes and sends $h_\ell(\omega^k)$ to P_k for $k = 1, 2, \ldots, n$ by secure channels.
2. P_k checks whether $h_\ell(\omega^k) = h_k(\omega^\ell)$. P_k then broadcasts an accusation $list_k$ which contains those ℓ such that $h_\ell(\omega^k) \neq h_k(\omega^\ell)$ or $h_\ell(\omega^k)$ was not received.
3. Each good server updates the list $\mathcal{L}$ so that it does not contain those ℓ accused by at least $b + 1$ servers of the system.

Recovery

1. For each $\ell \notin \mathcal{L}$, every good server P_i computes and sends $h_i(\omega^\ell)$ to P_ℓ.
2. Upon receiving the data, P_ℓ computes the polynomial $h_\ell(x)$ that agree with the majority of the values $h_\ell(\omega^k)$ he has received. P_ℓ sets $h_\ell(x)$ as his new share.

[4] We can assume that there is a distributed rebooting scheme enabling a majority of servers to decide to reboot some other servers when they detect that such servers have been corrupted. Otherwise, the system manager who installs the programs, is alerted by the good servers and reboots the bad ones [9].

To understand the above procedures, notice that, when the secret is shared by means of the VSS, the shares held by P_i and P_j satisfy $h_i(\omega^j) = h_j(\omega^i)$. This property even holds for the polynomials $h_i^*(x)$ and $h_j^*(x)$ generated during *Renewal*. Moreover, due to the choice of the updating rule, i.e., $h_m(x) \leftarrow h_m(x) + (x + \omega^m)h_m^*(x)$, the symmetry $h_i(\omega^j) = h_j(\omega^i)$ is still mantained after every update phase.

These three protocols provide the VSS scheme described in the previous section with proactive security, and they can be used, as we show later, to set up a proactive VDKDS.

The second approach for adding proactive security to the basic VSS given in [21] relies on the use of a generic (non-symmetric) polynomial $r^\ell(x, y)$. Let us consider the following procedure:

Renewal

1. Each server P_ℓ selects a random polynomial
$$r^\ell(x, y) = \sum_{i=0}^{t-1} \sum_{j=0}^{t-1} r_{i,j} x^i y^j,$$
where $r_{00} = 0$.
2. P_ℓ sends $f_k^\ell(x) = r^\ell(x, \omega^k)$ and $g_k^\ell(y) = r^\ell(\omega^k, y)$ to P_k by a secure channel, and broadcasts $g_0^\ell(x) = r^\ell(x, 0)$.
3. P_k checks whether $g_0^\ell(0) = 0$, $g_0^\ell(\omega^k) = g_k^\ell(0)$, and $f_k^\ell(\omega^k) = g_k^\ell(\omega^k)$. If the conditions are satisfied, then P_k computes and sends the value $f_k^\ell(\omega^m)$ to P_m by a secure channel, for $m = 1, \ldots, n$. Otherwise, P_k broadcasts an accusation of P_ℓ.
4. P_m checks whether $f_k^\ell(\omega^m) = g_m^\ell(\omega^k)$ for all values of ℓ not accused by $n - b$ servers of the system. If the equation is not true for more than b values of k, then P_m broadcasts an accusation of P_ℓ.
5. If P_ℓ is accused by at most b servers, then P_ℓ can defend himself as follows. For those P_k he is accused by, P_ℓ broadcasts $f_k^\ell(x)$ and $g_k^\ell(y)$. Then, server P_i checks whether $g_i^\ell(\omega^k) = f_k^\ell(\omega^i)$, $g_k^\ell(\omega^i) = f_i^\ell(\omega^k)$, and broadcast "yes" or "no". If, for every broadcasted pair of polynomials $(f_k^\ell(x), g_k^\ell(y))$, there are at least $n - b - 2$ servers broadcasting yes, then P_ℓ is not a bad server. In this case, if P_k has a pair $(f_k^\ell(x), g_k^\ell(y))$ different from the one that P_ℓ has broadcasted, then he stores the broadcasted one.
6. P_m updates the list of good servers $\mathcal{L}$ (i.e., the values ℓ for which P_ℓ is accused by at least $b+1$ servers, or found bad in the previous step are not in $\mathcal{L}$). Then, P_m updates his share as
$$h_m(x) \leftarrow h_m(x) + f_m^k(x)$$
for all $k \in \mathcal{L}$. Moreover, he updates his information for verification (which is $g_m(y) = h_m(x)$ at the first execution of *Renewal*) by setting
$$g_m(y) \leftarrow g_m(y) + g_m^k(y)$$
for all $k \in \mathcal{L}$. This information is used in the *Detection* procedure.

Security (Sketch). The security of the protocol follows from the following observations: first of all notice that from the broadcast $g_0^\ell(x) = r^\ell(x,0) \neq r^\ell(0,y)$, the value $f_k^\ell(0) = r^\ell(0,\omega^k)$ cannot be computed. Moreover, every participant, during steps 3 and 4, checks that the update does not destroy the shared secret, and that the polynomials they have received are consistent. Moreover, as we have already seen before, the condition $t > b+1$ ensures that the polynomials broadcasted in step 5 by P_ℓ, to defend himself against at most b bad P_i, do not give any information about $r^\ell(x,y)$, and hence do not give any information about $f_m^\ell(x)$ for any P_m not belonging to the coalition of bad servers.

This procedure can be applied to both VSSs given in [21] and [8] when $b \leq \frac{n}{4} - 1$. In fact, when applied to the scheme in [8], the polynomial $g_m(y)$ in Step 5 at the first execution of *Renewal* is already different from $h_m(x)$: it is the polynomial $g_m(y)$ used for verification given by the VSS described in [8]. Actually, the above procedure has the structure of the procedure given in [21], but it has been modified according to the design of the VSS given in [8].

The following protocols enable the detection of corrupted servers and the recovering of good shares for the rebooted servers.

Detection

1. P_ℓ computes and sends $h_\ell(\omega^k)$ to P_k for $k = 1, 2, \ldots, n$ by secure channels.
2. P_k checks whether $h_\ell(\omega^k) = g_k(\omega^\ell)$. P_k then broadcasts an accusation $list_k$ which contains those ℓ such that $h_\ell(\omega^k) \neq g_k(\omega^\ell)$ or $h_\ell(\omega^k)$ was not received.
3. Each good server updates the list $\mathcal{L}$ so that it does not contain those ℓ accused by at least $b+1$ servers of the system.

Recovering

1. For each $\ell \notin \mathcal{L}$, every good server P_i computes and sends $h_i(\omega^\ell)$ and $g_i(\omega^\ell)$ to P_ℓ.
2. Upon receiving the data, P_ℓ computes two polynomials $h_\ell(x)$ and $g_\ell(y)$ that agree with the majority of the values $h_\ell(\omega^k)$ and $g_\ell(\omega^k)$ it has received. P_ℓ sets $h_\ell(x)$ as its share and $g_\ell(y)$ as its verification information.

We would like to point out that both the Renewal phases described before can be implemented by using *random one-time pads* and the *broadcast channel*, instead of using secure channels for the checks of consistency of the shares. Such an approach enables saving one round of communication, but the resulting procedures are perhaps less readable than the previous ones.

5.2 A Proactive VDKDS

At this point, we have all the tools to set up a Proactive VDKDS. To summarize:

– In the protocol for a VDKDS, described in Section 4, the keys are values of a polynomial whose coefficients are (verifiably) shared among the servers. More precisely, to set up the DKDC, each server P_m chooses a *random* polynomial

$$K^m(x) = \sum_{z=0}^{\ell-1} k_z^{(m)} x^z.$$

Then, P_m uses ℓ different instances of the VSS given in Section 3, i.e., one for each coefficient, to distribute in a verifiable way the coefficients of his polynomial $K^m(x)$. According to the VSS, each server P_k receives ℓ polynomials from P_m, one for each coefficient $k_z^{(m)}$. The conference key for C_s is then defined to be $\kappa_s = K(s)$, where

$$K(x) = \sum_{z=0}^{\ell-1} k_z x^z = \sum_{m \in \mathcal{L}} K^m(x)$$

and $\mathcal{L}$ is the list of good servers. At the end of the set up phase, every server P_k stores ℓ polynomials, $h_k^{k_0}(x), \ldots, h_k^{k_{\ell-1}}(x)$, each sharing one coefficient of $K(x)$, by summing up the partial shares/polynomials received for each coefficient $k_z^{(m)}$ from servers P_m belonging to the list of good servers.

– Therefore, a straightforward solution to gain proactive security could be to directly apply, at the beginning of each time period, the *Detection, Recovery* and *Renewal* procedures for each coefficient of the polynomial $K(x)$, generated by the good servers during the set up phase of the system.

6 Conclusions

In this paper we have shown how to set up a Robust Distributed Key Distribution Scheme, enabling a set of servers to jointly realize a Key Distribution Center. We have used unconditionally secure verifiable proactive secret sharing schemes as a building block. As well, we have revised the unconditionally secure VSS described by Stinson and Wei in [21], proposing a modified version which is proactively secure. Moreover, we have given proactive routines that can be applied to both schemes given in [21, 8] when $b < \frac{n}{4}$. Since the proactive security property can be useful in several settings in which the adversary is mobile, the applicability of such schemes has independent interest of the specific application to key distribution that has been addressed in this paper. In the full version of the paper we will provide complete proofs, and the case in which the number of bad servers is $b < \frac{n}{3}$ will be considered as well.

Acknowledgements

We would like to thank Svetla Nikova, Ventzislav Nikov and Ruizhong Wei for helpful comments and discussions during the writing of this paper. D. R. Stinson's research is supported by NSERC grants IRC # 216431-96 and RGPIN # 203114-02.

References

1. M. Bellare and P. Rogaway, *Provably Secure Session Key Distribution: The Three Party Case*, Proc. of the 27th Annual Symposium on the Theory of Computing (STOC '95), ACM, pp. 57–66, 1995.
2. C. Blundo, and P. D'Arco, *Unconditionally Secure Distributed Key Distribution Schemes*, submitted for publication.
3. C. Blundo and P. D'Arco, *The Key Establishment Problem*, Lecture Notes in Computer Science, FOSAD 2001 (Tutorial), to appear.
4. C. Blundo, P. D'Arco, V. Daza and C. Padrò. *Bounds and Constructions for Unconditionally Secure Distributed Key Distribution Schemes for General Access Structures*, Proc. of the Information Security Conference (ISC 2001), Lecture Notes in Computer Science, vol. 2200, pp. 1–17, 2001.
5. B. Chor, S. Goldwasser, S. Micali, and B. Awerbach. *Verifiable Secret Sharing and Achieving Simultaneity in Presence of Faults*, Proc. of the 26-th Annual Symposium on the Foundations of Computer Science, IEEE, pp. 383–395, 1985.
6. P. D'Arco, *On the Distribution of a Key Distribution Center* (extended abstract), Proc. of the Italian Conference on Theoretical Computer Science (ICTCS '01), Lecture Notes in Computer Science, vol. 2202, pp. 357–369, 2001.
7. P. D'Arco and D. R. Stinson, *On Unconditionally Secure Proactive Verifiable Secret Sharing Schemes and Distributed Key Distribution Centers*, unpublished manuscript, May 2002.
8. R. Gennaro, Y. Ishai, E. Kushilevitz, and T. Rabin, *The Round Complexity of Verifiable Secret Sharing and Secure Multicast*, Proc. of the 33-rd Annual Symposium on the Theory of Computing (STOC '01), ACM, pp. 580–589, 2001.
9. A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. *Proactive Secret Sharing or: How to Cope with Perpetual Leakage*, Advances in Cryptology - Crypto '95, Lecture Notes in Computer Science, vol. 963, pp. 339–352, 1995.
10. F. J. MacWilliams and N. J. A. Sloane, **The Theory of Error-Correcting Codes**, North-Holland, Amsterdam, 1981.
11. A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone, **Handbook of Applied Cryptography**, CRC Press, 1996.
12. M. Naor, B. Pinkas, and O. Reingold. *Distributed Pseudo-random Functions and KDCs*, Advances in Cryptology - Eurocrypt'99, Lecture Notes in Computer Science, vol. 1592, pp. 327–346, 1999.
13. R. M. Needham and M. D. Schroeder. *Using Encryption for Authentication in Large Networks of Computers*, Communications of ACM, vol. 21, pp. 993–999, 1978.
14. B. C. Neuman and T. Tso. *Kerberos: An Authentication Service for Computer Networks*, IEEE Transactions on Communications, vol. 32, pp. 33–38, 1994.
15. V. Nikov, S. Nikova, B. Preneel and J. Vandewalle. *Applying General Access Structure to Proactive Secret Sharing Schemes.* Proc. of the 23rd Symposium on Information Theory in the Benelux, May 29-31, 2002, Université Catholique de Louvain (UCL), Louvain-la-Neuve, Belgium.
16. V. Nikov, S. Nikova, B. Preneel and J. Vandewalle. *On Distributed Key Distribution Centers and Unconditionally Secure Proactive Verifiable Secret Sharing Schemes Based on General Access Structures*, preprint, August 2002.
17. R. Ostrovsky and M. Yung, *How to Withstand Mobile Virus Attacks*, Symposium on Principles of Distributed Computing (PODC '91), ACM, pp. 51–59, 1991.

18. R. S. Rees, D. R. Stinson, R. Wei, and G. H. J. van Rees, *An Application of Covering Designs: Determining the Maximum Consistent Set of Shares in a Threshold Scheme*, Ars Combinatoria **53**, 225–237, 1999.
19. D.R. Stinson, **Cryptography: Theory and Practice**, CRC Press, 1995 (2nd Edition, 2002).
20. D. R. Stinson. *On Some Methods for Unconditional Secure Key Distribution and Broadcast Encryption*, Designs, Codes and Cryptography, vol. 12, pp. 215–243, 1997.
21. D. Stinson and R. Wei, *Unconditionally Secure Proactive Secret Sharing Scheme with Combinatorial Structures*, SAC'99. Lecture Notes in Computer Science, vol. 1758, pp. 200–214, 1999.

A A $(k, n, \mathcal{C}, \mathcal{G})$-DKDS

In a $(k, n, \mathcal{C}, \mathcal{G})$-DKDS, each user can compute a common key by interacting with any k-subset of the n servers at his choice. In [12], a construction based on bivariate polynomials for a $(k, n, \mathcal{C}, \mathcal{G})$-DKDS was proposed. Basically, it works as follows: Each of the servers $S_1, \ldots, S_k$, performing the initialization phase, constructs a random bivariate polynomial $P^i(x, y)$ of degree $k-1$ in x, and $\ell-1$ in y, and sends $Q^i_j(y) = P^i(j, y)$ to the server S_j, for $j = 1, \ldots, n$. Server S_j computes his private information, $Q_j(y)$, by adding the k polynomials received from $S_1, \ldots, S_k$. A user who wants to compute a conference key, κ_h, sends to (at least) k servers a key request. Each server S_j, invoked by the user, checks that the user belongs to C_h, and sends to the user the value $Q_j(h)$. Using the k values received from the servers, and applying the Lagrange formula for polynomial interpolation, each user in C_h recovers the secret key $P(0, h) = \sum_{i=1}^{k} P^i(0, h)$ (see [12] for details).

The construction is correct and secure, according to the model considered in [12]. In order to introduce verifiability and proactivity, the following approach was suggested in [12]. Time is divided in periods. At the beginning of period t, for $i = 1, \ldots, k$, each server S_i performing the initialization, chooses a random polynomial $P^t_i(x, y)$ of degree $k-1$ in x and $\ell-1$ in y such that $P^t_i(0, h) = 0$ for each $h \in Z_q$. Then, for $i = 1, \ldots, k$, server S_i sends, for $j = 1, \ldots, n$, the univariate polynomial $Q^t_{i,j}(y) = P^t_i(j, y)$ to server S_j, and broadcasts the univariate polynomial $P^t_i(x, c)$, where c is a public point. Then, for $j = 1, \ldots, n$, server S_j checks that $P^t_i(x, c)$ evaluated in $x = 0$ is zero (i.e., $P^t_i(0, c) = 0$) and that the broadcasted polynomial is consistent with $Q^t_{i,j}(y)$ (i.e., $Q^t_{i,j}(c) = P^t_i(j, c)$). Finally, if the check is satisfied, S_j updates his private information by computing $Q_j(y) \leftarrow Q_j(y) + \sum_{i=1}^{k} Q^t_{i,j}(y)$. Unfortunately, a server sending information during the update phase can cheat, as shown by the following example.

Example. Let us consider a $(3, 3, \mathcal{C}, \mathcal{G})$-DKDS. The polynomial $P^t_i(x, y)$ chosen by S_i at the beginning of the period in order to update the system is of degree 2 in x and $\ell-1$ in y. A cheating S_i can choose $P^t_i(x, y) = a + b_1 x + b_2 x^2 + P_Y(y)$ where $a = -P_Y(c)$ and $P_Y(y) = \sum_{i=1}^{\ell-1} p_j y^j$. It is not difficult to check that $P^t_i(0, c) = 0$ and that $P^t_i(x, c) = b_1 x + b_2 x^2$ is equal to $Q^t_{i,j}(y)$ when the first one is evaluated in j and the second one in c. But $P^t_i(0, c') \neq 0$ for any $c' \neq c$.