This is the Full Version of the Extended Abstract that appears
in the Proceedings of 8th Conference on Security and Cryptography for Networks (SCN 2012)
(5 – 7 september 2012, Amalfi, Italy)
I. Visconti and R. De Prisco Eds., Springer-Verlag, LNCS 7485, pages 166–183.

Decentralized Dynamic Broadcast Encryption

Duong Hieu Phan[1,2], David Pointcheval[2], and Mario Strefler[2]

[1] LAGA, University of Paris 8
[2] ENS / CNRS / INRIA
{phan,pointche,strefler}@di.ens.fr

Abstract A broadcast encryption system generally involves three kinds of entities: the group manager that deals with the membership, the encryptor that encrypts the data to the registered users according to a specific policy (the target set), and the users that decrypt the data if they are authorized by the policy. Public-key broadcast encryption can be seen as removing this special role of encryptor, by allowing anybody to send encrypted data. In this paper, we go a step further in the decentralization process, by removing the group manager: the initial setup of the group, as well as the addition of further members to the system, do not require any central authority. Our construction makes black-box use of well-known primitives and can be considered as an extension to the subset-cover framework. It allows for efficient concrete instantiations, with parameter sizes that match those of the subset-cover constructions, while at the same time achieving the highest security level in the standard model under the DDH assumption.

1 Introduction

Broadcast encryption (BE), introduced by Fiat and Naor [FN94] in 1993, allows a sender to securely send private messages to a subset of users, the target set. In 2001, Naor, Naor, and Lotspiech (NNL [NNL01]) introduced the subset-cover framework, where for any target set, the sender can find a partition of the user set, encrypt a session key using the keys associated to each subset in the partition, and finally encrypt the content using the session key. The ciphertext length of the subset-difference (SD) version of NNL depends linearly on the number of users in the revoked set, which was considered to be efficient enough for use in the AACS DRM standard [AAC09]. We generalize the subset-cover framework of NNL to deal with both public-key encryption and dynamic changes of the registered user sets. We furthermore remove the need for trusted authorities by eliminating the group manager, who typically interacts with users to distribute keys at the setup phase or when users join the system. Our approach makes use of group key exchange with subgroup keys [Man09, ACMP10], a primitive that simultaneously distributes different keys to certain subsets of the user group and applies well to the subset-cover framework if one can assign keys for the subgroups involved in the subset cover.

We first instantiate our construction with the Diffie-Hellman key agreement for the key generation and the ElGamal encryption for the public-key encryption, which leads to quite an efficient scheme. The complete-subtree (CS) tree construction resembles the tree-based group key agreement in [KPT04], with the exception that we also create key pairs for internal nodes, and we go beyond their scheme in our construction of SD trees. We then show how our scheme can be extended to achieve the strongest security notion by using Cramer-Shoup encryption, which allows adaptive corruptions and chosen-ciphertext attacks, in the standard model, under the DDH assumption. In addition, we consider various criteria of efficiency: ciphertext size, private part and public part of the decryption keys, number of rounds for the key generation, etc. Thanks to the modularity of our approach, we can use any appropriate group key exchange with subgroup keys: our initial technique iteratively uses the two-party Diffie-Hellman key exchange in a binary tree, which requires a logarithmic number of rounds; we can replace it by logarithmically many parallel executions of the Burmester-Desmedt group key exchange protocol [BD05], which reduces the number of rounds to two. Besides allowing members to join the system, we also sketch how groups could merge at low cost, and how to permanently revoke some users, but we cannot elaborate on this due to space constraints. Our scheme thus achieves a maximum of functionality and security under minimal assumptions, while still being reasonably efficient.

Related Work. Dodis and Fazio [DF03a] already constructed a public-key version of the subset-cover framework using IBE for the Complete-Subtree (CS) structure and HIBE of depth $\log N$ for the Subset-Difference (SD) structure. They retain the same efficiency, using (H)IBE keys instead of symmetric keys, and achieve generalized CCA security. In the same year, Dodis and Fazio presented a dynamic, IND-CCA-secure BE scheme [DF03b], where the adversary may corrupt users before the challenge phase. IND-CPA-security under adaptive corruption was first achieved by Boneh and Waters [BW06], who presented a fully-collusion resistant trace-and-revoke scheme. More recently, Gentry and Waters [GW09] described another adaptively IND-CPA-secure scheme. For both schemes, there is no obvious way to make them IND-CCA-secure in the standard model.

Delerablée [Del08] constructed selectively IND-CPA-secure ID-based BE, which allows adding users after the setup. The only existing dynamic BE scheme was developed by Delerablée, Paillier, and Pointcheval [DPP07]. However, their scheme does not provide forward-secrecy, i. e. a new user can decrypt all ciphertexts sent before he joined. Because our scheme provides forward-secrecy, we have to relax their definition of "dynamic". Forward-security has been considered by Yao, Fazio, Dodis, and Lysyanskaya [YFDL04], first for HIBE and then by extension for BE. Their notion of forward-security refers to security of ciphertexts against later corruption of users, which means that user keys must evolve so that previously sent messages remain secure. This is distinct from our notion of forward-secrecy, where we only require that newly joined users cannot decrypt previously sent ciphertexts. However, when a user gets corrupted, messages this user received prior to corruption can be read by the adversary, since the adversary gets the same power as the user. The scheme in [YFDL04] is IND-CCA-secure, but the adversary is more restricted in corrupting users after the challenge phase than in our setting.

Broadcast encryption without a central authority replaces the traditional setup with a group key exchange process that can be an interactive protocol. It was proposed under the name "contributory broadcast encryption" (CBE) in [WQZ+11], along with a semi-adaptively IND-CPA-secure scheme that is not dynamic. A possible application of this could be communication in a social network, where some private information is meant to be read only be a subset of a user's acquaintances, and the network is either peer-to-peer or the service provider is not trusted. The first steps toward subgroup key exchange were done by Manulis [Man09], who extended a group key exchange (GKE) protocol to allow any two users to compute a common key after the initial phase in which the group key is computed. Following this work, Abdalla et al. [ACMP10] generalized this approach to allow the computation of session keys for arbitrary subsets. We use such a group key exchange protocol with subgroup keys to derive asymmetric encryption keys for subsets. Something similar has been done under the name of "asymmetric group key agreement" (ASGKA) [WMS+09]. In [WMS+09], ASGKA is defined in a way that guarantees only that the keys held by the participants are good for use with a specific encryption scheme. We want to generalize this requirement so that at the end of the protocol run, each user has some randomness, which can thereafter be used for any key generation, and namely to generate key pairs for any key encapsulation mechanism. Since this randomness is shared between various subgroups, we call the scheme we use for the setup "subgroup key exchange" (SKE). Kurnio, Safavi-Naini, and Wang [KSNW03] explicitly consider sponsorship of group candidates by existing members. In our scheme, because of the tree structure, each user can act as a sponsor, and only one sponsor is required for a candidate to join the user set.

Contributions and Organization. In section 2, we define decentralized dynamic broadcast encryption and subgroup key exchange, a building block we use in our construction that may be of independent interest. We extend the security notions of adaptive IND-CPA and IND-CCA from [PPS11] to our case. We describe a black-box construction of decentralized dynamic broadcast encryption using the subset-cover framework in section 3 and prove the security of the construction, assuming that the building blocks are secure. In section 4, we construct a subgroup key exchange protocol based on any secure two-party key exchange protocol. We give two concrete instantiations using our methodology in section 5, that provide keys for subgroups in the

CS and SD structures. Combined with the Cramer-Shoup encryption scheme, this gives us a decentralized dynamic broadcast encryption schemes which additionally achieves the highest security level (fully adaptive IND-CCA-security) in the standard model under the DDH-assumption.

2 Definitions

In the following, we describe some generic constructions for broadcast encryption that make use of standard definitions of well-known primitives. We briefly review the notations here, but provide full definitions in the Appendix A.

A public-key encryption scheme is defined by a tuple of four algorithms $\mathcal{PKE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$. A two-party key exchange protocol is a tuple of two algorithms/protocols $\mathcal{KE} = (\mathsf{Setup}, \mathsf{CommonKey})$. Note that CommonKey is an interactive protocol, but we expect it to be one-round only for the efficiency of our constructions. A message authentication code is a tuple of three algorithms $\mathcal{MAC} = (\mathsf{KeyGen}, \mathsf{GenMac}, \mathsf{VerifMac})$. A pseudo-random generator is a function $\mathcal{F} : X \to Y$ with $|X| \leq |Y|$.

2.1 Decentralized Broadcast Encryption

Let us start with the main protocol we want to build: a broadcast encryption scheme, which aims at encrypting a message for a group of users, with a fine-grained selection of the target group. As in [FN94], broadcast encryption generally involves a group manager, that deals with the membership of the users, and an encryptor that specifies the target group (a subgroup of the registered members) for a ciphertext. The encryptor is either a specific person in case of secret-key broadcast encryption, or anybody in case of public-key broadcast encryption. The group manager is either involved once only, at the setup phase, in static schemes, or at any time a new member wants to join the system, in dynamic schemes [DPP07]. The latter dynamic situation is the most realistic, but makes the group manager quite sensitive, for both security and availability. Our goal is to get rid of such a centralized system.

We thus extend the dynamic broadcast encryption setting [DPP07] so that the membership management can be decentralized. At the same time, we would like to keep everything as small as possible.

1. The ciphertext size should be as small as possible: the ciphertext has to contain the target group structure, and so cannot be smaller than the representation of this structure, which can either be encoded on N bits, where N is the total number of users, and each bit tells whether a user is in the target group or not, or on $r \log N$ bits (resp. $s \log N$ bits), where r (resp. s) is the number of revoked users (resp. included users) among the N registered users. This is sometimes considered independently from the ciphertext, in the header, but anyway both the target set and the encrypted data have to be sent. Our goal is to make the global length as small as possible.
2. When a new user joins the system, it should have minimal impact on other users' secret information and the public information: no impact at all on the keys as in [DPP07] is of course optimal, but when one wants to achieve forward secrecy, it is not possible: some of the keys have to be modified. We will try to keep the impact as small as possible too.

Since we want to avoid any centralized group manager, we will also focus on public-key broadcast encryption, in which a public key is enough to target any subgroup at the encryption time. In addition, instead of encrypting a message, our schemes will generate an encapsulation (or key header) and session keys to be used with any symmetric encryption scheme [Sho00].

Definition 1 (Decentralized Dynamic Broadcast Encapsulation). A decentralized dynamic broadcast encapsulation scheme is a tuple of five algorithms or protocols $\mathcal{DBE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Join}, \mathsf{Encaps}, \mathsf{Decaps})$:

– $\mathsf{Setup}(1^k)$, where k is the security parameter, generates the global parameters param of the system.

$\mathrm{Exp}^{\mathsf{ind-acca}-b}_{\mathcal{DBE},\mathcal{A}}(k)$
$\mathcal{Q}_C \leftarrow \emptyset$; $\mathcal{Q}_D \leftarrow \emptyset$;
$\mathsf{param} \leftarrow \mathsf{Setup}(1^k)$;
$(state, U) \leftarrow \mathcal{A}(\mathsf{SETUP}; \mathsf{param})$;
$(\mathsf{EK}, Reg, \tau) \leftarrow \mathsf{OExecute}(U)$;
$(state, S) \leftarrow \mathcal{A}^{\mathsf{OJoin}(\cdot),\mathsf{OCorrupt}(\cdot),\mathsf{ODecaps}(\cdot,\cdot,\cdot)}(state; \mathsf{EK}, Reg, \tau)$;
$(H, K) \leftarrow \mathsf{Encaps}(\mathsf{EK}, Reg, S)$;
$K_b \leftarrow K$; $K_{1-b} \xleftarrow{\$} \mathcal{K}$;
$b' \leftarrow \mathcal{A}^{\mathsf{OJoin}(\cdot),\mathsf{OCorrupt}(\cdot),\mathsf{ODecaps}(\cdot,\cdot,\cdot)}(state; H, K_0, K_1)$;
`if` $\exists i \in S, (i, S, H) \in \mathcal{Q}_D$ `or` $i \in \mathcal{Q}_C$;
`then return` 0
`else return` b';

$\mathsf{OExecute}(U)$
$(\mathsf{EK}, Reg) \leftarrow \mathsf{KeyGen}(\mathsf{param}, U)$;
`return` EK, Reg, τ;

$\mathsf{OJoin}(v)$
$(\mathsf{EK}, Reg) \leftarrow \mathsf{Join}(v, U, Reg, \mathsf{EK})$;
`return` EK, Reg, τ;

$\mathsf{OCorrupt}(u)$
$\mathcal{Q}_C \leftarrow \mathcal{Q}_C \cup \{u\}$; `return` dk_u;

$\mathsf{ODecaps}(u, S, H)$
$\mathcal{Q}_D \leftarrow \mathcal{Q}_D \cup \{(u, S, H)\}$;
$K \leftarrow \mathsf{Decaps}(\mathsf{dk}_u, S, H)$;
`return` K;

Figure 1: $\mathcal{DBE}$: Key Indistinguishability (IND-ACCA)

- $\mathsf{KeyGen}(\mathsf{param}, U)$ is an interactive protocol between the users in the set U. After the protocol run, it returns the public encryption key EK and a list Reg of the registered users with additional public information. Each user $u \in U$ eventually gets a secret decryption key dk_u.
- $\mathsf{Join}(v, \{u(\mathsf{dk}_u)\}_{u \in U}, Reg, \mathsf{EK})$ is an interactive protocol run between a user v and the set of users U, described in Reg. Each user takes as input his secret key and/or some random coins, the list Reg, and the encryption key EK. After the protocol, Reg and EK are updated, and each user (including v) has a secret decryption key.
- $\mathsf{Encaps}(\mathsf{EK}, Reg, S)$ takes as input the encryption key EK, the user register Reg, and a target set S. It outputs a key header H and a session key $K \in \{0,1\}^k$.
- $\mathsf{Decaps}(\mathsf{dk}_u, S, H)$ takes as input the target set S and a user decryption key dk_u together with a key header H. If dk_u corresponds to a recipient user, it outputs the session key K, else it outputs the error symbol $\perp$.

The correctness requirement is that for all N, any target set $S \subset U_N = [1, N]$ and for any $u \in U_N$, if $u \in S$ then the decapsulation algorithm gives back the key. A decentralized scheme requires that no authority is involved in the KeyGen and Join protocols.

Security Notions A general overview of the security notions for broadcast encryption has been done in [PPS11]. We extend the strongest one to the decentralized setting. The adversary is still given unlimited access to the Join oracle (dynamic), the Corrupt oracle (adaptive) and Decaps oracle (chosen-ciphertext security). For the group key generation, the definition from [PPS11] models passive adversaries only, since they only receive the public keys. Since in our case this group key generation may be an interactive protocol, we make it more explicit with a Execute-oracle that outputs the public transcript of the full run of this protocol. The security game for DBE is presented in figure 1: the restriction for the adversary is not to ask for the decapsulation of the challenge ciphertext (which includes the target set S) nor corrupt any user in the target set.

The adversary can ask once the generation of the group structure with a single call to OExecute on a group U of its choice, from which it gets the transcript τ, the encryption key EK and the register Reg. It can thereafter make as many calls it wants to OJoin, to add a user to the structure Reg, which updates EK. The adversary also gets the transcript τ of this interactive protocol. At any time, the adversary can also corrupt a user with a key pair, calling OCorrupt and getting back all the secret information of the user, and decapsulate a ciphertext H, calling ODecaps in the name of a user u.

The main security goal of an encryption scheme (or an encapsulation scheme) is the indistinguishability of a challenge ciphertext: at some point, the adversary thus gets a challenge (H, K_0, K_1), where H encapsulates

either K_0 or K_1 for a target set S chosen by the adversary. It has to guess which key is actually encapsulated. Of course, there are the natural restrictions, which are controlled granted the lists $\mathcal{Q}_C$ and $\mathcal{Q}_D$:

- (S, H) has not been asked to the decapsulation oracle for a user u in S
- none of the users in S have been corrupted

A dynamic broadcast encapsulation scheme is $(t, N, q_C, q_D, \varepsilon)$-IND-ACCA-secure (security against adaptive corruption and chosen-ciphertext attacks) if in the security game presented in figure 1, the advantage $\mathrm{Adv}^{\mathsf{ind-acca}}_{\mathcal{DBE}}(k, t, N, q_C, q_D)$, of any t-time adversary $\mathcal{A}$ creating at most N users (OJoin oracle), corrupting at most q_C of them (OCorrupt oracle), and asking for at most q_D decapsulation queries (ODecaps oracle), is bounded by ε.

$$\mathrm{Adv}^{\mathsf{ind-acca}}_{\mathcal{DBE}}(k, t, N, q_C, q_D) = \max_{\mathcal{A}}\{\Pr[\mathrm{Exp}^{\mathsf{ind-acca}-1}_{\mathcal{DBE},\mathcal{A}}(k) = 1] - \Pr[\mathrm{Exp}^{\mathsf{ind-acca}-0}_{\mathcal{DBE},\mathcal{A}}(k) = 1]\}.$$

This definition includes IND-ACPA (for adaptive chosen-plaintext attacks) when $q_D = 0$.

Remark 2 (Forward-secrecy). This definition includes forward-secrecy against new users, i. e. a new user cannot decrypt ciphertexts that were created before he joined. For a definition without forward secrecy, the adversary is prohibited from corrupting users that joined after the challenge phase.

2.2 Subgroup Key Exchange

The novelty of our definition is the decentralized key generation procedure, that should also generate keys for certain subgroups in order to be able to broadcast to any target set. This is thus in the same vein as the notion of group key exchange with on-demand computation of subgroup keys (GKE+S) from [ACMP10], that allows some subgroups of users to run a protocol to establish keys between them. But we extend this definition by allowing for keys of some subgroups to be computed during the first protocol run that establishes the global key, without any additional interaction.

Since we want to remain independent of the encryption scheme to be used with the session key, we require that for each subgroup a proto-key is computed, whose entropy can be used as input to a PKE key-pair generation, or to generate a symmetric encryption key.

Definition 3 (Dynamic $\mathcal{S}$-Subgroup Key Exchange Protocol). For a collection $\mathcal{S} : \mathbb{N} \to \mathcal{P}(\mathbb{N})$ of subsets of the user set, where for any N, $\mathcal{S}(N) \in \mathcal{P}(N)$, a dynamic $\mathcal{S}$-subgroup key exchange protocol $\mathcal{SKE}$ is a tuple of three algorithms and interactive protocols:

- $\mathsf{Setup}(1^k)$, where k is the security parameter, generates the global parameters param of the system;
- $\mathsf{KeyGen}(\mathsf{param}, U)$ is an interactive protocol run between all users in U. It outputs a register Reg that contains a description of U and the subsets for which keys were established according to $\mathcal{S}$, and for each user $u \in U$ a secret usk_u that contains the proto-keys pt_S for all the sub-groups S containing u.
- $\mathsf{Join}(v, U, Reg)$ is an interactive protocol run between user v and the group of users U. It outputs an updated register Reg and for user v and some of the users in U a new secret usk_u that contains the proto-keys pt_S of all the subgroups S they are part of.

We require that all the users $u \in U$ that run $\mathsf{KeyGen}(\mathsf{param}, U)$ receive the same register Reg and compute matching proto-keys for the subsets they have in common. The same is required of Join.

For the security definition, we extend the definition given in [ACMP10], which seems to be most applicable to our case. Since the protocol is dynamic, the user set can change over time. As in the previous section, we stick to passive adversaries. This is a way of modularizing protocol construction, as passively secure protocols can be made secure against active adversaries using constructions such as [KY07], with additional authentication mechanisms.

$\mathrm{Exp}^{\mathsf{ind}-b}_{\mathcal{SKE},\mathcal{A}}(k)$
$Reg \leftarrow \emptyset$; $\mathcal{Q}_T \leftarrow \emptyset$;
$\mathsf{param} \leftarrow \mathsf{Setup}(1^k)$;
$(state, U) \leftarrow \mathcal{A}(\mathsf{param})$;
$(\mathsf{EK}, Reg, \tau) \leftarrow \mathsf{OExecute}(U)$;
$b' \leftarrow \mathcal{A}^{\mathsf{OJoin}(\cdot),\mathsf{OTest}(\cdot,\cdot)}(state; \mathsf{EK}, Reg, \tau)$;
return b';

$\mathsf{OExecute}(U)$
$t \leftarrow 0$;
$Reg \leftarrow \mathsf{KeyGen}(\mathsf{param}, U)$;
return Reg, τ;

$\mathsf{OTest}(t, S)$
if $\exists (t', K) \wedge t \equiv_S t' \wedge (t', S, K) \in \mathcal{Q}_T$
then return K;
else if $b = 0$ then $K \leftarrow \mathsf{pt}_S(t)$;
else $K \xleftarrow{\$} \mathcal{K}$;
$\mathcal{Q}_T \leftarrow \mathcal{Q}_T \cup \{(t, S, K)\}$;
return K;

$\mathsf{OJoin}(v)$
$t \leftarrow t + 1$;
$Reg \leftarrow \mathsf{Join}(v, U, Reg)$;
return Reg, τ;

Figure 2: $\mathcal{SKE}$: Key Indistinguishability (IND)

The adversary can ask once the generation of the group structure with a unique call to $\mathsf{OExecute}$, at time $t = 0$, on a group U of its choice from which it gets the transcript τ and the register Reg. It can thereafter make as many calls as it wants to OJoin, to add a user to the structure Reg. Each query increases the time index t. The adversary also gets the transcripts τ of these interactive protocols.

The main security goal of key exchange is the indistinguishability of the keys, and their independence. Hence, we use the stronger notion proposed in [AFP05], similar to the Real-or-Random [BDJR97] for encryption. The adversary has access to many $\mathsf{OTest}(t, S)$ queries, that are either answered by the real keys or by truly random and independent keys. Note that according to the protocol, some keys may remain unchanged even when the time period evolves. We even hope to have as many keys as possible that do not evolve, since we want that not too many users are impacted by a new member in the system. We thus say that two pairs (t_1, S) and (t_2, S) are equivalent (denoted by $t_1 \equiv_S t_2$) if S is unchanged between the time periods and therefore they should have the same key. For such equivalent pairs, the same random key is output. We do not provide direct access to a $\mathsf{OReveal}$ oracle, which returns the secret key of a user, because as explained in [AFP05], having access to many OTest queries annihilates the advantage provided by $\mathsf{OReveal}$ queries.

A subgroup key exchange scheme is said to be (t, N, q_T, ε)-IND-secure if, in the security game presented in figure 2, the advantage $\mathrm{Adv}^{\mathsf{ind}}_{\mathcal{SKE}}(k, t, N, q_T)$ of any t-time adversary $\mathcal{A}$ creating at most N users (the final size of the set U), testing at most q_T keys is bounded by ε.

$$\mathrm{Adv}^{\mathsf{ind}}_{\mathcal{SKE}}(k, t, N, q_T) = \max_{\mathcal{A}}\{\Pr[\mathrm{Exp}^{\mathsf{ind}-1}_{\mathcal{SKE},\mathcal{A}}(k) = 1] - \Pr[\mathrm{Exp}^{\mathsf{ind}-0}_{\mathcal{SKE},\mathcal{A}}(k) = 1]\}.$$

3 Generic Decentralized Broadcast Encryption

As already remarked, in the first definition of dynamic broadcast encryption schemes [DPP07], it is required that the existing users are not affected by a join: their decryption keys should not be modified. Only the encryption key could be modified. This constraint is actually achieved by their scheme, but this is possible because the scheme is not forward-secure: a new user can decrypt all ciphertexts that were sent before he joined (since he cannot be in any revoked set).

To achieve forward-secrecy, we have to relax their definition and allow updates of the user decryption keys. Namely, updates of the decryption keys are necessary for forward-secrecy in the subset-cover framework [NNL01], because some keys are shared by several users. With an appropriate subset-cover structure, it can reach asymptotically optimal overall ciphertext size. On the other hand, the naive scheme, where each user has a single key specific to him, can be made dynamic without decryption key updates, but has ciphertexts whose length is linear in the number of users. As soon as keys are shared between users, forward-secrecy

makes it necessary to update these shared keys. Hence our relaxation of the model. However, we require these updates of existing keys to be made via public channels.

3.1 Generic Public-Key Subset Cover

A subset-cover structure $\mathcal{SC} = \{S_i\}_{i\in I}$ is a set of subsets S_i of a user set U such that for any subset $S \subset U$ there is a subset $\mathcal{L} \subset I$ such that S can be *partitioned* as $S = \bigcup_{i\in\mathcal{L}} S_i$. In particular, this implies that for all users $u \in U$, $\{u\} \in \mathcal{SC}$. In [NNL01], a secret key is assigned to each set S_i, so a message can be encrypted to any subset $S \subset U$ by finding the cover $\mathcal{L}$ of S. Then a session key is encrypted under all the keys associated to the selected subsets. All the other users are then implicitly revoked, since they cannot decrypt the session key. Because of the partition property, a user in S is in one subset S_i only. Efficiency will thereafter depend on the subset-cover structure.

We extend this framework in three directions:

1. First, we transfer this approach to the public-key world. Each S_i is assigned a key pair of some PKE scheme by some key assignment procedure. This means that the assignment of keys to the subsets depends on the PKE scheme used as well as the assignment procedure. For example, for a subset-cover structure $\mathcal{SC}$ and a PKE $\mathcal{PKE}$, we can use the key assignment that assigns each subset with a key pair drawn independently at random by the trusted center.
2. Second, we replace the trusted center by an interactive protocol, a subgroup key exchange.
3. Third, we allow for the addition of users, hence using a *dynamic subgroup key exchange* to generate the keys for a dynamic subset-cover structure.

We first deal with a dynamic subset-cover structure, assuming a subgroup key exchange as a black box. Thereafter, we will consider concrete structures and efficient subgroup key exchanges.

3.2 Dynamic Subset-Cover

We define a dynamic subset-cover as a sequence of subset-covers $\{\mathcal{SC}_i\}$ for $i \geq 0$ users, where each $\mathcal{SC}_i$ contains subsets S_j. These subsets never change, so instead of adding a user to a subset, we remove the old one and add a new one. This also means that the same subset S_j can occur in different time periods (the time period changes each time a new user joins). We start with $\mathcal{SC}_0 = \emptyset$ and an empty user set $U_0 = \emptyset$, and then have $U_{n+1} = U_n \cup \{u_{n+1}\}$. From the definition, it is clear that $|U_n| = n$, and w.l.o.g. $U_n = [1, n]$.

For subset-cover based dynamic broadcast encryption, we will have to generate the keys for all the subsets that are involved in $\mathcal{SC}_n$. The following property will optimize efficiency, in the sense that a minimal number of existing users will be impacted by a new member.

Definition 4 (Splitting Property). We say that a dynamic subset cover $\mathcal{SC}$ has the *splitting property*, if the subset cover at time $n+1$ is composed of subsets that either were part of the subset cover at time n, or contain the new user. $\mathcal{SC}_{n+1} = \mathcal{SC}'_{n+1} \cup \mathcal{SC}''_{n+1}$, where $\mathcal{SC}'_{n+1} \subset \mathcal{SC}_n$ and $S_i \in \mathcal{SC}''_{n+1} \Rightarrow u_{n+1} \in S_i$.

With this property, if a subset changes, it is either removed, or it contains u_{n+1}. Then only sets with the new user need new key generation, which is a minimal requirement anyway.

3.3 SC-based Decentralized Dynamic Broadcast Encryption

We first assume we have a dynamic subgroup key exchange $\mathcal{SKE}$ that is compatible with our dynamic subset-cover structure. It means that for any n, the subgroup key exchange provides keys for all the subsets S in $\mathcal{SC}_n$. We will later instantiate such a dynamic subgroup key exchange for some dynamic subset-cover structures.

Let us recall that the SC-based broadcast encryption [NNL01] consists in encrypting the same message under the keys of all the subsets that cover the target set. Since one of our goals is to achieve the highest

security level, adaptive chosen-ciphertext security, any modification of the description of the target set or one of the ciphertexts in the list should make the global ciphertext invalid, otherwise the scheme is somewhat malleable, and thus insecure against chosen-ciphertext attacks. We will add a MAC to bind the target header and the ciphertexts together. A similar approach has been used by [BK05, DK05]. Instead of a master secret key, our scheme needs only a public register *Reg* to keep track of the users currently enrolled in the system and their public keys.

We first present in details our construction, and then state the security of the construction. It is important to remember that the subgroup key exchange scheme is only assumed to be passively secure, meaning that the protocol requires authenticated channels. This can be achieved in several ways that we will not discuss here. Because the subset cover is a fixed part of the protocol and defines the subsets for each number of users, and we assume that the number of users in the system is always known, the number of a new user and the subsets he belongs to can be computed deterministically by all users. Meta-issues like trust between users and how they should agree on which users to allow into the group are beyond the scope of this paper.

Definition 5 (dBE). Let $\mathcal{PKE}$ be a PKE, $\mathcal{MAC}$ a MAC, $\mathcal{F} : \mathcal{K} \to \mathcal{R}$ a pseudo-random generator, $\mathcal{SC}$ a dynamic subset-cover, and $\mathcal{SKE}$ a dynamic subgroup key exchange compatible with $\mathcal{SC}$ with keys in $\mathcal{K}$. Our Broadcast Encryption Scheme is defined as follows.

- $\mathsf{Setup}(1^k)$:
  1. Run $\mathcal{PKE}.\mathsf{Setup}(1^k)$ to get $\mathsf{param}_{\mathcal{PKE}}$;
  2. Run $\mathcal{SKE}.\mathsf{Setup}(1^k)$ to get $\mathsf{param}_{\mathcal{SKE}}$;
  3. Publish $\mathsf{param} = (\mathsf{param}_{\mathcal{PKE}}, \mathsf{param}_{\mathcal{SKE}})$.
- $\mathsf{KeyGen}(\mathsf{param}, U_n)$, for some integer n:
  1. Run $\mathcal{SKE}.\mathsf{KeyGen}(\mathsf{param}_{\mathcal{SKE}}, U_n)$ to get *Reg*; Each user $u \in U_n$ gets as output of the protocol the proto-keys pt_S for all subsets S he belongs to according to $\mathcal{SC}$. The decryption key dk_u consists of all these pt_S.
  2. He computes $(\mathsf{dk}_S, \mathsf{ek}_S) \leftarrow \mathcal{PKE}.\mathsf{KeyGen}(\mathsf{param}_{\mathcal{PKE}}; \mathcal{F}(\mathsf{pt}_S))$, where we use the PRG to generate from the proto-key the random coins of the key generation algorithm;
  3. All the encryption keys ek_S are published as EK;
  4. The decryption keys dk_S can be either stored in dk_u for users $u \in S$, or deleted since they can be recomputed;
- $\mathsf{Join}(v, \{u(\mathsf{dk}_u)\}_{u \in U_n}, Reg, \mathsf{EK})$:
  1. Run $\mathcal{SKE}.\mathsf{Join}(v, \{u(\mathsf{dk}_u)\}_{u \in U_n}, Reg)$ to get the new *Reg*;
  2. Each user u does as above to compute dk_S, ek_S and dk_u. Note that granted the splitting properties, only dk_S, and thus ek_S, for S that contain v are affected;
- $\mathsf{Encaps}(\mathsf{EK}, Reg, S)$:
  1. From the target set S, generate the partition $\mathcal{L}$ with $S = \cup_{\mathcal{L}} S_i$;
  2. Generate a session key $\mathcal{K}_e$ and a MAC key $\mathcal{K}_m$;
  3. For each subset $i \in \mathcal{L}$, generate $c_i = \mathcal{PKE}.\mathsf{Encrypt}(\mathsf{ek}_{S_i}, \mathcal{K}_e || \mathcal{K}_m)$;
  4. Compute $\sigma = \mathcal{MAC}.\mathsf{GenMac}(\mathcal{K}_m, S || (c_i)_{i \in \mathcal{L}})$;
  5. Output $\mathcal{K}_e$ and $H = ((c_i)_{i \in \mathcal{L}}, \sigma)$.
- $\mathsf{Decaps}(\mathsf{dk}_u, S, H)$:
  1. If $u \in S$, then there is a unique i such that $u \in S_i$, and then dk_u allows to derive $\mathsf{dk} = \mathsf{dk}_{S_i}$;
  2. Extract $\mathcal{K}_e || \mathcal{K}_m = \mathcal{PKE}.\mathsf{Decrypt}(\mathsf{dk}, c_i)$;
  3. Check if σ is a valid MAC under key $\mathcal{K}_m$;
  4. In case of validity, output $\mathcal{K}_e$, otherwise output $\perp$.

The scheme is a correct dynamic broadcast encryption scheme, because of the correctness of the basic primitives $\mathcal{PKE}$, $\mathcal{MAC}$ and $\mathcal{F}$, but also $\mathcal{SKE}$.

Theorem 6. *Let us consider the scheme $\mathcal{BE}^{\mathcal{PKE},\mathcal{MAC},\mathcal{F},\mathcal{SKE}}$ from definition 5. We define L_N to be the total number of distinct subsets over all time periods and ℓ_N to be the maximal number of subsets necessary to cover any authorized target set S in $\mathcal{SC}_i$ for any i. If $\mathcal{PKE}$ is an IND-CCA-secure PKE, $\mathcal{MAC}$ is a SUF-CMA-secure MAC, $\mathcal{SKE}$ is a IND-secure SKE, and $\mathcal{F}$ is a pseudo-random generator, then this scheme is a forward-secure IND-ACCA-secure BE scheme:*

$$\begin{aligned}\mathrm{Adv}^{\mathsf{ind-acca}}_{\mathcal{DBE}}(k,t,N,q_C,q_D) \leq\ & 2\mathrm{Adv}^{\mathsf{ind}}_{\mathcal{SKE}}(k,t,L_N,L_N) + 3\ell_N L_N \mathrm{Adv}^{\mathsf{ind-cca}}_{\mathcal{PKE}}(k,t,q_D) \\ & + 2L_N \mathrm{Adv}^{\mathsf{prg}}_{\mathcal{F}}(k,t) + 2\mathsf{Succ}^{\mathsf{suf-cma}}_{\mathcal{MAC}}(k,t,1,q_D).\end{aligned}$$

The variables L_N and ℓ_N depend on the type of subset cover used in the scheme. For CS, L_N is less than $N \log N$ (since at most $\log N$ sets change in each of the at most N steps), and ℓ_N is $r \log \frac{N}{r}$, which is bounded by $N/2$ (the worst-case ciphertext length). For SD, we have $L_N \leq N \log^2 N$ and $\ell_N = 2r - 1$. The complete security proof can be found in the Appendix B.

4 Tree-based Subgroup Key Exchange

In this section, we define two subgroup key exchange protocols compatible with the efficient tree-based methods defined in [NNL01]. The tree-based methods are special cases in the subset-cover framework, where the users are organized as leaves in a binary tree, and the subsets S_i can be described in terms of subtrees of this tree.

Complete Subtree. We first review the static complete subtree (CS) structure for N users $\{u_0, \ldots, u_{N-1}\}$. For simplicity, we assume $N = 2^d$, but the description can be generalized to any N. All the users are leaves of the tree, and can be seen as singletons $S_{2^d+i} = \{u_i\}$, for $i = 0, \ldots, 2^d - 1$. Then, for $i = 2^d - 1$ to 1, $S_i = S_{2i} \cup S_{2i+1}$ which contains all the leaves below the node with index i.

Subset Difference. The subset difference (SD) method uses subsets $S_{i,j} = S_i \setminus S_j$, where S_i, S_j are defined as in the CS method, and S_j is a subtree of S_i. All sets S_i from the CS tree are also contained in the SD method, because $S_i = S_{\mathrm{parent}(i),\mathrm{sibling}(i)}$; S_0 is included as a special set.

4.1 Static Tree Construction

Let us show how such subset-cover structures naturally give rise to subgroup key exchange protocols. The main tools for our construction of the subgroup key exchange are two primitives: a 2-party key exchange protocol $\mathcal{KE}$ that outputs keys in $\mathcal{K}_{\mathcal{KE}}$ and a pseudo-random generator $\mathcal{G} : \mathcal{K}_{\mathcal{KE}} \to \mathcal{K} \times \mathcal{R}_{\mathcal{KE}}$.

Two users start from random coins in $\mathcal{R}_{\mathcal{KE}}$, and run a key exchange protocol $\mathcal{KE}.\mathsf{CommonKey}$ in order to derive a secret value ck for the subset represented by the node in the tree that is their parent. This common key ck is used as the seed for the PRG $\mathcal{G}$ to derive the two secret keys, the proto-key $\mathsf{pt} \in \mathcal{K}$ and the random coins $r \in R_{\mathcal{KE}}$ for the next key exchange at the level above. Internal nodes thus involve "virtual" users. In summary, the tree is constructed by executing $\mathcal{KE}.\mathsf{CommonKey}$, then computing $\mathcal{G}$, at each level from the bottom up. We derive generic instantiations of the complete subtree (CS) and subset difference (SD) methods on binary trees described in [NNL01].

CS Tree. We define the neighbour of user u with identifier i to be the user u' with identifier $i + 1$ if $i \equiv 0 \mod 2$, $i - 1$ else and its parent to be the user w with identifier $\lfloor i/2 \rfloor$. At round r, each (virtual) user u created in round $r - 1$ has a uniquely defined neighbour u' and a parent w. If he does not, the protocol run is completed: we are either at the root of the tree, or the tree is not complete. The users u and u' have random coins r_u and $r_{u'}$, which they use to run the $\mathcal{KE}$ protocol, resulting in a common key ck_w. From this common

key, they derive the proto-key of node w and the randomness for the virtual user w to participate in the next round of key exchanges. The user with the smaller identifier then plays the role of the virtual user w in the next round. As a consequence, for N users, there are $\log N$ rounds. Round r involves $N/2^{r-1}$ (virtual) users.

- $\mathsf{KeyGen}(U_n)$: In round r, for $r = 1, \ldots, \log n$, the users u, u' with parent w at level $(\log n - r)$ proceed as follows:
  1. $\mathsf{ck}_w \leftarrow \mathcal{KE}.\mathsf{CommonKey}(u, u')$;
  2. $(\mathsf{pt}_w, r_w) \leftarrow \mathcal{G}(\mathsf{ck}_w)$;
  3. If $u < u'$, set $u \stackrel{\mathsf{def}}{=} w$;

A similar construction is possible for the more efficient SD scheme. Due to lack of space, we present this construction in the Appendix D.

4.2 Dynamic Tree Construction

Dynamic CS. We define a join procedure for the CS tree described above. We go from $\mathcal{SC}_n$ to $\mathcal{SC}_{n+1}$ by taking the leaf u' with the lowest distance to the root, and if there are several with that property, the one with the lowest index. We then replace it with an inner node w, to which we append both the leaf u' and the new user v. We note that the user identifiers will not be in the same order as the node numbers in the tree.Then we replace the subsets S_j where j is an ancestor of the new user with the new subsets. This ensures that our dynamic CS scheme is forward-secure and has the splitting property of definition 4. The CS key assignment is done as follows.

First the new user v derives a common key c_w with its sibling u'. From this common key, he derives the proto-key of node w and the randomness for the virtual user w to participate in the next round of key exchanges. The user with the smaller identifier then plays the role of w in the next round. This procedure is repeated until the keys of all ancestors of v are recomputed.

- $\mathsf{Join}(v, U_n)$ In the first round, set $u \stackrel{\mathsf{def}}{=} v$. In round r, for $r = 1, \ldots, \log(n+1)$, the user u with neighbour u' and parent w at level $(\log(n+1) - r)$ proceeds as follows:
  1. $\mathsf{ck}_w \leftarrow \mathcal{KE}.\mathsf{CommonKey}(u, u')$;
  2. $(\mathsf{pt}_w, r_w) \leftarrow \mathcal{G}(\mathsf{ck}_w)$;
  3. set $u \stackrel{\mathsf{def}}{=} w$, $u' \stackrel{\mathsf{def}}{=} \mathit{neighbour}(w)$, $w \stackrel{\mathsf{def}}{=} \mathit{parent}(w)$;

A similar construction is possible for the more efficient SD scheme. Due to lack of space, we present this construction in the Appendix D. We state exactly the security of the dynamic CS construction. Because of the similarities in the construction, a similar result can be obtained for SD.

Theorem 7. *Let $\mathcal{KE}$ be an IND-secure KE scheme with session keys in $\mathcal{K}_{\mathcal{KE}}$, and $\mathcal{G} : \mathcal{K}_{\mathcal{KE}} \to \mathcal{K} \times \mathcal{R}_{\mathcal{KE}}$ be a PRG. Then our dynamic CS construction of a $\mathcal{SKE}$ is IND-secure and*

$$\mathrm{Adv}^{\mathsf{ind}}_{\mathcal{SKE}}(k, t, N, q_T) \leq (N \log N) \left(\mathrm{Adv}^{\mathsf{ind}}_{\mathcal{KE}}(k, t) + \mathrm{Adv}^{\mathsf{prg}}_{\mathcal{G}}(k, t) \right).$$

The full proof can be found in the Appendix C.

4.3 Efficiency Properties

One of the main advantages of the NNL constructions [NNL01] is the efficient revocation with small ciphertext lengths ($\mathcal{O}(r \log N/r)$ for CS, $\mathcal{O}(2r-1)$ for SD) which is immediately inherited in our public-key scheme. The decryption key is the same length for CS, where each user has to store $\log N$ keys only, and longer ($\mathcal{O}(N \log N)$ for SD), where we cannot use the same key derivation.

In our scheme, for many instantiations of the 2-party key exchange, the private part of the decryption key can even be constant-size: each user keeps his secret random coins r_i, which is enough to iteratively generate all the private information from the public transcript of the key exchange protocols (stored in *Reg* or in the public key). Then, granted the key exchange scheme and $\log N$ public keys, each user can iteratively compute the decryption keys along the path to the root of the tree, and it is in this sense that the user random coins "contain" the keys used to decrypt, as required by the decapsulation algorithm.

Permanent Revocation. Because the length of the ciphertext for SC schemes depends on the number of revoked users, it is desirable to be able to completely remove users from a group. To permanently remove a user at leaf $2i$, we remove it and its sibling leaf $2i+1$ and simply move the user at $2i+1$ to be at node i which becomes a leaf. The keys of the user now at i remain the same as his own key before (at node $2i+1$) and we thus have to update the keys of all subsets in which the revoked user was a member. Concerning the security, it is easy to see that the user $2i$, not having the key of the user $2i+1$, can not learn anything about the updated keys, and this ensures the forward secrecy.

The only problem we face is that we need to keep the tree balanced. Fortunately, our constructions allow a re-organization of the tree in a very efficient manner. Indeed, the tree could be maintained to be an AVL tree at low cost [AVL62]. Whenever a user leaves the system and makes the tree unbalanced, by using $\log N$ rotations, we can re-balance the tree. Note that a rotation needs $\log N$ update operations at worst, so the total cost for a re-balancing is just $\log^2 N$ update operations at worst.

Merging Groups. Instead of joining a single user, we can also efficiently merge two existing groups by executing the key exchange protocol for their root nodes. This will allow every user in the two groups to compute the keys of the new root node.

5 Concrete Instantiations

We now give two instantiations of our scheme. The first one is probably the simplest possible case, and achieves IND-ACPA-security under the DDH-assumption. We use the Diffie-Hellman protocol [DH76] as our KE (where the users publish g^x and g^y from their random coins x and y, and get g^{xy} as common key) and ElGamal [ElG85] as the PKE where $\mathsf{ek} = g^{\mathsf{dk}}$, for a random scalar dk). A similar idea can be found in [KPT04], where the authors use a group key exchange protocol on a DH-tree. Because the random coin spaces of both protocols are identical, when we run both in the same group G of order q (scalars in $\mathbb{Z}_q$), if we only want to prove IND-ACPA-security, we can identify dk with the random coins for the key exchange, and thus ek is part of the transcript of the key exchange protocol, leaving us with a single key pair for both schemes. There are several alternatives for the PRG, the simplest one being a hash function modeled by a random oracle, to extract $\mathsf{dk} \in \mathbb{Z}_q$ from the proto-key $\mathsf{pt} \in G$. But we can avoid it, and even any computational assumption, by using a deterministic randomness extractor, as described in [CFGP06, Th. 7], that is a bijection and thus a perfect generator(see definition 14):

Definition 8. If $p = 2q+1$, and G is defined as the sub-group of the squares in $\mathbb{Z}_p^*$, then $\mathrm{ord}(G) = q$ and f is a bijection from G onto $\mathbb{Z}_q$: $f(x) = x$ ($\mathtt{if}\ \ x \leq q$) or $p - x$ ($\mathtt{if}\ \ x > q$).

The second instantiation is more involved. To achieve IND-ACCA-security, we use Cramer-Shoup encryption [CS98] as our PKE. Because the keys in Cramer-Shoup are larger, our KE is a 3-to-8 parallel Diffie-Hellman, where we use public and private keys consisting of three elements each to generate a shared key consisting of eight elements, which allows us to generate additional pseudo-randomness in each step. Our PRG is an embedding function $G^8 \to \mathbb{Z}_q^3 \times \mathbb{Z}_q^5$ that applies the above function f to all components. The first part in $\mathbb{Z}_q^3$ will be used again as random coins for the key exchange, whereas the second part in $\mathbb{Z}_q^5$ leads to the Cramer-Shoup decryption key. To counter malleability of our scheme, we also need a SUF-CMA-secure MAC scheme. As the first scheme, this one relies only on the DDH assumption.

When using the Cramer-Shoup PKE, the decryption key of node i is the tuple $\mathsf{dk}_i = (v_i, w_i, x_i, y_i, z_i)$, the corresponding encryption key ek_i is $(X_i, Y_i, H_i) = (g^{x_i}h^{v_i}, g^{y_i}h^{w_i}, g^{z_i})$. We need to generate more pseudo-randomness than before, so we define a new key exchange that is essentially a parallel Diffie-Hellman.

Definition 9 (3-8-DHKE). We define a modified Diffie-Hellman key exchange scheme.

- User i draws $a_i, b_i, c_i \xleftarrow{\$} \mathbb{Z}_q$, and sends $(A_i, B_i, C_i) = (g^{a_i}, g^{b_i}, g^{c_i})$;
- User j draws $a_j, b_j, c_j \xleftarrow{\$} \mathbb{Z}_q$, and sends $(A_j, B_j, C_j) = (g^{a_j}, g^{b_j}, g^{c_j})$;
- Then $\mathsf{ck} = (A_i^{a_j}, A_i^{b_j}, A_i^{c_j}, B_i^{a_j}, B_i^{b_j}, B_i^{c_j}, C_i^{a_j}, C_i^{b_j})$.

This easily defines the CommonKey protocol. Its key indistinguishability follows from the following theorem.

Theorem 10 (3-8-DDH). *Under the DDH assumption, it is infeasible to distinguish the 14-tuple* $(g^a, g^b, g^c, g^{a'}, g^{b'}, g^{c'}, g^{aa'}, g^{ab'}, g^{ac'}, g^{ba'}, g^{bb'}, g^{bc'}, g^{ca'}, g^{cb'})$ *from a random 14-tuple even when given* g*, and* $\mathrm{Adv}^{3-8-\mathsf{ddh}}(k,t) \leq 8 \cdot \mathrm{Adv}^{\mathsf{ddh}}(k, t + 11\tau_{exp})$*, where* τ_{exp} *is the time for an exponentiation.*

Proof. We define tuple T_0 to be the tuple as defined above, T_i as the same tuple with all "combined" elements up to the i-th one replaced by a random element. T_8 is therefore a tuple of 14 random elements. Given a distinguisher $\mathcal{A}$ between T_i and T_{i+1}, we construct a solver $\mathcal{B}$ for DDH as follows. Let $(X, Y, Z) = (g^x, g^y, g^z)$ be a DDH challenge tuple. Let $g^{de'}$ be the $i+1$-st combined element. $\mathcal{B}$ chooses a tuple T_i and replaces g^d with X, $g^{e'}$ with Y, and $g^{de'}$ with Y. All other combined elements can be constructed because at least one exponent is known, which takes 11 exponentiations ($11\tau_{exp}$) time. If $z = xy$, $T' = T_i$, else $T' = T_{i+1}$ and the theorem follows.

As a PRG we use the PRG of definition 8 on each component of the common key. This gives us all the components we need to construct an IND-ACCA-secure BE scheme, whose security is based only on the DDH-assumption. (The DDH-assumption implies the existence of OWF, which is sufficient for MACs.)

Constant-Round Key Generation. While this construction achieves constant-size secrets for the users and requires very little interaction during the Join-procedure, it requires a logarithmic number of rounds for the subgroup key exchange protocol to complete. The Burmester-Desmedt group key exchange protocol [BD05] is, like the above scheme, passively secure in the standard model under the DDH assumption [KY07]. It requires only two rounds, and several instances could be run in parallel to compute keys for all subsets in two rounds. This would however require interaction between all the users each time a new users wants to join.

Acknowledgments

This work was supported by the French ANR-09-VERS-016 BEST Project and the European Commission through the ICT Programme under Contract ICT-2007-216676 ECRYPT II.

References

AAC09. AACS Consortium. Advanced Access Content System (AACS) - introduction and common cryptographic elements book. `http://www.aacsla.com/specifications/`, September 2009. Revision 0.951.

ACMP10. Michel Abdalla, Céline Chevalier, Mark Manulis, and David Pointcheval. Flexible group key exchange with on-demand computation of subgroup keys. In Daniel J. Bernstein and Tanja Lange, editors, *AFRICACRYPT 10: 3rd International Conference on Cryptology in Africa*, volume 6055 of *Lecture Notes in Computer Science*, pages 351–368. Springer, May 2010.

AFP05. Michel Abdalla, Pierre-Alain Fouque, and David Pointcheval. Password-based authenticated key exchange in the three-party setting. In Serge Vaudenay, editor, *PKC 2005: 8th International Workshop on Theory and Practice in Public Key Cryptography*, volume 3386 of *Lecture Notes in Computer Science*, pages 65–84. Springer, January 2005.

AVL62. G. Adelson-Velskii and E. M. Landis. An algorithm for the organization of information. *Proceedings of the USSR Academy of Sciences*, 146:263–266, 1962.

BD05. Mike Burmester and Yvo Desmedt. A secure and scalable group key exchange system. *Inf. Proc. Letters*, 94(3):137–143, May 2005.

BDJR97. Mihir Bellare, Anand Desai, Eric Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *38th Annual Symposium on Foundations of Computer Science*, pages 394–403. IEEE Computer Society Press, October 1997.

BK05. Dan Boneh and Jonathan Katz. Improved efficiency for cca-secure cryptosystems built using identity-based encryption. In A. J. Menezes, editor, *CT-RSA*, volume 3376 of *LNCS*, pages 87–103. Springer, 2005.

BW06. Dan Boneh and Brent Waters. A fully collusion resistant broadcast, trace, and revoke system. In *ACM CCS*, pages 211–220. ACM, 2006. Full version available at Cryptology ePrint Archive `http://eprint.iacr.org/2006/298`.

CFGP06. Olivier Chevassut, Pierre-Alain Fouque, Pierrick Gaudry, and David Pointcheval. The twist-augmented technique for key exchange. In M. Yung, editor, *PKC 2006*, volume 3958 of *LNCS*, pages 410–426. Springer, 2006. FV at `http://eprint.iacr.org/2005/061`.

CS98. Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In Hugo Krawczyk, editor, *CRYPTO '98*, volume 1462 of *LNCS*, pages 13–25. Springer, 1998.

Del08. Cécile Delerablée. Identity-based broadcast encryption with constant size ciphertexts and private keys. In K. Kurosawa, editor, *ASIACRYPT 2007*, volume 4833 of *LNCS*, pages 200–215. Springer, 2008.

DF03a. Yevgeniy Dodis and Nelly Fazio. Public key broadcast encryption for stateless receivers. In J. Feigenbaum, editor, *DRM 2003*, volume 2696 of *LNCS*, pages 61–80. Springer, 2003.

DF03b. Yevgeniy Dodis and Nelly Fazio. Public key trace and revoke scheme secure against adaptive chosen ciphertext attack. In Y. G. Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 100–115. Springer, 2003. FV at `http://eprint.iacr.org/2003/095`.

DH76. Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Trans. on Info. Theory*, 22(6):644–654, November 1976.

DK05. Yevgeniy Dodis and Jonathan Katz. Chosen-ciphertext security of multiple encryption. In J. Kilian, editor, *TCC*, volume 3378 of *LNCS*, pages 188–209. Springer, 2005.

DPP07. Cécile Delerablée, Pascal Paillier, and David Pointcheval. Fully collusion secure dynamic broadcast encryption with constant-size ciphertexts or decryption keys. In T. Takagi et al., editor, *Pairing 2007*, volume 4575 of *LNCS*, pages 39–59. Springer, 2007.

ElG85. Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31:469–472, 1985.

FN94. Amos Fiat and Moni Naor. Broadcast encryption. In D. R. Stinson, editor, *CRYPTO '93*, volume 773 of *LNCS*, pages 480–491. Springer, 1994.

GW09. Craig Gentry and Brent Waters. Adaptive security in broadcast encryption systems (with short ciphertexts). In A. Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 171–188. Springer, 2009. FV at `http://eprint.iacr.org/2008/268`.

KPT04. Yongdae Kim, Adrian Perrig, and Gene Tsudik. Tree-based group key agreement. *ACM Trans. on Inf. Systems Security*, 7(1):60–96, May 2004.

KSNW03. Hartono Kurnio, Rei Safavi-Naini, and Huaxiong Wang. A group key distribution scheme with decentralised user join. In *SCN*, volume 2576 of *LNCS*, pages 146–163. Springer, 2003.

KY07. Jonathan Katz and Moti Yung. Scalable protocols for authenticated group key exchange. *Journal of Cryptology*, 20(1):85–113, January 2007.

Man09. Mark Manulis. Group key exchange enabling on-demand derivation of peer-to-peer keys. In *ACNS*, volume 5536 of *LNCS*, pages 1–19. Springer, 2009. FV at `http://www.manulis.eu/pub.html`.

NNL01. Dalit Naor, Moni Naor, and Jeff Lotspiech. Revocation and tracing schemes for stateless receivers. In J. Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 41–62. Springer, 2001. FV at `http://eprint.iacr.org/2001/059`.

PPS11. Duong Hieu Phan, David Pointcheval, and Mario Strefler. Security notions for broadcast encryption. In *Applied Cryptography and Network Security 2011*, volume 6715 of *LNCS*, pages 377–394. Springer, 2011. full version available from the author's webpage.

Sho00. Victor Shoup. Using hash functions as a hedge against chosen ciphertext attack. In B. Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 275–288. Springer, 2000.

WMS+09. Qianhong Wu, Yi Mu, Willy Susilo, Bo Qin, and Josep Domingo-Ferrer. Asymmetric group key agreement. In Antoine Joux, editor, *Eurocrypt*, volume 5479 of *LNCS*, pages 153–170. Springer, 2009.

WQZ+11. Qianhong Wu, Bo Qin, Lei Zhang, Josep Domingo-Ferrer, and Oriol Farras. Bridging broadcast encryption and group key agreement. In D.H. Lee and X. Wang, editors, *Asiacrypt*, volume 7073 of *LNCS*, pages 143–160. Springer, 2011.

YFDL04. Danfeng Yao, Nelly Fazio, Yevgeniy Dodis, and Anna Lysyanskaya. Id-based encryption for complex hierarchies with applications to forward security and broadcast encryption. In *ACM CCS '04*. ACM, 2004. FV at `http://www.cs.brown.edu/~anna/research.html`.

A Definitions

Definition 11 (Encryption Scheme). A public-key encryption scheme is a 4-tuple of algorithms $\mathcal{PKE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$:

- $\mathsf{Setup}(1^k)$, where k is the security parameter, generates the global parameters param of the system;
- $\mathsf{KeyGen}(\mathsf{param}; r)$ generates a pair of keys, the public (encryption) key ek and the associated private (decryption) key dk, using the random coins r (we may omit r when the notation is obvious);
- $\mathsf{Encrypt}(\mathsf{ek}, m; r)$ produces a ciphertext c on the input message m and the public key ek, using the random coins r (we may omit r when the notation is obvious);
- $\mathsf{Decrypt}(\mathsf{dk}, c)$ decrypts the ciphertext c under the private key dk. It outputs the plaintext, or $\perp$ if the ciphertext is invalid.

We require that $\mathsf{Decrypt}(\mathsf{dk}, \mathsf{Encrypt}(\mathsf{ek}, m)) = m$ if $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{KeyGen}(\mathsf{param})$ for some parameters.

Such an encryption scheme is said to be (t, q_D, ε)-IND-CCA-secure (semantic security against chosen-ciphertext attacks) if in the security game presented in figure 3, the advantage, denoted $\mathrm{Adv}^{\mathsf{ind-cca}}_{\mathcal{PKE}}(k, t, q_D)$, of any t-time adversary $\mathcal{A}$ asking at most q_D decryption queries to the $\mathsf{ODecrypt}$ oracle is bounded by ε:

$$\mathrm{Adv}^{\mathsf{ind-cca}}_{\mathcal{PKE}}(k, t, q_D) = \max_{\mathcal{A}}\{\Pr[\mathrm{Exp}^{\mathsf{ind-cca}-1}_{\mathcal{PKE},\mathcal{A}}(k) = 1] - \Pr[\mathrm{Exp}^{\mathsf{ind-cca}-0}_{\mathcal{PKE},\mathcal{A}}(k) = 1]\}.$$

This definition includes IND-CPA (for Chosen-Plaintext Attacks) when $q_D = 0$.

$\mathrm{Exp}^{\mathsf{ind-cca}-b}_{\mathcal{PKE},\mathcal{A}}(k)$
  $\mathsf{param} \leftarrow \mathsf{Setup}(1^k)$;
  $\mathcal{Q}_D \leftarrow \emptyset$, $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{KeyGen}(\mathsf{param})$;
  $(state, m_0, m_1) \leftarrow \mathcal{A}^{\mathsf{ODecrypt}(\cdot)}(\mathsf{FIND}; \mathsf{param}, \mathsf{ek})$;
  $c^* \leftarrow \mathsf{Encrypt}(\mathsf{ek}, m_b)$;
  $b' \leftarrow \mathcal{A}^{\mathsf{ODecrypt}}(\mathsf{GUESS}, state; c^*)$;
  `if` $c^* \in \mathcal{Q}_D$ `then return` 0;
  `else return` b';

$\mathsf{ODecrypt}(c)$
  $\mathcal{Q}_D \leftarrow \mathcal{Q}_D \cup \{c\}$;
  $m \leftarrow \mathsf{Decrypt}(\mathsf{dk}, c)$;
  `return` m;

Figure 3: $\mathcal{PKE}$: Semantic Security against Chosen-Ciphertext Attacks (IND-CCA)

Definition 12 (Two-Party Key Exchange). A two-party key exchange protocol is a 2-tuple $\mathcal{KE} = (\mathsf{Setup}, \mathsf{CommonKey})$:

- $\mathsf{Setup}(1^k)$, where k is the security parameter, generates the global parameters param of the system;
- $\mathsf{CommonKey}(u, v)$ is an interactive protocol between two users u and v. Both take as private input their random coins, and obtain a common key ck.

We require that users u and v that run $\mathsf{CommonKey}(u, v)$ both get the same ck.

For the sake of clarity, we might omit param in the rest of the paper, but global parameters are always implicit for all the primitives. Such a key exchange scheme is said to be (t, ε)-IND-secure (semantic security or key indistinguishability) if in the security game presented in figure 4, the advantage $\mathrm{Adv}^{\mathsf{ind}}_{\mathcal{KE}}(k, t)$ of any t-time adversary $\mathcal{A}$ is bounded by ε, where the adversary gets the transcript τ of the communications between u and v during the execution of $\mathsf{CommonKey}$:

$$\mathrm{Adv}^{\mathsf{ind}}_{\mathcal{KE}}(k, t) = \max_{\mathcal{A}}\{\Pr[\mathrm{Exp}^{\mathsf{ind}-1}_{\mathcal{KE},\mathcal{A}}(k) = 1] - \Pr[\mathrm{Exp}^{\mathsf{ind}-0}_{\mathcal{KE},\mathcal{A}}(k) = 1]\}.$$

$\mathrm{Exp}^{\mathsf{ind}-b}_{\mathcal{KE},\mathcal{A}}(k)$
$\mathsf{param} \leftarrow \mathsf{Setup}(1^k)$;
$(K, \tau) \leftarrow \mathsf{CommonKey}(u, v)$; $K_b \leftarrow K$; $K_{1-b} \xleftarrow{\$} \mathcal{K}$;
$b' \leftarrow \mathcal{A}(\tau, K_0, K_1)$;
`return` b';

Figure 4: $\mathcal{KE}$: Key Indistinguishability (IND)

In our construction, we will need two additional classical primitives: a message authentication code and pseudo-random functions.

Definition 13 (Message Authentication Code). A message authentication code is a 3-tuple of algorithms $\mathcal{MAC} = (\mathsf{KeyGen}, \mathsf{GenMac}, \mathsf{VerifMac})$:

- $\mathsf{KeyGen}(1^k)$, where k is the security parameter, generates a secret key $\mathsf{sk} \xleftarrow{\$} \mathcal{K}_m$.
- $\mathsf{GenMac}(\mathsf{sk}, m)$ takes as input the secret key and a message, and generates the MAC value σ.
- $\mathsf{VerifMac}(\mathsf{sk}, m, \sigma)$ takes as input the secret key, the message and the alleged signature. It checks the validity of the signature and returns 1 if it is valid, 0 else.

In the following, we will require the strong unforgeability of a one-time MAC: even after one MAC generation query, the adversary cannot generate a new valid pair, even for the already authenticated message. This strong unforgeability is formalized in the security game presented in figure 5, where the adversary wins if it successfully verifies a pair that has not been generated by the authentication algorithm. Such

$\mathrm{Exp}^{\mathsf{suf}-\mathsf{cma}}_{\mathcal{MAC},\mathcal{A}}(k)$
$\mathsf{sk} \leftarrow \mathsf{KeyGen}(1^k)$;
$\mathcal{Q}_S \leftarrow \emptyset$; $\mathcal{Q}_V \leftarrow \emptyset$;
$\mathcal{A}^{\mathsf{OGenMac}(\cdot),\mathsf{OVerifMac}(\cdot,\cdot)}(1^k)$;
`if` $\exists (m, \sigma) \in \mathcal{Q}_V$, $(m, \sigma) \notin \mathcal{Q}_S$ `then return` 1;
`else return` 0;

$\mathsf{OGenMac}(m)$
$\sigma \leftarrow \mathsf{GenMac}(\mathsf{sk}, m)$;
$\mathcal{Q}_S \leftarrow \mathcal{Q}_S \cup \{(m, \sigma)\}$
`return` σ;

$\mathsf{OVerifMac}(m, \sigma)$
$c = \mathsf{VerifMac}(\mathsf{sk}, m, \sigma)$;
`if` $c = 1$ `then` $\mathcal{Q}_V \leftarrow \mathcal{Q}_V \cup \{(m, \sigma)\}$;
`return` c;

Figure 5: $\mathcal{MAC}$: Unforgeability (SUF-CMA)

a message authentication code is said to be $(t, q_M, q_V, \varepsilon)$-SUF-CMA-secure (strong existential unforgeability against chosen-message attacks) if in the security game presented in figure 5, the success probability $\mathsf{Succ}^{\mathsf{suf}-\mathsf{cma}}_{\mathcal{MAC}}(k, t, q_M, q_V)$ of any t-time adversary $\mathcal{A}$, asking at most q_M MAC values (OGenMac oracle) and q_V verifications (OVerifMac oracle) is bounded by ε:

$$\mathsf{Succ}^{\mathsf{suf}-\mathsf{cma}}_{\mathcal{MAC}}(k, t, q_M, q_V) = \max_{\mathcal{A}} \{\Pr[\mathrm{Exp}^{\mathsf{suf}-\mathsf{cma}}_{\mathcal{MAC},\mathcal{A}}(k) = 1]\}.$$

This definition includes one-time MAC when $q_M = 1$.

Definition 14 (Pseudo-Random Generator). A generator $\mathcal{F} : X \to Y$ is (t, ε)-pseudo-random if the advantage, denoted $\mathrm{Adv}^{\mathsf{prg}}_{\mathcal{F}}(k, t)$, of any t-time adversary $\mathcal{A}$ is bounded by ε:

$$\mathrm{Adv}^{\mathsf{prg}}_{\mathcal{F}}(k, t) = \max_{\mathcal{A}}\{\Pr[\mathcal{A}(y) = 1 \,|\, y \xleftarrow{\$} Y] - \Pr[\mathcal{A}(\mathcal{F}(x)) = 1 \,|\, x \xleftarrow{\$} X]\}.$$

In the following, Y may be the product of two sets $Y_1 \times Y_2$. We will then parse $\mathcal{F}(x) = (f_1(x), f_2(x))$. If $\mathcal{F}$ is a bijection (which implies that the PRG is not expanding), then $\mathcal{F}$ is a perfect generator, with $\varepsilon = 0$ and no computational assumption.

B Proof of Theorem 6

We assume that $\mathcal{A}$ is an adversary against the IND-ACCA security game. We define a sequence of games, $\mathbf{G}_0$, ..., $\mathbf{G}_9$, where $\mathbf{G}_0$ is the IND-ACCA experiment with $b = 0$ and $\mathbf{G}_9$ is the IND-ACCA experiment with $b = 1$. Let ℓ be the number of components in a challenge ciphertext (the size of the partition $\mathcal{L}^*$ of the challenge target set S^*). By definition, ℓ is not greater than ℓ_N.

Game $\mathbf{G}_0$: This is the IND-ACCA-Experiment with $b = 0$. We just recall the generation of the challenge ciphertext (the Encaps oracle), and the simulation of the ODecaps oracle:

$\mathsf{Setup}(1^k)$:

1. Run $\mathcal{PKE}.\mathsf{Setup}(1^k)$ to get $\mathsf{param}_{\mathcal{PKE}}$;
2. Run $\mathcal{SKE}.\mathsf{Setup}(1^k)$ to get $\mathsf{param}_{\mathcal{SKE}}$;
3. Publish $\mathsf{param} = (\mathsf{param}_{\mathcal{PKE}}, \mathsf{param}_{\mathcal{SKE}})$.

$\mathsf{KeyGen}(\mathsf{param}, U_n)$:

1. All the proto-keys pt_S, for all the subsets S in $\mathcal{SC}_n$, are generated using the $\mathcal{SKE}.\mathsf{KeyGen}$ protocol;
2. Each user $u \in U_n$ gets the proto-keys for all subsets S he belongs to.
   The decryption key dk_u consists of all these pt_S;
3. He computes $(\mathsf{dk}_S, \mathsf{ek}_S) \leftarrow \mathcal{PKE}.\mathsf{KeyGen}(\mathsf{param}_{\mathcal{PKE}}; \mathcal{F}(\mathsf{pt}_S))$, where we use the PRG to generate the random coins of the key generation algorithm;
4. The adversary receives the transcript of the execution of the $\mathcal{SKE}.\mathsf{KeyGen}$ protocol.

$\mathsf{Join}(v, \{u(\mathsf{dk}_u)\}_{u \in U_n}, Reg, \mathsf{EK})$: similar to KeyGen

$\mathsf{Encaps}(\mathsf{EK}, Reg, \mathcal{S}^*)$:

1. From the target set S^*, generate a partition $S^* = \cup_{\mathcal{L}^*} S_i$, we assume of size ℓ;
2. Generate two session keys $\mathcal{K}^0_e$ and $\mathcal{K}^1_e$, as well as a MAC key $\mathcal{K}^0_m$;
3. For each subset $i \in \mathcal{L}^*$, generate $c^*_i = \mathcal{PKE}.\mathsf{Encrypt}(\mathsf{ek}_{S_i}, \mathcal{K}^0_e || \mathcal{K}^0_m)$;
4. Then, compute $\sigma^* = \mathcal{MAC}.\mathsf{GenMac}(\mathcal{K}^0_m, S^* || (c^*_i)_{i \in \mathcal{L}^*})$;
5. Outputs $\mathcal{K}^0_e$, $\mathcal{K}^1_e$ and $H^* = ((c^*_i)_{i \in \mathcal{L}^*}, \sigma^*)$.

$\mathsf{ODecaps}(u, S, H)$:

1. If u is in S, then there is a unique i such that $u \in S_i$, and then dk_u allows to derive dk_{S_i};
2. Extract $\mathcal{K}_e||\mathcal{K}_m = \mathcal{PKE}.\mathsf{Decrypt}(\mathsf{dk}_{S_i}, c_i)$;
3. If $i \in \mathcal{L}^*$ and $c_i = c_i^*$, check if σ is a valid MAC under key $\mathcal{K}_m$;
4. Else, check if σ is a valid MAC under key $\mathcal{K}_m$;
5. In case of validity, output $\mathcal{K}_e$, otherwise output $\perp$.

Game $\mathbf{G}_1$: We first replace all the proto-keys by random keys: we thus apply the key indistinguishability of the $\mathcal{SKE}$ scheme:

$\mathsf{KeyGen}(\mathsf{param}, U_n)$:

1. All the proto-keys pt_S are drawn independently at random for all subsets S;

The difference between $\mathbf{G}_1$ and $\mathbf{G}_0$ is bounded by

$$\Pr_1[\mathcal{A} \to 1] - \Pr_0[\mathcal{A} \to 1] \leq \mathrm{Adv}^{\mathsf{ind}}_{\mathcal{SKE}}(k, t, L_N, L_N).$$

Game $\mathbf{G}_2$: We now replace all PKE keys by random keys: we thus apply the pseudo-randomness of the PRG $\mathcal{F}$:

$\mathsf{KeyGen}(\mathsf{param}, U_n)$:

3. Each user gets $(\mathsf{dk}_S, \mathsf{ek}_S) \leftarrow \mathcal{PKE}.\mathsf{KeyGen}(\mathsf{param}_{\mathcal{PKE}}; r_S)$, where r_S are random coins, for all subsets S he belongs to;

Using a classical hybrid proof, the difference between $\mathbf{G}_2$ and $\mathbf{G}_1$ is bounded by

$$\Pr_2[\mathcal{A} \to 1] - \Pr_1[\mathcal{A} \to 1] \leq L_N \times \mathrm{Adv}^{\mathsf{prg}}_{\mathcal{F}}(k, t).$$

Game $\mathbf{G}_3$: We introduce an additional MAC key that will be used later in the sub-ciphertexts:

$\mathsf{Encaps}(\mathsf{EK}, Reg, S^*)$:

2. Generate two session keys $\mathcal{K}_e^0$ and $\mathcal{K}_e^1$, as well as two MAC keys $\mathcal{K}_m^0$ and $\mathcal{K}_m^1$;

$\mathbf{G}_3$ and $\mathbf{G}_2$ are perfectly indistinguishable:

$$\Pr_3[\mathcal{A} \to 1] = \Pr_2[\mathcal{A} \to 1].$$

Game $\mathbf{G}_4$: We now use the additional MAC key $\mathcal{K}_m^1$ in the challenge sub-ciphertexts, but still use $\mathcal{K}_m^0$ for the MAC computation:

$\mathsf{Encaps}(\mathsf{EK}, Reg, S^*)$:

3. For each subset $i \in \mathcal{L}^*$, generate $c_i^* = \mathcal{PKE}.\mathsf{Encrypt}(\mathsf{ek}_i, \mathcal{K}_e^0||\mathcal{K}_m^1)$;

Lemma 15. *The difference between $\boldsymbol{G}_4$ and $\boldsymbol{G}_3$ is bounded by*

$$\Pr_4[\mathcal{A} \to 1] - \Pr_3[\mathcal{A} \to 1] \leq \ell \times L_N \times \mathrm{Adv}^{\mathsf{ind-cca}}_{\mathcal{PKE}}(k, t, q_D).$$

Game $\mathbf{G}_5$: In this game, we reject decryption queries that should decrypt a sub-ciphertext from the challenge ciphertext.

> $\mathsf{ODecaps}(u, S, H = ((c_i)_{i\in\mathcal{L}}, \sigma))$:
>
> 3. If $i \in \mathcal{L}^*$ and $c_i = c_i^*$, output $\perp$;

Lemma 16. *The difference between $\boldsymbol{G}_5$ and $\boldsymbol{G}_4$ is bounded by*

$$\Pr_5[\mathcal{A} \to 1] - \Pr_4[\mathcal{A} \to 1] \leq \mathsf{Succ}^{\mathsf{suf-cma}}_{\mathcal{MAC}}(k, t, 1, q_D).$$

Game $\mathbf{G}_6$: We define the game $\mathbf{G}_6$ as the game $\mathbf{G}_5$, but we encapsulate $\mathcal{K}_e^1$ instead of $\mathcal{K}_e^0$:

> $\mathsf{Encaps}(\mathsf{EK}, \mathit{Reg}, S^*)$:
>
> 3. For each subset $i \in \mathcal{L}^*$, generate $c_i^* = \mathcal{PKE}.\mathsf{Encrypt}(\mathsf{ek}_{S_i}, \mathcal{K}_e^1 || \mathcal{K}_m^1)$;

Lemma 17. *The difference between $\boldsymbol{G}_6$ and $\boldsymbol{G}_5$ is bounded by*

$$\Pr_6[\mathcal{A} \to 1] - \Pr_5[\mathcal{A} \to 1] \leq \ell \times L_N \times \mathrm{Adv}^{\mathsf{ind-cca}}_{\mathcal{PKE}}(k, t, q_D).$$

Game $\mathbf{G}_7$: Previous game is similar to $\mathbf{G}_5$, but with $\mathcal{K}_e^1$ in the challenge ciphertext. We now go back, as in game $\mathbf{G}_4$: we check MAC values of sub-ciphertexts of the challenge ciphertext under $\mathcal{K}_m^0$:

> $\mathsf{ODecaps}(u, S, H)$:
>
> 3. If $i \in \mathcal{L}^*$ and $c_i = c_i^*$, check if σ is a valid MAC under key $\mathcal{K}_m^0$.

Since we have the same gap as from $\mathbf{G}_4$ to $\mathbf{G}_5$:

$$\Pr_7[\mathcal{A} \to 1] - \Pr_6[\mathcal{A} \to 1] \leq \mathsf{Succ}^{\mathsf{suf-cma}}_{\mathcal{MAC}}(k, t, 1, q_D).$$

Game $\mathbf{G}_8$: We eventually change back the use of the MAC key $\mathcal{K}_m^0$ in the challenge sub-ciphertexts:

> $\mathsf{Encaps}(\mathsf{EK}, \mathit{Reg}, S^*)$:
>
> 3. For each subset $i \in \mathcal{L}^*$, generate $c_i^* = \mathcal{PKE}.\mathsf{Encrypt}(\mathsf{ek}_{S_i}, \mathcal{K}_e^1 || \mathcal{K}_m^0)$;

Since we have the same gap as from $\mathbf{G}_3$ to $\mathbf{G}_4$:

$$\Pr_8[\mathcal{A} \to 1] - \Pr_7[\mathcal{A} \to 1] \leq \ell \times L_N \times \mathrm{Adv}^{\mathsf{ind-cca}}_{\mathcal{PKE}}(k, t, q_D).$$

We do not use anymore the key $\mathcal{K}_m^1$: this is exactly the $\mathsf{IND\text{-}ACCA}$ security game with $b = 1$, except for the generation of the encryption keys.

Game $\mathbf{G}_9$: We now change back the generation of the encryption keys, using the $\mathcal{SKE}$ protocol and the PRG:

$$\Pr_9[\mathcal{A} \to 1] - \Pr_8[\mathcal{A} \to 1] \leq \mathrm{Adv}^{\mathsf{ind}}_{\mathcal{SKE}}(k, t, L_N, L_N) + L_N \times \mathrm{Adv}^{\mathsf{prg}}_{\mathcal{F}}(k, t).$$

If we sum up all the gaps, we obtain:

$$\begin{aligned}
\Pr_1[\mathcal{A}\to 1] - \Pr_0[\mathcal{A}\to 1] &\leq \mathrm{Adv}^{\mathsf{ind}}_{\mathcal{SKE}}(k,t,L_N,L_N)\\
\Pr_2[\mathcal{A}\to 1] - \Pr_1[\mathcal{A}\to 1] &\leq L_N \times \mathrm{Adv}^{\mathsf{prg}}_{\mathcal{F}}(k,t)\\
\Pr_4[\mathcal{A}\to 1] - \Pr_2[\mathcal{A}\to 1] &\leq \ell \times L_N \times \mathrm{Adv}^{\mathsf{ind-cca}}_{\mathcal{PKE}}(k,t,q_D)\\
\Pr_5[\mathcal{A}\to 1] - \Pr_4[\mathcal{A}\to 1] &\leq \mathsf{Succ}^{\mathsf{suf-cma}}_{\mathcal{MAC}}(k,t,1,q_D)\\
\Pr_6[\mathcal{A}\to 1] - \Pr_5[\mathcal{A}\to 1] &\leq \ell \times L_N \times \mathrm{Adv}^{\mathsf{ind-cca}}_{\mathcal{E}}(k,t,q_D)\\
\Pr_7[\mathcal{A}\to 1] - \Pr_6[\mathcal{A}\to 1] &\leq \mathsf{Succ}^{\mathsf{suf-cma}}_{\mathcal{MAC}}(k,t,1,q_D)\\
\Pr_8[\mathcal{A}\to 1] - \Pr_7[\mathcal{A}\to 1] &\leq \ell \times L_N \times \mathrm{Adv}^{\mathsf{ind-cca}}_{\mathcal{PKE}}(k,t,q_D)\\
\Pr_9[\mathcal{A}\to 1] - \Pr_8[\mathcal{A}\to 1] &\leq \mathrm{Adv}^{\mathsf{ind}}_{\mathcal{SKE}}(k,t,L_N,L_N) + L_N \times \mathrm{Adv}^{\mathsf{prg}}_{\mathcal{F}}(k,t)
\end{aligned}$$

And this concludes the proof, since $\ell \leq \ell_N$.

Proof of Lemma 15. Let ℓ be the size of the partition $\mathcal{L}^*$. In order to so show that the adversary cannot detect whether we use the same MAC key that is part of the ciphertext or not, we proceed by another sequence of hybrid games: We define the game G_j (for $j = 0,\ldots,\ell$), in which the j-first sub-ciphertexts c_i^* are defined as in $\mathbf{G}_4$, that is $c_i^* = \mathcal{PKE}.\mathsf{Encrypt}(\mathsf{ek}_i, \mathcal{K}_e^0||\mathcal{K}_m^1)$, and the next ones are defined as in $\mathbf{G}_3$, that is $c_i^* = \mathcal{PKE}.\mathsf{Encrypt}(\mathsf{ek}_{S_i}, \mathcal{K}_e^0||\mathcal{K}_m^0)$. It is clear that $G_0 = \mathbf{G}_3$, whereas $G_\ell = \mathbf{G}_4$.

For any $J \in [0,\ell]$, let us play the following game against the IND-CCA challenger of the $\mathcal{PKE}$ encryption scheme:

- Setup/KeyGen:
  1. We receive the challenge public key ek;
  2. We randomly choose one subset $I \in [1, L_N]$ (we bet it will correspond to the J-th ciphertext in the target partition $\mathcal{L}^*$. This guess is correct with probability $1/L_N$, otherwise we abort the game and make $\mathcal{B}$ output 0);
  3. We generate all the pairs $(\mathsf{dk}_{S_i}, \mathsf{ek}_{S_i})$ at random, except for $i = I$, where $\mathsf{ek}_{S_I} = \mathsf{ek}$;
- Encaps(EK, Reg, S^*):
  1. From the target set S^*, generate a partition $S^* = \cup_{\mathcal{L}^*} S_i$, we assume of size ℓ;
  2. If our guess at setup time was correct, the J-th element in $\mathcal{L}^*$ is I;
  3. Generate two session keys $\mathcal{K}_e^0$ and $\mathcal{K}_e^1$, as well as two MAC keys $\mathcal{K}_m^0$ and $\mathcal{K}_m^1$;
  4. For the $J-1$-first elements $i \in \mathcal{L}^*$, generate $c_i^* = \mathcal{PKE}.\mathsf{Encrypt}(\mathsf{ek}_{S_i}, \mathcal{K}_e^0||\mathcal{K}_m^1)$;
  5. For the J-th element $i \in \mathcal{L}^*$, assumed to be I, ask to the IND-CCA-challenger on the two plaintexts $\mathcal{K}_e^0||\mathcal{K}_m^0$ and $\mathcal{K}_e^0||\mathcal{K}_m^1$, and set c_I^* to be the answer, according to the internal bit b of the IND-CCA challenger;
  6. For the next elements $i \in \mathcal{L}^*$, generate $c_i^* = \mathcal{PKE}.\mathsf{Encrypt}(\mathsf{ek}_{S_i}, \mathcal{K}_e^0||\mathcal{K}_m^0)$;
  7. Then, compute $\sigma^* = \mathcal{MAC}.\mathsf{GenMac}(\mathcal{K}_m^0, S^*||(c_i^*)_{i\in\mathcal{L}^*})$;
  8. Output $\mathcal{K}_e^0$, $\mathcal{K}_e^1$ and $H^* = ((c_i^*)_{i\in\mathcal{L}^*}, \sigma^*)$.
- OCorrupt Queries: Since we condition on the good choice for I, we can answer all the OCorrupt queries by outputting the corresponding decryption keys (they cannot be for I, otherwise the challenge target set would contain corrupted players);
- ODecaps Queries:
  1. If this is for a player that lies in a set $S_i \notin S_I$, we can easily decrypt c_i;
  2. If this is for a player that lies in S_I,

- either $c_I \neq c_I^*$, and then we can ask the decryption query to the decryption oracle
- or $c_I = c_I^*$, then check the MAC value with $\mathcal{K}_m^0$. If it is valid, output $\mathcal{K}_e^0$, otherwise output $\perp$.

Our adversary $\mathcal{B}$ against IND-CCA simply forwards the output b' of $\mathcal{A}$ (or outputs zero in case of abort):

$$\begin{aligned}\Pr[\mathcal{B}\to 1|b=0]-\Pr[\mathcal{B}\to 1|b=1] &= \Pr[\mathcal{B}\to 1\wedge I|b=0]-\Pr[\mathcal{B}\to 1\wedge I|b=1]\\ &\quad+\Pr[\mathcal{B}\to 1\wedge \neg I|b=0]-\Pr[\mathcal{B}\to 1\wedge \neg I|b=1]\\ &= \frac{1}{L_N}\times(\Pr[\mathcal{B}\to 1|b=0\wedge I]-\Pr[\mathcal{B}\to 1|b=1\wedge I])\\ &= \frac{1}{L_N}\times(\Pr[\mathcal{A}\to 1|b=0\wedge I]-\Pr[\mathcal{A}\to 1|b=1\wedge I])\,.\end{aligned}$$

In the RHS, the output is independent of the correct guess of I, whereas the LHS is bounded by the best advantage against IND-CCA within time t:

$$\frac{1}{L_N}\times|\Pr[\mathcal{A}\to 1|b=0]-\Pr[\mathcal{A}\to 1|b=1]|\leq \mathrm{Adv}^{\mathsf{ind-cca}}_{\mathcal{PKE}}(k,t,q_D).$$

Furthermore, the above game with $b=0$ is exactly G_{J-1}, whereas with $b=1$ this is G_J:

$$\left|\Pr_{G_{J-1}}[\mathcal{A}\to 1]-\Pr_{G_J}[\mathcal{A}\to 1]\right|\leq L_N\times\mathrm{Adv}^{\mathsf{ind-cca}}_{\mathcal{PKE}}(k,t,q_D).$$

This concludes the proof.

Proof of Lemma 16. In order to show that the adversary cannot detect whether we reject a valid MAC under the unknown key $\mathcal{K}_m^0$, we use the following game against the MAC: more precisely, we play the SUF-CMA security game against the MAC, using the challenge MAC key sk as the unknown $\mathcal{K}_m^0$ key. All the other keys are known to the simulator. The MAC generation oracle OGenMac is called for the challenge MAC value by the Encaps oracle, and the MAC verification oracle OVerifMac is called in case of a challenge sub-ciphertext in a decapsulation ODecaps oracle query. A valid MAC value asked to OVerifMac is a forgery, otherwise it should be a reject. Hence, the probability that a valid MAC value is refused is bounded by $\mathsf{Succ}^{\mathsf{suf-cma}}_{\mathcal{MAC}}(k,t,1,q_D)$.

Proof of Lemma 17. Let ℓ be the size of the partition $\mathcal{L}^*$. In order to so show that the adversary cannot detect whether we encrypt $\mathcal{K}_e^0$ or $\mathcal{K}_e^1$, we proceed as for the proof of Lemma 15, by a sequence of hybrid games: We define the game G_j (for $j=0,\ldots,\ell$), in which the j-first sub-ciphertexts c_i^* are defined as in $\mathbf{G}_6$, that is $c_i^* = \mathcal{PKE}.\mathsf{Encrypt}(\mathsf{ek}_i, \mathcal{K}_e^1||\mathcal{K}_m^1)$, and the next ones are defined as in $\mathbf{G}_5$, that is $c_i^* = \mathcal{PKE}.\mathsf{Encrypt}(\mathsf{ek}_i, \mathcal{K}_e^0||\mathcal{K}_m^1)$. It is clear that $G_0 = \mathbf{G}_5$, whereas $G_\ell = \mathbf{G}_6$. Exactly the same analysis as in the proof of Lemma 15 leads to the result. The trick comes from the simulation of ODecaps Queries, in which we output $\perp$ in case a sub-ciphertext of the challenge ciphertext is involved. We do not have to care whether this is $\mathcal{K}_e^0$ or $\mathcal{K}_e^1$.

C Proof of Theorem 7

Let $\mathcal{A}$ be an adversary against the IND-security of our CS construction $\mathcal{SKE}$ that invokes at most N users, among them, the user set U_M that runs the KeyGen protocol (where $M=|U_M|$ denotes its size) and T users that join once at a time, in T time steps. Since in each of the T time periods at most $\log N$ nodes are updated, at most $N\log N$ keys will be generated overall (for $M=1$). Game $\mathrm{Exp}^{\mathsf{ind-0}}_{\mathcal{SKE},\mathcal{A}}(k)$ is the experiment where all keys are generated as usual. This will be our initial game. Game $\mathrm{Exp}^{\mathsf{ind-1}}_{\mathcal{SKE},\mathcal{A}}(k)$ is the experiment where all

keys are chosen uniformly at random. This will be our final game. To go from the first game to the final one, we define intermediate games, in which we first replace the session keys that are produced by the two-player key exchange protocols by random keys, and then we replace the proto-keys by random keys.

Game $\mathbf{G}_0$: This is the initial game, that appears in the experiment where $b = 0$.

> $\mathsf{KeyGen}(U_M)$: In round r, for $r = \log M, \ldots, 1$, the simulator executes the following steps for each node w at level $(\log M - r)$ of the tree with children u, u':
>
> 1. $\mathsf{ck}_w \leftarrow \mathcal{KE}.\mathsf{CommonKey}(u, u')$;
> 2. $(\mathsf{pt}_w, r_w) \leftarrow \mathcal{G}(\mathsf{ck}_w)$;
> 3. If $u < u'$, set $u \stackrel{\mathsf{def}}{=} w$.

> $\mathsf{Join}(v, U_n)$ In the first round, set $u = v$. In round r, for $r = \log(n+1), \ldots, 1$, the simulator executes the following steps for user u with neighbour u' and parent w at level $(\log(n+1) - r)$:
>
> 1. $\mathsf{ck}_w \leftarrow \mathcal{KE}.\mathsf{CommonKey}(u, u')$;
> 2. $(\mathsf{pt}_w, r_w) \leftarrow \mathcal{G}(\mathsf{ck}_w)$;
> 3. set $u \stackrel{\mathsf{def}}{=} w$, $u' \stackrel{\mathsf{def}}{=} neighbour(w)$, $w \stackrel{\mathsf{def}}{=} parent(w)$;

Game $\mathbf{G}_1$: We replace all KE session keys on level 1 of the tree with random keys.

> $\mathsf{KeyGen}(U_M)$: In round 1, the simulator executes the following steps for each node w at level 1 of the tree with children u, u':
>
> 1. $\mathsf{ck}_w \leftarrow \mathcal{KE}.\mathsf{CommonKey}(u, u')$; $\mathsf{ck}_w \stackrel{\$}{\leftarrow} \mathcal{K}_{\mathcal{KE}}$;

With a classical hybrid proof, where we successively replace all the real keys by random keys in the $M/2$ two-party key exchanges, we get that the difference between $\mathbf{G}_1$ and $\mathbf{G}_0$ is bounded by

$$\Pr_1[\mathcal{A} \to 1] - \Pr_0[\mathcal{A} \to 1] \leq \frac{M}{2}\mathrm{Adv}^{\mathsf{ind}}_{\mathcal{KE}}(k, t).$$

Game $\mathbf{G}_2$: We replace all proto-keys on level 1 of the tree with random keys.

> $\mathsf{KeyGen}(U_M)$: In round 1, the simulator executes the following steps for each node w at level 1 of the tree with children u, u':
>
> 2. $(\mathsf{pt}_w, r_w) \stackrel{\$}{\leftarrow} \mathcal{K} \times \mathcal{R}_{\mathcal{KE}}$;

With a classical hybrid proof, where we successively replace all the real values by random values in the $M/2$ key derivations, we get that the difference between $\mathbf{G}_2$ and $\mathbf{G}_1$ is bounded by

$$\Pr_2[\mathcal{A} \to 1] - \Pr_1[\mathcal{A} \to 1] \leq \frac{M}{2}\mathrm{Adv}^{\mathsf{prg}}_{\mathcal{G}}(k, t).$$

Game $\mathbf{G}_3$: We replace all proto-keys in the initial tree with random keys.

$\mathsf{KeyGen}(U_M)$: In round r, the simulator executes the following steps for each node w at level r of the tree with children u, u':

1. $\mathsf{ck}_w \leftarrow \mathcal{KE}.\mathsf{CommonKey}(u, u')$; $\mathsf{ck}_w \xleftarrow{\$} \mathcal{K}_{\mathcal{KE}}$;
2. $(\mathsf{pt}_j, r_j) \xleftarrow{\$} \mathcal{K} \times \mathcal{R}_{\mathcal{KE}}$;

By applying iteratively the 2 previous hops at level 2 on $M/2^2$ pairs, and at level 3 on $M/2^3$ pairs, etc, we get that the difference between $\mathbf{G}_3$ and $\mathbf{G}_2$ is bounded by

$$\Pr_3[\mathcal{A} \to 1] - \Pr_2[\mathcal{A} \to 1] \leq (\frac{M}{2} - 1) \left(\mathrm{Adv}^{\mathsf{ind}}_{\mathcal{KE}}(k,t) + \mathrm{Adv}^{\mathsf{prg}}_{\mathcal{G}}(k,t) \right).$$

Game $\mathbf{G}_4$: We replace all proto-keys created during joins with random keys. The result is a protocol where all proto-keys are drawn independently at random, which describes the experiment with $b = 1$.

$\mathsf{Join}(v, U_n)$

1. $\mathsf{ck}_w \leftarrow \mathcal{KE}.\mathsf{CommonKey}(u, u')$; $\mathsf{ck}_w \xleftarrow{\$} \mathcal{K}_{\mathcal{KE}}$;
2. $(\mathsf{pt}_j, r_j) \xleftarrow{\$} \mathcal{K} \times \mathcal{R}_{\mathcal{KE}}$;

Lemma 18. *The difference between $\boldsymbol{G}_4$ and $\boldsymbol{G}_3$ is bounded by*

$$\Pr_4[\mathcal{A} \to 1] - \Pr_3[\mathcal{A} \to 1] \leq (T \log N) \left(\mathrm{Adv}^{\mathsf{ind}}_{\mathcal{KE}}(k,t) + \mathrm{Adv}^{\mathsf{prg}}_{\mathcal{G}}(k,t) \right).$$

In summary, we have

$$\Pr_3[\mathcal{A} \to 1] - \Pr_0[\mathcal{A} \to 1] \leq (M-1) \left(\mathrm{Adv}^{\mathsf{ind}}_{\mathcal{KE}}(k,t) + \mathrm{Adv}^{\mathsf{prg}}_{\mathcal{G}}(k,t) \right)$$
$$\Pr_4[\mathcal{A} \to 1] - \Pr_3[\mathcal{A} \to 1] \leq (T \log N) \left(\mathrm{Adv}^{\mathsf{ind}}_{\mathcal{KE}}(k,t) + \mathrm{Adv}^{\mathsf{prg}}_{\mathcal{G}}(k,t) \right).$$

Because $M + T = N$, we obtain

$$\mathrm{Adv}^{\mathsf{ind}}_{\mathcal{SKE}}(k,t,N,q_T) \leq (N \log N) \left(\mathrm{Adv}^{\mathsf{ind}}_{\mathcal{KE}}(k,t) + \mathrm{Adv}^{\mathsf{prg}}_{\mathcal{G}}(k,t) \right).$$

Note that this is independent of q_T, because we change all the keys.

Proof of Lemma 18. Let G_0 be the game $\mathbf{G}_3$, G_T be the game $\mathbf{G}_4$. We define $T-1$ intermediate hybrid games G_j ($j = 1 \ldots T-1$), in which we replace all session keys and proto-keys computed during the first j joins with random keys. We proceed as in the previous proofs and obtain

$$\Pr_j[\mathcal{A} \to 1] - \Pr_{j-1}[\mathcal{A} \to 1] \leq \log N \cdot \left(\mathrm{Adv}^{\mathsf{ind}}_{\mathcal{KE}}(k,t) + \mathrm{Adv}^{\mathsf{prg}}_{\mathcal{G}}(k,t) \right).$$

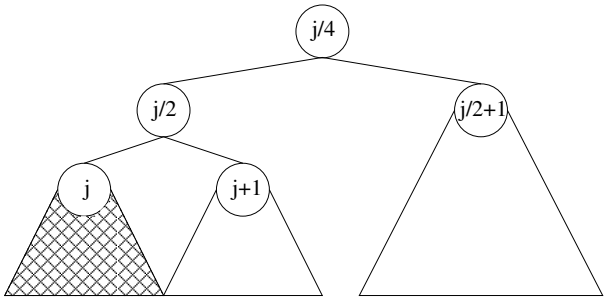

Figure 6: SD Key Assignment for $S_{j/4,j}$

D Constructions for the Subset Difference Method

SD Tree. We can modify the construction of the above CS tree to obtain keys for any subset $S_{i,j} = S_i \setminus S_j$, when $S_j \subset S_i$: To exclude all leaves below node j (w.l.o.g. $j \equiv 0 \mod 4$), we skip the key exchange between j and $j+1$ and directly compute a common key $\mathsf{ck}_{j/2+1,j+1}$ between $j/2+1$ (j's uncle) and $j+1$ (j's sibling). Basically, we identify the public key of $j+1$ with that of $S_{j/2,j}$. After applying $\mathcal{G}$, we have the two key pairs for $S_{j/4,j}$, which we treat as being at node $j/4$. We then continue with $\mathcal{KE}.\mathsf{CommonKey}$ and $\mathcal{G}$ as normal up to node i, to get $S_{i,j}$. This allows us to construct an SD tree in much the same way as a CS tree, except that we "omit" one node in the computation of the key (see figure 6). Each node i at depth ℓ ($2^\ell \leq i < 2^{\ell+1}$) contains $2^{d-\ell+1} - 4$ blocks of keys that can be computed iteratively, excluding all the possible subtrees, from depth $\ell + 2$ (4 of them) to d ($2^{d-\ell}$ of them).

We define the neighbour and parent of user i as for the CS scheme, and the neighbour of (i, j) to be the neighbour of i. At round r, each user (i, j) created in round $r - 1$ has a uniquely defined neighbour i' (if he does not, the protocol run is completed). They both have random coins r_i and $r_{i'}$:

- If $j \neq i'$, it runs $\mathsf{ck}_{i/2,j} \leftarrow \mathcal{KE}.\mathsf{CommonKey}((i, j)(r_i), i'(r_{i'}))$ and sets $(\mathsf{pt}_{i/2}, r_{i/2,j}) \leftarrow \mathcal{G}(c_{i/2,j})$ to derive the information for the node $(i/2, j)$.
- If $j = i'$, it runs $\mathsf{ck}_{i/2} \leftarrow \mathcal{KE}.\mathsf{CommonKey}((i, j)(r_i), i'(r_{i'}))$ and sets $(\mathsf{pt}_{i/2}, r_{i/2}) \leftarrow \mathcal{G}(c_{i/2})$ to derive the information for the node $i/2$.

If $i < i'$, it plays the role of the virtual user $i/2$ in the next round.

Dynamic SD. To join a user, we go from $\mathcal{SC}_n$ to $\mathcal{SC}_{n+1}$ by appending the new user as described for CS. Then we replace those subsets $S_{i,j}$ that contain the new user with the new subsets.

We show that our dynamic SD scheme has the splitting property of definition 4. All $S_{i,j}$ for which i is not an ancestor of the new node are unchanged. All $S_{i,j}$ for which i, but not j is an ancestor of the new node contain the new user. All $S_{i,j}$ for which i is an ancestor of the new node and j is a true ancestor of the new node are unchanged as well. All $S_{i,j}$ for which i is an ancestor and j is the new node correspond to full subtrees $S_{\mathrm{parent}(i),\mathrm{sibling}(i)}$ in the old subset cover. The key assignment for SD is similar to the CS key assignment, but we cannot identify nodes and subsets, and must "jump" the omitted subtree in the computation (figure 6).