Public Key Broadcast Encryption for Stateless Receivers

Yevgeniy Dodis and Nelly Fazio

Computer Science Department
New York University
New York, NY 10012, USA
{dodis, fazio}@cs.nyu.edu

Abstract. A *broadcast encryption* scheme allows the sender to securely distribute data to a dynamically changing set of users over an insecure channel. One of the most challenging settings for this problem is that of *stateless receivers*, where each user is given a fixed set of keys which cannot be updated through the lifetime of the system. This setting was considered by Naor, Naor and Lotspiech [17], who also present a very efficient "subset difference" (SD) method for solving this problem. The efficiency of this method was recently improved by Halevi and Shamir [12], who called their refinement the "Layered SD" (LSD) method. Both of the above methods were originally designed to work in the symmetric key setting, where only the trusted designer of the system can encrypt messages to users. On the other hand, in many applications it is desirable not to store the secret keys "on-line", or to allow untrusted users to broadcast information. This leads to the question of building a *public key* broadcast encryption scheme for stateless receivers; in particular, of extending the elegant SD/LSD methods to the public key setting. Naor et al. [17] notice that the natural technique for doing so will result in an enormous public key and very large storage for every user. In fact, [17] pose this question of reducing the public key size and user's storage as the first open problem of their paper. We resolve this question in the affirmative, by demonstrating that an $O(1)$ size public key can be achieved for both of SD/LSD methods, in addition to the same (small) user's storage and ciphertext size as in the symmetric key setting.

1 Introduction

Broadcast Encryption. Broadcast encryption provides a convenient way to distribute digital content to subscribers over an insecure broadcast channel. Namely, it allows the sender to deliver information to a dynamically changing sets of users in such a way that only the "qualified" users can recover the data. Not surprisingly, it has found many applications including pay-TV systems, distribution of copyrighted material, streaming audio/video and many others.

Since its introduction by Fiat and Naor [8], the problem received significant attention, and many of its variants have been studied. To name just a few, the set of receivers can be fixed, slowly changing or rapidly changing; the scheme can support a single, bounded or unbounded number of broadcasts; it might or might not be possible to periodically refresh users' secret keys; the scheme might support bounded or unbounded number of "revoked" users; it might be possible to trace "pirates" who gave away an illegal decryption device (this is called *traitor tracing*); the scheme could be private or public key based; etc. We mention just several of the relevant works: [22, 15, 16, 23, 9, 4, 5, 14, 10, 18, 21].

We study one of the most difficult variants of the problem when the receivers are *stateless*. Namely, each user is given a fixed set of keys which cannot be updated through the lifetime of the system. In particular, they do not change when other users join or leave the system, or evolve based on the history of past transmissions. Instead, each transmission must be decrypted solely on the base of the fixed initial configuration of each user's decryption device. As argued by Naor, Naor and Lotspiech [17] (who were the first to explicitly concentrate on this scenario), the stateless receivers case is quite common. For example, the receivers might not be constantly on-line to view past history or update their secret keys, or the keys might be put "once-and-for-all" into a tamper-resistant device. Additionally, the scheme should support an unbounded number of broadcasts, and be capable — at least in principle — to revoke an a-priori unbounded number of users (possibly at the cost of reduced efficiency). In particular, even the coalition of *all* the "non-privileged" users combined cannot decrypt a given transmission, even if this set is adaptively chosen by a central adversary. Finally, the above features also imply that consecutive broadcasts can revoke arbitrary and potentially unrelated subsets of users, and no "key maintenance" is necessary.

Up to date, the only type of scheme enjoying all these properties was designed by Naor et al. [17] (and was recently improved by Halevi and Shamir [12]). We will describe these schemes in more detail shortly.

Public vs. Symmetric Key. As we mentioned, one important distinction between various broadcast encryption schemes is whether they are public key or symmetric key based. In the latter variant, only the trusted designer of the system can broadcast data to the receivers. In other words, in order to encrypt the content, one needs to know some sensitive information (typically, the secret keys of all the users of the system) whose disclosure will compromise the security of the system. Even though symmetric key broadcast encryption is sufficient for many applications, it has a few shortcomings. First, in certain situations we would like to allow possibly untrusted users to broadcast information. Additionally, it requires the sender to store all the secret keys of the system, making it a single point of failure. In contrast, in the public key setting the trusted designer of the system publishes a short public key which enables anybody to broadcast data, thus overcoming the above mentioned deficiencies of the symmetric setting.

The original schemes of [17] were primarily designed for the symmetric key setting. Briefly, the so called *Subset-Cover* methodology of [17] (described in detail later) has the system designer (called the *Center*) generate many "computationally unrelated" secret keys $k_1 \ldots k_w$ (where w is "large") and distribute various subsets of these keys to different users. To encrypt the message to a specified subset of privileged users, a certain small, carefully chosen subset of these keys is used. Even though this suggests that the Center must store all w keys, this typically does not have to be the case. Indeed, standard symmetric key tools like pseudorandom functions can be used to significantly compress the storage requirement of the Center (typically, to a single random seed). This is indeed the case for the two specific instantiations of the Subset-Cover framework proposed by [17] — the *Complete Subtree* (CS) method and a more efficient *Subset Difference* (SD) method — as well as for the further improved *Layered Subset Difference* (LSD) method of [12]. Similarly, even though each user might need to have too many of the secret keys $k_1 \ldots, k_w$ (which is really the case in the more efficient SD/LSD methods), it is possible — albeit somewhat more difficult — to compress the user's storage using similar tools, as was indeed done by [17].

As already noted by Naor et al. [17], the general Subset-Cover framework can in principle be adapted to the public key setting, by having each key k_j replaced by some pair of public/secret keys (PK_j, SK_j). Unfortunately, the simple compression methods of the symmetric key setting are much harder to come by in the public key setting. Even ignoring the problem with the user's storage, the natural implementation will have to publish all the local public keys $PK_1, \ldots, PK_w$, yielding a huge public key for the system. Naor et al. [17] briefly mention that the tools from *Identity-Based Cryptography* [19] seem to overcome this problem, at least for the (less efficient) CS method, where each user needs to know very few secret keys anyway. However, the Identity-Based Encryption (IBE) scheme alone does not seem to be sufficient for the more efficient SD/LSD methods, since it does not resolve the problem of compressing large storage requirement of each user. In fact, the question of efficiently extending the SD (and similar LSD) method(s) to the public key setting was given as the first open problem in [17].

Our Main Result. We resolve this problem in the affirmative, by non-trivially utilizing the concept of *Hierarchical Identity-Based Encryption* (HIBE) [11, 13]. In particular, we show that one can get essentially all the benefits of the symmetric key versions of the SD/LSD methods (including the same small storage per user) in the public key setting, while having a fixed *constant size* public key. As an intermediate step toward this goal, we indicate which changes should be made to the general Subset-Cover framework of [17] in order to translate it to the public key setting, and also formally verify that "plain" IBE is indeed sufficient to translate the (less efficient) CS method to the public key setting. The particular parameters we get can be summarized as follows when revoking r out of N total users (in all cases, the public key size and the storage of the Center are $O(1)$):

- **CS method**. The ciphertext consists of $r\log(N/r)$ identity based encryptions, each users stores $O(\log N)$ keys and needs to perform a single identity based decryption.
- **SD method**. The ciphertext consists of $(2r-1)$ hierarchical identity based encryptions (of "depth" at most $\log N$ each), each users stores $O(\log^2 N)$ keys and needs to perform a single hierarchical identity based decryption.
- **LSD method**. For any $\epsilon > 0$, the ciphertext consists of $O(r/\epsilon)$ hierarchical identity based encryptions (of "depth" at most $\log N$ each), each users stores $O(\log^{1+\epsilon} N)$ keys and needs to perform a single hierarchical identity based decryption.

Interestingly, when instantiated with best currently known IBE [3] and HIBE [11] schemes, the CS method actually becomes slightly preferable to the "in principle" more efficient SD/LSD methods. This is due to the fact that the length of the encryption of the specific HIBE [11] is proportional to the "depth" in the hierarchy (see Appendix A). Thus, the actual transmission rate in SD/LSD methods deteriorates to $O(r \log N)$, as in the CS method (while the latter still having a smaller storage requirement per user and a slightly cheaper decryption time). Still, if a more efficient HIBE is found, the original "transmission rate" advantages of the SD/LSD methods will again kick into effect.

Comparison to Existing Public Key Schemes. There already exist several public key broadcast encryption schemes [18, 21, 6] in the stateless receivers scenario, all based on the decisional Diffie-Hellman assumption. However, all these schemes can revoke up to at most an a-priori fixed number of users, $r_{\max}$. Moreover, the size of the transmission is $O(r_{\max})$ even if no users are revoked. In contrast, the SD/LSD methods allow to revoke a dynamically changing (and potentially unbounded) number of users r, at the cost of having $O(r)$-size ciphertext transmission. More importantly, the reason the schemes of [18, 21, 6] support only a bounded number of revoked users, is that the public key (as well as encryption/decryption times) are proportional to $r_{\max}$. In contrast, the analogs of CS/SD/LSD schemes we construct all have a constant size public key, and the decryption time is at most logarithmic in the total number of users N. Finally, the schemes of [18, 21, 6] support only a limited form of traitor tracing (either "non-black-box" or "black-box confirmation"; see [6] for more discussion), while (as was shown in [17]) the CS/SD/LSD methods enjoy a significantly more powerful kind of "black-box" traitor tracing.

On a technical note, the Subset-Cover framework of [17] supports only the so called $\mathsf{CCA1}$-security [2] (chosen ciphertext security in the pre-processing mode [7]), since the message is encrypted independently with several "computationally unrelated" keys. On the other hand, the recently proposed scheme of [6] supports full chosen ciphertext security (so called $\mathsf{CCA2}$ [2, 7]). Even though it seems hard to extend the Subset-Cover framework to achieve $\mathsf{CCA2}$-security, it is possible to achieve a slightly relaxed (but essentially as useful) notion of $\mathsf{gCCA2}$-security recently proposed by [20, 1]. Since this is orthogonal to the main contribution of the paper, we postpone the details to the final version, concentrating on the $\mathsf{CCA1}$-security originally considered by [17] (which is sufficient for most applications).

2 Definitions

2.1 Broadcast Encryption

Definition 1 (Broadcast Encryption Scheme). *A Broadcast Encryption Scheme is a quadruple of* poly*-time algorithms* (KeyGen, Reg, Enc, Dec)*, where:*

– KeyGen*, the* key generation algorithm*, is a probabilistic algorithm used by the Center to set up all the parameters of the scheme.* KeyGen *takes as input a security parameter* 1^λ *and possibly a revocation threshold* $r_{\max}$ *(i.e. the maximum number of users that can be revoked) and generate the public key* PK *and the master secret key* SK*.*
– Reg*, the* registration algorithm*, is a probabilistic algorithm used by the Center to compute the secret initialization data to be delivered to a new user when he/she subscribes to the system.*
– Enc*, the* encryption algorithm*, is a probabilistic algorithm used to encapsulate a given session key* k *in such a way that the revoked users cannot recover it.* Enc *takes as input the public key* PK*, the session key* k *and a set* $\mathcal{R}$ *of revoked users (with* $|\mathcal{R}| \leq r_{\max}$*, if a threshold has been specified to the* KeyGen *algorithm) and returns the ciphertext to be broadcast.*
– Dec*, the* decryption algorithm*, is a deterministic algorithm that takes as input the secret data of a user* u *and the ciphertext broadcast by the center and returns the session key* k *that was sent if* u *was not in the set* $\mathcal{R}$ *when the ciphertext was constructed, or the special symbol* $\perp$ *otherwise.*

All the schemes that we will discuss are completely flexible in terms of the revocation threshold $r_{\max}$, i.e. they can tolerate an unbounded number of revoked users, at the only cost of increasing the length of the ciphertext.

Following [17], we briefly define the $\mathsf{CCA1}$-security of broadcast encryption (as stated earlier, one can define $\mathsf{CCA2}$-security as well). Upon seeing the public key PK, the adversary repeatedly perform (in any adaptively-chosen order) the following two steps: (1) corrupt any user u, thus obtaining the secret information u got when joining the system (let $\mathcal{R}$ be the final set of corrupted users; in case $r_{\max}$ is specified, we require $|\mathcal{R}| \leq r_{\max}$); (2) ask any user to decrypt a given ciphertext C chosen by the adversary. Then the adversary selects some session key k and gets back the value $\mathsf{Enc}(PK, k', \mathcal{R})$, where k' is either equal to k, or equal to a totally random session key. The scheme is $\mathsf{CCA1}$-secure if no polynomial adversary can distinguish these two cases with non-negligible advantage.

2.2 Identity-Based Encryption

An *Identity-Based Encryption* (IBE) scheme is a Public Key Cryptosystem where public keys can be arbitrary bitstrings, from which a trusted entity known as *Private Key Generator* (PKG) can extract the corresponding private keys. The main advantage of such Cryptosystems is that each user can have as public key some identifier ID that everybody knows (e.g. his/her e-mail address), thus avoiding the use of certificates.

Although a formal definition of IBE cryptosystems have been known for a while [19], the first fully functional proposal fitting all the requirements appeared only quite recently in [3] (see Appendix A).

Definition 2 (Identity-Based Encryption Scheme). *An Identity-Based Encryption scheme is a quadruple of* poly*-time algorithms* (Setup, Extract, Encrypt, Decrypt)*, where:*

- Setup *is a probabilistic algorithm used by the PKG to initialize the global parameters of the system. Given a security parameter* 1^λ*,* Setup *generates the system parameters* params *and a secret key* master-key*. Then, the PKG publishes* params *as the global public key and keeps* master-key *secret.*
- Extract *is a (possibly) probabilistic algorithm used by the PKG to derive private keys from arbitrary identifiers.* Extract *takes as input* params*, an identifier* $\mathrm{ID} \in \{0,1\}^*$ *and* master-key*, and returns the private key d capable of decrypting ciphertexts intended for the holder of the given identifier* ID*.*
- Encrypt *is a probabilistic algorithm used to securely send a message M to the user with identifier* ID *within the* IBE *system with global public key* params*.* Encrypt *takes as input* params*,* ID *and M and returns a ciphertext C.*
- Decrypt *is a deterministic algorithm used to recover the message M from a ciphertext C intended for a user with identifier* ID*.* Decrypt *takes as input* params*,* ID*, C and the private key d (corresponding to* ID*) and returns M.*

Clearly, these four algorithms should satisfy the standard consistency constraint: for all possible values of the global parameters params output by Setup, and for all identifiers $\mathrm{ID} \in \{0,1\}^*$, if d is the private key extracted from ID using master-key then for all message M it must be that:

$$\mathsf{Decrypt}(\mathtt{params}, \mathrm{ID}, \mathsf{Encrypt}(\mathtt{params}, \mathrm{ID}, M), d) = M.$$

As before, we briefly define the CCA1-security of IBE's, even though the currently known IBE's support a stronger kind of CCA2-security. Upon seeing the public params, the adversary repeatedly perform (in any adaptively-chosen order) the following two steps: (1) learn the private key d corresponding to any identifier ID that it chooses, by means of an extraction query (let $\mathcal{R}$ be the final set of corrupted users); (2) ask any user to decrypt a given ciphertext C chosen by the adversary. Then the adversary selects some message M and some identifier $\mathrm{ID} \notin \mathcal{R}$, and gets back the value $\mathsf{Encrypt}(\mathtt{params}, \mathrm{ID}, M')$, where M' is either equal to M, or is equal to a totally random message. The scheme is CCA1-secure if no polynomial adversary can distinguish these two cases with non-negligible advantage.

2.3 Hierarchical Identity-Based Encryption

HIBE is a natural and very powerful extension of a regular Identity-Based Encryption. Intuitively, HIBE allows to organize the users into a tree hierarchy. Each user gets the secret key from its parent in the hierarchy (and all the users share a few global parameters). Now, anybody can encrypt message to any given user by only knowing *its position in the hierarchy*, specified as an *ID-tuple* (or *hierarchical* identifier), $\text{HID} \equiv (\text{ID}_1, \ldots, \text{ID}_t)$. Such hierarchical identifier designates a user located at level t, whose ancestors (starting from the parent up to the root) have hierarchical identifiers $(\text{ID}_1, \ldots, \text{ID}_{t-1})$, $(\text{ID}_1, \ldots, \text{ID}_{t-2})$, …, (ID_1), root.

Definition 3 (Hierarchical Identity-Based Encryption Scheme).
*A Hierarchical Identity-Based Encryption (*HIBE*) scheme is a five-tuple of probabilistic polynomial-time algorithms* (Root Setup, Lower-level Setup, Extract, Encrypt, Decrypt)*, where:*

– Root Setup *is run by* root *to start-up an instance of* HIBE. Root Setup *takes as input a security parameter* 1^λ*, and returns the global public key* `params` *to be made available to everybody, and the master secret key* `master-key` *to be known only by the* root.
– Lower-level Setup *takes as input an ID-tuple* $(\text{ID}_1, \ldots, \text{ID}_t)$ *(*$t > 0$*) and the corresponding secret key, and returns some local secret information which can be used in the* Extract *procedure below. Notice that the output cannot contain any parameter that needs to be made public, but only private information to be stored at the local node.*
– Extract *is run by a user with ID-tuple* $(\text{ID}_1, \ldots, \text{ID}_t)$ *(*$t = 0$ *corresponds to* root*) to compute, using* `params`*, its secret key, and maybe other local secret data output by* Lower-level Setup *when* $t > 0$*, the secret key for an immediate lower level child with ID-tuple of the form* $(\text{ID}_1, \ldots, \text{ID}_t, \text{ID}_{t+1})$.
– Encrypt *takes as input* `params`*, the recipient's ID-tuple* $(\text{ID}_1, \ldots, \text{ID}_t)$ *and a message* M*, and returns the ciphertext* C *intended for user* $(\text{ID}_1, \ldots, \text{ID}_t)$.
– Decrypt *is run by the user* $(\text{ID}_1, \ldots, \text{ID}_t)$ *to recover the plaintext* M *from the ciphertext* C*, given as input* `params`, $(\text{ID}_1, \ldots, \text{ID}_t)$, C *and the user's private key.*

As expected, the correctness property states that the user with hierarchical identifier $\text{HID} \equiv (\text{ID}_1, \ldots, \text{ID}_t)$ should always correctly recover messages encrypted for him/her. In fact, in any HIBE scheme all the ancestors of the given user can understand messages encrypted for this user, since they can derive the corresponding secret key by running a series of Extract operations. In fact, in specific schemes (such as [11]), the ancestor might perform such decryption even faster (indeed, we will use this property of [11] later to our advantage).

Finally, we briefly define the CCA1-security of HIBE's. Intuitively, it states that only the designated user $(\text{ID}_1, \ldots, \text{ID}_t)$ *and its ancestors* can decrypt messages sent to this user, while no other user of the system can. Upon seeing the public key `params`, the adversary repeatedly perform (in any adaptively-chosen order) the following two steps: (1) learn the private key d corresponding to any ID-tuple $(\text{ID}_1, \ldots, \text{ID}_t)$ that it chooses, by means of an extraction query (let $\mathcal{R}$ be the final set of corrupted users); (2) ask any user to decrypt a given ciphertext C chosen by the adversary. Then the adversary selects some message M and some ID-tuple $(\text{ID}_1, \ldots, \text{ID}_t)$ such that $(\text{ID}_1, \ldots, \text{ID}_i) \notin \mathcal{R}$ for $0 \leq i \leq t$ (so that no ancestor of this user is corrupted), and gets back the value $\mathsf{Encrypt}(\texttt{params}, (\text{ID}_1, \ldots, \text{ID}_t), M')$, where M' is either equal to M, or is equal to a totally random message. The scheme is CCA1-secure if no polynomial adversary can distinguish these two cases with non-negligible advantage.

3 The Subset-Cover Framework

In [17], the authors presented the *Subset-Cover Framework* as a formal environment within which one can define and analyze the security of revocation schemes. Briefly, the main idea of the framework is to define a family $\mathcal{S}$ of subsets of the universe $\mathcal{N}$ of users in the system, and to associate each subset with a key, which is made available exactly to all the users belonging to the given subset. When the Center wants to broadcast a message to all the subscribers but those in some set $\mathcal{R}$, it "covers" the set $\mathcal{N} \setminus \mathcal{R}$ of "privileged" users using subsets from the family $\mathcal{S}$ (i.e. the Center determines a partition of $\mathcal{N} \setminus \mathcal{R}$, where all the subsets are elements of $\mathcal{S}$), and then encrypts the session key used to masquerade the message with all the keys associated to the subsets in the found partition.

A revocation scheme within the Subset-Cover framework is fully specified by defining the particular Subset-Cover family $\mathcal{S}$ used, the cover-finding algorithm and the key assignment employed to deliver to each user the keys corresponding to all the sets the user belongs to. We remark that the key assignment method does not necessarily give each user all the needed keys *explicitly*, but may provide some succinct representation sufficient to efficiently derive all the needed keys.

As specific examples, the *Complete Subtree* (CS) method and the *Subset Difference* (SD) method were formalized and proven secure within the framework; recently, in [12] the *Layered Subset Difference* (LSD) method was introduced as an improvement on the SD method, that achieves a smaller storage requirement per user at the cost of a slightly increase in the length of each broadcast.

Although all the above methods were proposed for the symmetric setting, a general technique (proposed in [17]) can be used to transpose them and any Subset-Cover revocation scheme to the asymmetric setting. The basic idea of this method is to make the public keys associated to each subset in the family $\mathcal{S}$ available to all the (not necessarily trusted) parties interested in broadcasting information, in the form of a *Public Key File* (PKF).

The price paid for the full generality of this technique is a high inefficiency in term of storage required to maintain and distribute the PKF. However, for specific schemes, it might be possible to come up with public key cryptosystems that allows to compress the PKF to a reasonable size. For instance, it was already observed in [17] that the use of an Identity-Based Encryption (IBE) scheme (such as the one proposed in [3]) would be helpful for the CS method. A solution for the more interesting case of the SD method (or equivalently for the LSD scheme) was left as an open problem.

We answer the question in the affirmative, by showing that any Hierarchical Identity-Based Encryption (HIBE) scheme can be used to reduce the PKF to $O(1)$ size, while maintaining the same small storage for every user. As a warm-up, we first briefly describe the CS method (referring the interested reader to [17] for more details) and then we show how to take advantage of the properties of an IBE scheme to extend the CS method. Afterwards, we describe the SD method and its extension to the public key setting by means of any HIBE scheme. We also show that the same technique can be used for the LSD variant as well.

For each method, our emphasis will be on developing its characteristic key assignment to the users, since this is the main difficulty we will face. In other words, we will not discuss in any detail the algorithmic technicalities needed to find the subset cover for the set of privileged users, since these methods remain identical to the symmetric key setting.

A note on Key Indistinguishability. To prove the generic security of the Subset-Cover framework for a given key assignment in the symmetric setting, [17] introduced an intermediate notion of *key indistinguishability*. Intuitively, it stated that any secret key k_j corresponding to the subset S_j remains pseudorandom to the adversary, even if he/she learns all the secret information belonging to all the users outside of S_j. Obviously, such intermediate notion does not make sense in the public key setting, since secret keys are never pseudorandom in public key cryptography. Instead, we notice that the argument of [17] easily extends to the public key setting, provided the public key encryption corresponding the the set S_j remains "secure" (in this case, CCA1-secure) even when the adversary learns all the secret information belonging to all the users outside of S_j. We omit the obvious formalization of this claim.

4 Public Key Extension of the CS Method

The Original Scheme. In the CS scheme, the users are organized in a tree structure: for the sake of simplicity, let us assume that the total number N of users in the system is a power of 2 (i.e. $N = 2^t$, for some integer t), and let us associate each user to a leaf of the complete binary tree $\mathcal{T}$ of height t. The Subset-Cover family $\mathcal{S}$ is then set to be the collection of all the complete subtrees of $\mathcal{T}$. More precisely, if v_j is a node in $\mathcal{T}$, the generic $S_j \in \mathcal{S}$ is the set of all the leaves of the complete subtree of $\mathcal{T}$ rooted at v_j (thus, in this case $|\mathcal{S}| = 2N - 1$).

To associate a key to each element of $\mathcal{S}$, the Center simply assigns, during an initialization step, a random number $\mathcal{L}_j$ to each node v_j in $\mathcal{T}$, and then $\mathcal{L}_j$ is used to perform all the encryption/decryption operations relative to the subset S_j. Furthermore, since each user needs to know the keys corresponding to all the subsets he/she belongs to, during the subscription step the Center gives the subscriber all the keys $\mathcal{L}_j$ relative to each node v_j in the path from the root down to the leaf representing the subscriber.

Notice that also the Center needs to keep track of all these keys: to limit the memory usage, a solution could be to use a pseudo-random function to derive all the $2N - 1$ keys from some fixed, short seed.

As for the efficiency of the scheme, we notice that the storage requirement on each subscriber is just $O(\log N)$, with a transmission rate (i.e. the length of the broadcast message) of $r \log \frac{N}{r}$, due to the fact that the cover algorithm needs a logarithmic number of subtrees to exclude each of the r revoked users in $\mathcal{R}$ (see [17] for more details).

Extension to the Public Key Setting. As mentioned above, a direct transposition of the CS method to the asymmetric setting yields a total number of $2N - 1$ public keys, explicitly stored in the PKF. To overcome the inefficiency of this approach we have to employ a scheme that allows an implicit and compact representation of the PKF from which to easily extract the needed information: the functionalities of any Identity-Based Encryption scheme come handy in this situation, yielding the efficient solution described below.

As a preliminary step, a fixed mapping is introduced to assign an identifier $\mathrm{ID}(S_j)$ to each subset S_j of the family $\mathcal{S}$. For example, a simple mapping could be to label each edge in the complete binary tree $\mathcal{T}$ with 0 or 1 (depending on whether the edge connects the node with its right or left child), and then assign to the subset S_j rooted at v_j the bitstring obtained reading off all the labels in the path from the root down to v_j.

Afterwards, the Center runs the Setup algorithm of an IBE scheme to create an instance of the system in which it will play the role of the *Private Key Generator* (PKG). Then, the Center publishes the parameters of the system params and the description of the mapping used to assign an identifier to each subset: these two pieces of data constitute the PKF, and requires $O(1)$ space.

For each subset $S_j \in \mathcal{S}$, the Center generates the corresponding private key $\mathcal{L}_j^{\mathrm{PRI}}$ as:

$$\mathcal{L}_j^{\mathrm{PRI}} \leftarrow \mathsf{Extract}(\mathtt{params}, \mathrm{ID}(S_j), \mathtt{master\text{-}key}).$$

At this point, the Center can distribute to each subscriber the private data necessary to decrypt the broadcast, as in the original, symmetric scheme. Moreover, whenever a (not necessarily trusted) party wants to broadcast a message, it can encrypt the session key k used to protect the broadcast under the public keys $\mathcal{L}_{i_j}^{\mathrm{PUB}} = \mathrm{ID}(S_{i_j})$ relative to all the subsets that make up the cover of the chosen set of privileged users. To this aim, this party only needs to know the parameters of the IBE system params and the description of the mapping $\mathrm{ID}(\cdot)$, and then it can compute:

$$\mathcal{C}_j \leftarrow \mathsf{Encrypt}(\texttt{params}, \mathrm{ID}(S_{i_j}), k)$$

for all the subset S_{i_j} in the cover.

Security. The formal CCA1-security of the scheme follows almost immediately from the powerful security definition of IBE. Indeed, when revoking some set $\mathcal{R}$ of users, the adversary does not learn any of the secret keys used for transmitting the message to the remaining users $\mathcal{N} \backslash \mathcal{R}$ (since only sets disjoint from $\mathcal{R}$ are used in the cover), so the CCA1-security of broadcast encryption immediately follows by a simple hybrid argument over the sets covering $\mathcal{N} \backslash \mathcal{R}$.

A Concrete Instantiation. Finally, if we apply the above idea in conjunction with the specific IBE scheme proposed in [3] (see Appendix A), the public key extension matches the original variant in all the efficiency parameters; more precisely, the storage requirement on each user is still $O(\log N)$ and the transmission rate is $r \log \frac{N}{r}$, where $r = |\mathcal{R}|$.

5 Public Key Extension of the SD Method

To improve the transmission rate, the SD scheme uses a more sophisticated Subset-Cover family $\mathcal{S}$: each user will belong to more subsets, thus allowing for greater freedom (and hence higher efficiency) in the choice of the cover. On the flip side, this will create a problem of compressing the user's storage which will need to be addressed.

As before, the users are associated to the leaves of the complete binary tree $\mathcal{T}$, but the generic subset S_{ij} is now defined in term of two nodes $v_i, v_j \in \mathcal{T}$ (with v_i ancestor of v_j), which we will call respectively *primary root* and *secondary root* of S_{ij}. Specifically, each subset S_{ij} consists of all the leaves of the subtree rooted at v_i except those in the subtree rooted at v_j.[1]

Due to the large number of subsets that contain a given user, it is no longer possible to employ an information-theoretic key assignment, directly associating a random key to each element in the family $\mathcal{S}$ (as it was done in the CS method), because this would require each subscriber to store a huge amount of secret data: to overcome this problem, a more involved, computational technique is required.

The idea behind the solution proposed in [17] is to derive the set of actual keys $\{\mathcal{L}_{ij}\}$ from some set of "proto-keys" $\{\mathcal{P}_{ij}\}$ satisfying the following properties:

1. given the proto-key $\mathcal{P}_{ij}$ it is easy to derive the key $\mathcal{L}_{ij}$;
2. given the proto-key $\mathcal{P}_{il}$ it is easy to derive the proto-key $\mathcal{P}_{ij}$, for any node v_j descendent of node v_l;
3. it is computationally difficult to obtain any information about a proto-key $\mathcal{P}_{ij}$ without knowing the proto-key $\mathcal{P}_{il}$ for some ancestor v_l of v_j (and descendent of v_i).

In particular, the last property implies that given the knowledge of the key $\mathcal{L}_{ij}$ it is computationally difficult to recover the proto-key $\mathcal{P}_{ij}$.

Once we have defined a way to generate a family of proto-keys featuring the above properties (which we will call a "proto-key assignment"), it is possible to make available to each subscriber the $O(N)$ secret keys corresponding to all the subsets he/she belongs to, by giving him/her only $O(\log^2 N)$ proto-keys, as described below.

Let u be the leaf representing the user within the tree $\mathcal{T}$ and let $r_\mathcal{T}$ be the root of $\mathcal{T}$. Furthermore, let $r_\mathcal{T} \equiv u_0, u_1, \ldots, u_t \equiv u$ be all the ancestors of u on the path from $r_\mathcal{T}$ down to u, and denote by s_h the sibling of u_h, $h = 1, \ldots, t$.

[1] The denomination of the SD method is due to the fact that each subset S_{ij} can be expressed as the set-difference of the two subsets S_i and S_j as defined in the CS method: $S_{ij} = S_i \setminus S_j$.

By definition, the subtree difference sets S_{ij} containing u are precisely those whose primary root v_i is one of the u_h's and whose secondary root v_j is a descendent[2] of $s_{h'}$ for some $h' > h$.

For instance, among the subsets whose primary root is $r_{\mathcal{T}}$, the ones containing u are those whose secondary root v_j is a descendent of some s_h. Notice that, by the first property of the proto-keys assignment described above, to compute the key $\mathcal{L}_{r_{\mathcal{T}} v_j}$ corresponding to such subset, it is enough to know the proto-key $\mathcal{P}_{r_{\mathcal{T}} v_j}$, which in turn (for the second property) can be obtained from the proto-key $\mathcal{P}_{r_{\mathcal{T}} s_h}$; thus, by giving the user the $t = \log N$ proto-keys $\mathcal{P}_{r_{\mathcal{T}} s_1}, \ldots, \mathcal{P}_{r_{\mathcal{T}} s_t}$, he/she will be able to efficiently compute the keys relative to all the subsets $S_{r_{\mathcal{T}} v_j}$ he/she belongs to.

Repeating the same reasoning for all the $\log N$ ancestor u_h of u, we can conclude that $O(\log^2 N)$ proto-keys suffice to allow the user u to recover all the $O(N)$ relevant keys.

The Original Scheme. We now describe the key assignment for the SD method of [17] as a particular instance of the proto-key assignment described above.

In the initialization phase, the Center associates to each internal node v_i in $\mathcal{T}$ a random number Label$_i$, which can be thought as the proto-key $\mathcal{P}_{ii}$ for the improper subtree difference set S_{ii}. Then, to generate the proto-keys for all the subsets S_{ij}, a pseudo random generator $\mathcal{G} : \{0,1\}^n \longrightarrow \{0,1\}^{3n}$ is used, where n is the desired length of the keys $\mathcal{L}_{ij}$. For notational convenience, given an input x, we will denote with $\mathcal{G}_L(x)$ the n leftmost bits of $\mathcal{G}(x)$, with $\mathcal{G}_R(x)$ the n rightmost bits of $\mathcal{G}(x)$, and with $\mathcal{G}_M(x)$ the remaining n central bits of $\mathcal{G}(x)$.

Using the generator $\mathcal{G}$, we can express the relationship between a proto-key $\mathcal{P}_{ij}$ and the proto-key $\mathcal{P}_{il}$ (with v_l parent of v_j) as follows:[3]

$$\mathcal{P}_{ij} = \begin{cases} \mathcal{G}_L(\mathcal{P}_{il}) & \text{if } v_j \text{ is the left child of } v_l \\ \mathcal{G}_R(\mathcal{P}_{il}) & \text{if } v_j \text{ is the right child of } v_l \end{cases}$$

Furthermore, the key $\mathcal{L}_{ij}$ associated to the subset S_{ij} can be derived from the proto-key $\mathcal{P}_{ij}$ as $\mathcal{L}_{ij} = \mathcal{G}_M(\mathcal{P}_{ij})$.

By construction, the first two properties of the proto-key assignment are satisfied; as for the third one, the use of a pseudorandom generator guarantees the computational hardness of obtaining any information about a proto-key $\mathcal{P}_{ij}$ or a key $\mathcal{L}_{ij}$, without knowledge of any proto-key $\mathcal{P}_{il}$, for some v_l ancestor of v_j.

Notice that the Center can avoid to store all the $N-1$ labels Label$_i$ by reusing the technique of the generator $\mathcal{G}$. Namely, the Center associates to the root $r_{\mathcal{T}}$ of the tree $\mathcal{T}$ a random seed s of length n; to generate each Label$_i$, it repeatedly applies the generator $\mathcal{G}$ taking, at each edge on the path going from the root down to the node v_i, the left part $\mathcal{G}_L$ or right part $\mathcal{G}_R$ depending on the direction of the edge, and finally applying $\mathcal{G}_M$ once it gets to the node v_i.

As already observed, the use of a proto-key assignment allows to cut the storage requirement on the subscribers down to $O(\log^2 N)$. More interestingly, since in [17] the authors showed how to cover any privileged set excluding r revoked users using only $2r-1$ subsets, the SD scheme enjoys an $O(r)$ transmission rate, thus being the only known broadcast encryption scheme supporting any number of revocations at the cost of a proportional increase in the length of the ciphertext (and independent of the total number of users).

Extension to the Public Key Setting. To extend the SD scheme to the asymmetric scenario, one would like to generalize the basic idea used for the case of the CS method:

[2] For our purposes, a node v will be considered among its own descendents.

[3] In [17], the authors refer to what we call here the "proto-key" $\mathcal{P}_{ij}$ as Label$_i$.

namely, define an ID mapping for all the subsets $S_{ij} \in \mathcal{S}$ and then employ an IBE scheme to extract all the relevant private keys. However, as already observed, to avoid an explosion of the user's storage, it is necessary to use a scheme satisfying the characteristic properties of a "proto-key assignment", whereas ordinary IBE schemes do not seem to support the crucial property, since this requires the capability of deriving "children" proto-keys from a given proto-key. Luckily, the more powerful notion of general Hierarchical Identity-Based Encryption (such as the one recently proposed in [11]), offers all the functionalities needed, leading to the solution described below.

First, to define a mapping $\mathrm{HID}(\cdot)$ assigning a hierarchical identifier to each set S_{ij} of the family $\mathcal{S}$, we will reuse the $\mathrm{ID}(\cdot)$ mapping introduced in the public key extension of the CS method, which associates to each node in the tree $\mathcal{T}$ a bitstring of 0's and 1's, depending on its position within $\mathcal{T}$.

Preliminarily, we extend the $\mathrm{ID}(\cdot)$ mapping to the improper subsets of the form S_{ii}, letting $\mathrm{ID}(S_{ii}) = \mathrm{ID}(v_i)$. Next, we notice that if v_i is an ancestor of v_j and we think of $\mathrm{ID}(v_i)$ and $\mathrm{ID}(v_j)$ as hierarchical sequences of one-digit identifiers (rather than as unique, monolithic IDs), then $\mathrm{ID}(v_i)$ will be a prefix of $\mathrm{ID}(v_j)$. So let us denote with $\mathrm{ID}(v_j)\backslash\mathrm{ID}(v_i)$ the *hierarchical* identifier made up by the sequence of single-bit identifiers in the suffix of $\mathrm{ID}(v_j)$ coming right after the prefix $\mathrm{ID}(v_i)$.

Now we can define the $\mathrm{HID}(\cdot)$ mapping on all the elements of $\mathcal{S}$ as follows:

$$\mathrm{HID}(S_{ij}) = (\mathrm{ID}(S_{ii}), [\mathrm{ID}(v_j)\backslash\mathrm{ID}(v_i)], 2)$$

where the operator "," is used to highlight the juxtaposition of hierarchical identifiers. Notice, the depth of this identifier is two plus the depth of v_i relative to v_j in our tree, and the the symbol 2 is used as terminator (we will see why soon).

Once the $\mathrm{HID}(\cdot)$ mapping has been specified, to complete the initialization phase, the Center runs the Setup algorithm of a HIBE scheme and publishes params and a description of the mapping $\mathrm{HID}(\cdot)$ as the Public Key File. Besides, the distribution of the secret decryption data to the subscribers will be carried out as another instantiation of the proto-keys assignment, as described below.

The key $\mathcal{L}_{ij}^{\mathrm{PRI}}$ relative to a given subset S_{ij} will be the private key extracted from the public key $\mathcal{L}_{ij}^{\mathrm{PUB}} = \mathrm{HID}(S_{ij})$. As described in Section 2.3, to extract the private key $\mathcal{L}_{ij}^{\mathrm{PRI}}$ for the hierarchical identifier $\mathrm{HID}(S_{ij}) = (\mathrm{ID}(S_{ii}), [\mathrm{ID}(v_j)\backslash\mathrm{ID}(v_i)], 2)$, it is necessary to know the private key $\mathcal{P}_{ij}$ of the local PKG with hierarchical identifier $(\mathrm{ID}(S_{ii}), [\mathrm{ID}(v_j)\backslash\mathrm{ID}(v_i)])$, (or the private key of any other ancestor of $\mathrm{HID}(S_{ij})$ lying higher in the HIBE hierarchy). Such key $\mathcal{P}_{ij}$ is defined to be the proto-key associated to S_{ij}; formally:[4]

$$\begin{aligned}
\mathcal{L}_{ij}^{\mathrm{PRI}} &\leftarrow \mathsf{Extract}(\mathtt{params}, (\mathrm{ID}(S_{ii}), [\mathrm{ID}(v_j)\backslash\mathrm{ID}(v_i)], 2), \mathcal{P}_{ij}) \\
\mathcal{P}_{ij} &\leftarrow \mathsf{Extract}(\mathtt{params}, (\mathrm{ID}(S_{ii}), [\mathrm{ID}(v_j)\backslash\mathrm{ID}(v_i)]), \mathcal{P}_{il}) \\
\mathcal{P}_{ii} &\leftarrow \mathsf{Extract}(\mathtt{params}, (\mathrm{ID}(S_{ii})), \mathtt{master\text{-}key})
\end{aligned}$$

where v_l is the parent of v_j, and `master-key` is the master key output by the Setup algorithm and known only to the root PKG, role that in our setting is played by the Center.

From the above definitions, it is clear that the first two properties of a proto-key assignment are fulfilled; on the other hand, the validity of the third one hinges upon the security

[4] We remark that the values of keys and proto-keys are not uniquely defined by these probabilistic assignments. In particular, deriving the value of the "same" key twice from some of its ancestors will likely result in different keys. However, any value we get is equally functional by the definition of HIBE.

of the HIBE scheme, that ensures the computational difficulty of obtaining a private key for any identifier without knowing the private key of a local PKG lying higher in the hierarchy of the system.

Direct consequence of the application of the proto-key assignment to the public key extension is that the storage requirement on each subscriber is still $O(\log^2 N)$. On the other hand, the cover finding algorithm characteristic of the SD method ensures that $2r-1$ ciphertexts will suffice in the worst case to broadcast the session key to all the privileged users in the system.

Security. The formal CCA1-security of the scheme again follows almost immediately from the powerful security definition of HIBE. Indeed, when revoking some set $\mathcal{R}$ of users, none of the proto-keys the adversary learns is an ancestor of any of the hierarchical identifies corresponding to the sets covering $\mathcal{N}\backslash\mathcal{R}$. This property is fairly easy to verify, and a simple hybrid argument will again complete the security proof. We remark that only CCA1-security is achieved by the SD (as well as the CS) scheme(s), since the adversary is disallowed to ask the decryption oracle after the challenge is obtained. And, indeed, it is easy to see that one cannot achieve CCA2-security by following the Subset-Cover framework, since each user can decrypt only one of several independent encryptions of the message.

A Concrete Instantiation. We now consider how an actual implementation of our public key extension would perform in the practice. Since the only known implementation of a fully functional HIBE is the one recently proposed in [11], we discuss its efficiency below (see also Appendix A).

One interesting characteristic of the HIBE of [11] is that a ciphertext encrypted for a given user in the system can be easily recovered by any of its ancestor — actually, the decryption process gets more and more efficient as we go higher in the hierarchy! As a consequence (and unlike the symmetric key setting), instead of deriving the private key $\mathcal{L}_{ij}^{\text{Pri}}$ required to decrypt the ciphertext from its "ancestor" proto-key $\mathcal{P}_{il}$, the user can directly obtain the message broadcast using $\mathcal{P}_{il}$ itself, thus saving up to $O(\log N)$ factor in the decryption time.

On the flip side, the specific HIBE of [11] yields ciphertexts whose length is proportional to the nesting depth of the hierarchical identifier to which the encrypted message is being sent: it follows that the transmission rate of such a concrete instantiation of our public key extension would be $O(r \log N)$, due to the fact that the hierarchical identifier $\text{HID}(S_{ij})$ can have nesting depth proportional to the height $t = \log N$ of the tree $\mathcal{T}$.

Therefore, when used in conjunction with the HIBE of [11], the asymmetric variation of the SD scheme proposed above leads to the same decryption time and transmission rate of the public key extension of the CS method, while imposing a greater storage requirement on each single user. Nevertheless, we feel that our technique gives an interesting solution to the problem of obtaining a fixed Public Key File size, when generalizing the SD method to the asymmetric setting: besides, if a more efficient implementation of HIBE should become available, the parameters of our scheme would automatically improve, possibly reaching the efficiency of the SD method for the symmetric scenario.

6 Public Key Extension of the LSD Method

The Original Scheme. Recently, an improvement to the SD method, known as the *Layered Subset Difference* (LSD) method, was proposed in [12]. In its basic form, this method reduces the amount of secret data that each subscriber needs to store from $O(\log^2 N)$ to $O(\log^{3/2} N)$, at the cost of doubling the maximum size of the cover. The authors also presented a generalization of the basic scheme that achieves a storage requirement of $O(\log^{1+\epsilon} N)$, for any

$\epsilon > 0$, while increasing the length of the broadcast by a factor of $1/\epsilon$, which still yields a transmission rate of $O(r)$, for fixed values of ϵ (see [12] for the details of the construction).

The main idea behind the LSD scheme is to reduce the size of the family $\mathcal{S}$ by only considering a subcollection $\mathcal{S}'$ of *useful* subsets. The key observation to reach this goal is that any subtree difference set S_{ij} can be rewritten as the disjoint union $S_{ik} \cup S_{kj}$, for any node v_k lying in the path from v_i to v_j.

To define the sub-collection $\mathcal{S}'$, consecutive levels of the tree $\mathcal{T}$ are grouped into *layers*, with certain subsets of $\mathcal{S}$ defined as *local* and *special*. In particular, local subsets are those whose primary and secondary roots both lie within the same layer, while *special* subset are those having as their primary root a node lying exactly on the boundary between two adjacent layers. The sub-collection $\mathcal{S}'$ consists precisely of all the local and special subsets of $\mathcal{S}$. In this way, the number of proto-keys that each user needs in order to decrypt each broadcast can be reduced by a factor of 2, while the Center can preserve the functionalities of the system by at most doubling the size of the cover. This is because any subset $S_{ij} \in \mathcal{S}$ can be obtained as the union of a local subset and a special subset in $\mathcal{S}'$.

Extension to the Public Key Setting. Since the LSD scheme only differs from the SD method of [17] for the use of a smaller subcollection $\mathcal{S}'$ of the Subset-Cover family $\mathcal{S}$, we can extend it to the asymmetric setting applying exactly the same idea used to generalize the SD method to the public key scenario. Indeed, any HIBE scheme can be employed to distribute the necessary proto-keys to the users of the system, according to the same label-distribution strategy defined for the original LSD scheme in its conventional symmetric mode.

A Concrete Instantiation. As for the efficiency parameters of such public key extension, we can repeat the same discussion outlined for the SD scheme: namely, if we use the HIBE proposed in implementation of a fully functional HIBE scheme), the public key extension maintains the same storage requirement as the original, symmetric LSD scheme, whereas the transmission rate deteriorates by a factor of $\log N$. Again, should a more efficient HIBE scheme be proposed, our solution would consequently improve, approaching the performance of the conventional LSD scheme.

6.1 Inclusion-Exclusion Trees

Inclusion-Exclusion Trees. In [12], the authors also considered an alternative approach to the problem of specifying the set of revoked users $\mathcal{R}$. Such technique is based on the use of *Inclusion-Exclusion Trees* (IE-Trees), which offer a convenient way of describing a large set of revoked users with relative few nested inclusion and exclusion conditions on the nodes of the tree $\mathcal{T}$. The advantage of using an IE-Tree it that it is possible to derive a cover whose size is proportional to the number of conditions specified by the IE-Tree itself, rather than to the number of revoked users.

Without going in the details of this approach (for which we refer the reader to [12]), we notice here that our extension to the Public Key setting can be coupled with the use of IE-Trees in the case of both the SD scheme and the LSD scheme, since once a cover of the set of privileged users has been obtained, both the encryption and the decryption steps can be performed making use of our HIBE-based technique presented above.

Acknowledgments

We would like to thank Moni Naor for valuable discussions.

References

1. J.H. An, Y. Dodis, and T. Rabin. On the Security of Joint Signature and Encryption. In *Advances in Cryptology - EuroCrypt '02*, pages 83–107, Berlin, 2002. Springer-Verlag. Lecture Notes in Computer Science Volume 2332.
2. M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A Concrete Security Treatment of Symmetric Encryption: Analysis of the DES Modes of Operation. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science - FOCS '97*, pages 394–403, 1997.
3. D. Boneh and M. Frankling. Identity-Based Encryption from the Weil Pairing. In *Advances in Cryptology - Crypto '01*, Berlin, 2001. Springer-Verlag. Lecture Notes in Computer Science Volume 2139.
4. R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. Multicast Security: A Taxonomy and some Efficient Constructions. In *Proceedings of IEEE INFOCOM '99*, volume 2, pages 708–716, 1999.
5. R. Canetti, T. Malkin, and K. Nissim. Efficient Communication-Storage Tradeoffs for Multicast Encryption. In *Theory and Application of Cryptographic Techniques*, pages 459–474, 1999.
6. Y. Dodis and N. Fazio. Public Key Broadcast Encryption Secure against Adaptive Chosen Ciphertext Attack. To appear in PKC '03, 2002.
7. D. Dolev, C. Dwork, and M. Naor. Nonmalleable Criptography. *SIAM Journal on Discrete Mathematics*, 30(2):391–437, 2000.
8. A. Fiat and M. Naor. Broadcast Encryption. In *Advances in Cryptology - Crypto '93*, pages 480–491, Berlin, 1993. Springer-Verlag. Lecture Notes in Computer Science Volume 773.
9. E. Gafni, J. Staddon, and Y. L. Yin. Efficient Methods for Integrating Traceability and Broadcast Encryption. In *Advances in Cryptology - Crypto '99*, pages 372–387, Berlin, 1999. Springer-Verlag. Lecture Notes in Computer Science Volume 1666.
10. A Garay, J. Staddon, and A. Wool. Long-Lived Broadcast Encryption. In *Advances in Cryptology - Crypto 2000*, pages 333–352, Berlin, 2000. Springer-Verlag. Lecture Notes in Computer Science Volume 1880.
11. C. Gentry and A. Silverberg. Hierarchical ID-Based Cryptography. To appear in AsiaCrypt '02. Available at `http://eprint.iacr.org/2002/056/`, 2002.
12. D. Halevy and A. Shamir. The LSD Broadcast Encryption Scheme. In *Advances in Cryptology - Crypto '02*, pages 47–60, Berlin, 2002. Springer-Verlag. Lecture Notes in Computer Science Volume 2442.
13. J. Horwitz and B. Lynn. Toward Hierarchical Identity-Based Encryption. In *Advances in Cryptology - EuroCrypt '02*, pages 466–481, Berlin, 2002. Springer-Verlag. Lecture Notes in Computer Science Volume 2332.
14. R. Kumar, S. Rajagopalan, and A. Sahai. Coding Constructions for Blacklisting Problems without Computational Assumptions. In *Advances in Cryptology - Crypto '99*, pages 609–623, Berlin, 1999. Springer-Verlag. Lecture Notes in Computer Science Volume 1666.
15. M. Luby and J. Staddon. Combinatorial Bounds for Broadcast Encryption. In *Advances in Cryptology - EuroCrypt '98*, pages 512–526, Berlin, 1998. Springer-Verlag. Lecture Notes in Computer Science Volume 1403.
16. D.A. McGrew and A.T. Sherman. Key Establishment in Large Dynamic Groups Using One-Way Function Trees. Manuscript, 1998.
17. D. Naor, M. Naor, and J. Lotspiech. Revocation and Tracing Schemes for Stateless Receivers. In *Advances in Cryptology - Crypto '01*, pages 41–62, Berlin, 2001. Springer-Verlag. Lecture Notes in Computer Science Volume 2139.
18. M. Naor and B. Pinkas. Efficient Trace and Revoke Schemes. In *Financial Cryptography - FC 2000*, pages 1–20, Berlin, 2000. Springer-Verlag. Lecture Notes in Computer Science Volume 1962.
19. A. Shamir. Identity Based Cryptosystems and Signatures Schemes. In *Advances in Cryptology - Crypto '84*, pages 47–53, Berlin, 1984. Springer-Verlag. Lecture Notes in Computer Science Volume 196.

20. V. Shoup. A Proposal for an ISO Standard for Public-Key Encryption. Manuscript, 2001.
21. W.G. Tzeng and Z.J. Tzeng. A Public-Key Traitor Tracing Scheme with Revocation Using Dynamics Shares. In *Public Key Cryptography - PKC '01*, pages 207–224, Berlin, 2001. Springer-Verlag. Lecture Notes in Computer Science Volume 1992.
22. D. Wallner, E. Harder, and R. Agee. Key Management for Multicast: Issues and Architectures. Available at `ftp://ftp.ietf.org/rfc/rfc2627.txt`, 1997.
23. C.K. Wong, M. Gouda, and S. Lam. Secure Group Communications Using Key Graphs. In *Proceedings of the ACM SIGCOMM '98*, 1998.

A Currently Best IBE/HIBE Schemes

We briefly describe the currently best IBE scheme of [3] and the HIBE [11]. We will only describe the "basic" chosen plaintext (CPA) secure versions of these schemes, since both schemes utilize random oracles, and amplifying the security from CPA to CCA1/CCA2 can be done by a variety of standard means in the random oracle model (see [3, 11] for the details). Also, since the HIBE of [11] is a generalization of the IBE of [3], we first describe their common features.

Common Features.. Let $\mathbb{G}_1, \mathbb{G}_2$ be two cyclic groups of a large prime order q, where $\mathbb{G}_1$ is represented additively, and $\mathbb{G}_2$ multiplicatively. We assume the existence of a symmetric *bilinear mapping* $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$. Namely, for any $P, Q \in \mathbb{G}_1$, $a, b \in \mathbb{Z}_q$, we have:

$$\hat{e}(aP, bQ) = \hat{e}(bP, aQ) = \hat{e}(P, Q)^{ab} = \hat{e}(Q, P)^{ab} \tag{1}$$

We assume also the existence of the parameter generation algorithm $\mathcal{I}$ which, on input 1^λ, outputs a prime q, the description of $\mathbb{G}_1, \mathbb{G}_2$ of order q and a bilinear map $\hat{e}$, so that $\hat{e}$ is polynomial-time computable in λ. We mention that the security of both schemes below is based on the *Bilinear Diffie-Hellman (*BDH*) assumption*: for random $P \in \mathbb{G}_1$, $a, b, c \in \mathbb{Z}_q$, it is computationally hard to compute $\hat{e}(P, P)^{abc} \in \mathbb{G}_2$ when given only P, aP, bP, cP.

IBE of [3]. We follow the same notation as the the one we will later use for the HIBE of [11].

- Setup. Run $\mathcal{I}(1^\lambda)$ to get $\mathbb{G}_1, \mathbb{G}_2, \hat{e}$, pick a random $s_0 \in \mathbb{Z}_q$, $P_0 \in \mathbb{G}_1$, set $Q_0 = s_0 P_0$, and output `params` $= (\mathbb{G}_1, \mathbb{G}_2, \hat{e}, P_0, Q_0, H_1, H_2)$, `master-key` $= s_0$. Here $H_1 : \{0,1\}^* \to \mathbb{G}_1$, $H_2 : \mathbb{G}_2 \to \{0,1\}^n$ are cryptographic hash functions, modeled as random oracles (i.e., they output a truly random string on every input), and n is the length of the messages encrypted.
- Extract. Set the secret key of user ID to $S_1 = s_0 P_1$, where $P_1 = H_1(\mathrm{ID})$ is a random point in $\mathbb{G}_1$ derived from ID by means of a random oracle.
- Encrypt. To encrypt a message $M \in \{0,1\}^n$ for user ID using public value Q_0, compute $P_1 = H_1(\mathrm{ID}) \in \mathbb{G}_1$, choose a random $r \in \mathbb{Z}_q$, set $g = \hat{e}(Q_0,\ rP_1) \in \mathbb{G}_2$ and return $C = [rP_0,\ M \oplus H_2(g)]$.
- Decrypt. To decrypt $C = [U_0, V]$ using S_1 and Q_0, set $f_0 = \hat{e}(U_0, S_1)$ and output $V \oplus H_2(f_0)$.

To see the correctness of the decryption, notice that:

$$f_0 \ = \ \hat{e}(U_0, S_1) \ = \ \hat{e}(rP_0, s_0P_1) \ \overset{(1)}{=} \ \hat{e}(s_0P_0, rP_1) \ = \ \hat{e}(Q_0, rP_1) = g.$$

HIBE of [11]. We will see that the IBE scheme above is a special case of the scheme below when depth $t = 1$.

- Root Setup. Same as Setup for IBE. Namely, run $\mathcal{I}(1^\lambda)$ to get $\mathbb{G}_1, G_2, \hat{e}$, pick a random $s_0 \in \mathbb{Z}_q$, $P_0 \in \mathbb{G}_1$, set $Q_0 = s_0 P_0$, and output `params` $= (\mathbb{G}_1, \mathbb{G}_2, \hat{e}, P_0, Q_0, H_1, H_2)$, `master-key` $= s_0$.

- Lower-level Setup. Each user at level $t \geq 1$ picks a random local secret $s_t \in \mathbb{Z}_q$ (recall, root has s_0) and keeps it secret.
- Extract. Every user $(\mathrm{ID}_1, \ldots, \mathrm{ID}_t)$ at level $t \geq 0$ will have a secret point $S_t \in \mathbb{G}_1$ (see below; we assume that the root has $S_0 = 0_{\mathbb{G}_1}$), and $(t-1)$ "translation points" $Q_1 \ldots Q_{t-1} \in \mathbb{G}_1$ (notice, Q_0 is in the public key). Recursively, to assign the secret key to its child ID_{t+1}, the parent $(\mathrm{ID}_1, \ldots, \mathrm{ID}_t)$ computes $P_{t+1} = H_1(\mathrm{ID}_1 \ldots \mathrm{ID}_{t+1}) \in \mathbb{G}_1$, picks a random $s_t \in \mathbb{Z}_q$, sets the child's secret point $S_{t+1} = S_t + s_t P_{t+1}$, the child's final translation point $Q_t = s_t P_0$, and sends to the child the values S_{t+1}, Q_t together with its own $t-1$ translation points $Q_1 \ldots Q_{t-1}$. Unwrapping the notation, the child's secret key is $(S_{t+1} = \sum_{i=1}^{t+1} s_{i-1} P_i,\ Q_1 = s_1 P_0, \ldots,\ Q_t = s_t P_0)$.
- Encrypt. To encrypt a message $M \in \{0,1\}^n$ for $(\mathrm{ID}_1, \ldots, \mathrm{ID}_t)$ using public value Q_0, compute $P_i = H_1(\mathrm{ID}_1 \ldots \mathrm{ID}_i) \in \mathbb{G}_1$ for all $1 \leq i \leq t$, choose a random $r \in \mathbb{Z}_q$, set $g = \hat{e}(Q_0,\ rP_1) \in \mathbb{G}_2$ and return:
$$C = [rP_0,\ M \oplus H_2(g),\ rP_2, \ldots,\ rP_t]$$
Intuitively, the first two components correspond to the IBE encryption we described earlier for the top-level user (ID_1). Unfortunately, user $(\mathrm{ID}_1, \ldots, \mathrm{ID}_t)$ cannot quite decrypt it using its "translated" secret point S_{t+1}, so additional values $rP_2, \ldots, rP_t$ are given. Combining them with secret translation points $Q_1 \ldots Q_{t-1}$, the message M is recovered. This is described below.
- Decrypt. To decrypt $C = [U_0, V, U_2, \ldots, U_t]$ using S_t and $Q_1 \ldots Q_{t-1}$, set $f_0 = \hat{e}(U_0, S_t)$, $f_i = \hat{e}(Q_{i-1}, U_i)$ for $2 \leq i \leq t$ and output $M = V \oplus H_2(f_0/(f_2 \ldots f_t))$.

To see the correctness of the decryption, notice that:
$$\begin{aligned} f_0 &= \hat{e}(U_0, S_t) = \hat{e}(rP_0,\ \sum_{i=1}^{t} s_{i-1} P_i) = \prod_{i=1}^{t} \hat{e}(rP_0,\ s_{i-1} P_i) \\ &\overset{(1)}{=} \prod_{i=1}^{t} \hat{e}(s_{i-1} P_0,\ rP_i) f_2 \cdots f_t \end{aligned}$$

Finally, we remark on the specific feature of the above scheme. The ciphertext for the user at level t literally contains the shorter ciphertext for every ancestor of the user. Thus, it is more efficient to decrypt for the ancestor than for the user itself!