ID-Based Encryption for Complex Hierarchies with Applications to Forward Security and Broadcast Encryption

Danfeng Yao* Nelly Fazio† Yevgeniy Dodis † Anna Lysyanskaya*

Abstract

A forward-secure encryption scheme protects secret keys from exposure by evolving the keys with time. Forward security has several unique requirements in hierarchical identity-based encryption (HIBE) scheme: (1) users join dynamically; (2) encryption is joining-time-oblivious; (3) users evolve secret keys autonomously.

We present a scalable forward-secure HIBE (fs-HIBE) scheme satisfying the above properties. We also show how our fs-HIBE scheme can be used to construct a forward-secure public-key broadcast encryption scheme, which protects the secrecy of prior transmissions in the broadcast encryption setting. We further generalize fs-HIBE into a collusion-resistant multiple hierarchical ID-based encryption scheme, which can be used for secure communications with entities having multiple roles in role-based access control. The security of our schemes is based on the bilinear Diffie-Hellman assumption in the random oracle model.

1 Introduction

The idea of an identity-based encryption (IBE) scheme is that an arbitrary string can serve as a public key. The main advantage of this approach is to largely reduce the need for public key certificates and certificate authorities, because a public key is associated with identity information such as a user's email address. A first scheme for identity-based encryption (BF-IBE) was based on the bilinear Diffie-Hellman assumption in the random oracle model by Boneh and Franklin [11]. In IBE schemes private key generator (PKG) is responsible for generating private keys for all users, and therefore is a performance bottleneck for organizations with large number of users. Hierarchical identity-based encryption (HIBE) schemes [9, 22, 26] were proposed to alleviate the workload of a root PKG by delegating private key generation and identity authentication to lower-level PKGs. In a HIBE scheme, a root PKG needs only to generate private keys for domain-level PKGs, who in turn generate private keys for users in their domains in the next level. The organization of PKGs and users forms a hierarchy that is rooted by the root PKG. To encrypt a message, Alice needs to obtain the public parameters of Bob's root PKG, and the ID for Bob and for those domain-level PKGs that are on the path from the root to Bob; there are no lower-level parameters. Gentry and Silverberg [22] extended BF-IBE scheme and presented a fully scalable hierarchical identity-based encryption (GS-HIBE) scheme. Later, a HIBE construction with a weaker notion of security was given by Boneh and Boyen [9]. Most recently, new IBE and HIBE constructions that can be proved to have the full security without the random oracle model [8, 36] were given.

*Department of Computer Science, Brown University, Providence, RI 02912, {dyao, anna}@cs.brown.edu.

†Department of Computer Science, Courant Institute of Mathematical Sciences, New York University, New York, NY 10012, {fazio, dodis}@cs.nyu.edu

Due to the inherent key-escrow property[1], the standard notion of HIBE security crucially depends on secret keys remaining secret. Key exposure is a realistic threat over the lifetime of such a scheme. To mitigate the damage caused by the exposure of secret key information in HIBE, one way is to construct a forward-secure hierarchical identity-based encryption (fs-HIBE) scheme that allows each user in the hierarchy to refresh his or her private keys periodically while keeping the public key the same. A forward-secure public-key encryption scheme has recently been presented by Canetti, Halevi and Katz [13]. But surprisingly, a practical fs-HIBE scheme has several unique requirements that cannot be achieved by trivial combinations of the existing fs-PKE schemes [13, 27] and HIBE scheme [9, 22].

Apart from being interesting on its own, fs-HIBE is a useful tool that lends itself to several applications. One such application is the implementation of forward secrecy for public-key broadcast encryption. While forward secrecy is an important requirement in any context, it is especially needed for broadcast encryption [7, 18, 20, 29, 37]. This is because by design an adversary can freely listen to any broadcast and store it. Then, should the adversary ever succeed in recovering *any* user's secret key, she will manage to decrypt all past broadcasts that such user was authorized to receive *unless* we have forward secrecy.

Below, we discuss the notion of forward security for HIBE in more detail, and then explain why it cannot be trivially achieved by existing techniques such as a combination of fs-PKE [13] and HIBE [9, 22] schemes.

1.1 Forward Security

The central idea of forward secrecy is that the compromise of long-term keys does not compromise past session keys and therefore past communications. This notion was first proposed by Günther [21] and later by Diffie *et al.* [14] in key exchange protocols. The notion of non-interactive forward security was proposed by Anderson [3] in 1997 and later formalized by Bellare and Miner [4], who also gave a forward-secure signature scheme followed by a line of improvement [1, 30]. In this model, secret keys are updated at regular intervals throughout the lifetime of the system; furthermore, exposure of a secret key corresponding to a given interval does not enable an adversary to break the system (in the appropriate sense) for any prior time period. The model inherently cannot prevent the adversary from breaking the security of the system for any subsequent time period. Bellare and Yee [6] provided a comprehensive treatment of forward security in the context of private key based cryptographic primitives.

The first forward-secure public-key encryption (fs-PKE) scheme was given by Canetti, Halevi, and Katz [13] based on the Gentry-Silverberg HIBE [22] scheme. The fs-PKE scheme constructs a binary tree, in which a tree node corresponds to a time period and has a secret key. Children of a node w are labeled $w0$ and $w1$, respectively. Given the secrets corresponding to a prefix of a node representing time t, one can compute the secrets of time t. In order to make future keys computable from the current key, the secrets associated with a prefix of a future time are stored in the current key. After the key for the next time period is generated, the current decryption key is erased. The state-of-the-art fs-PKE scheme [13] is based on the decisional bilinear Diffie-Hellman assumption [11] in the standard model. Canetti, Halevi and Katz also gave a more efficient scheme in the random oracle model [13].

[1] A PKG knows the private keys of its child nodes.

1.2 Requirements of an fs-HIBE Scheme

Intuitively, forward security in a HIBE scheme implies that compromise of the current secret key of a user only leads to the compromise of the user and his descendants' subsequent communications. We will give a formal definition of security in Section 2.3. Our design of a forward-secure HIBE scheme also takes system properties such as scalability and efficiency into consideration. This is essential in the management of large scale distributed systems. Below, we define the requirements for a scalable forward-secure HIBE scheme.

- New users should be able to join the hierarchy and receive secret keys from their parent nodes *at any time*.
- Encryption is *joining-time-oblivious*, which means that the encryption does not require knowledge of when a user or any of his ancestors joined the hierarchy. The sender can encrypt the message as long as he knows the current time and the ID-tuple of the receiver, along with the public parameters of the system.
- The scheme should be forward-secure.
- Refreshing secret keys can be carried out *autonomously*, that is, users can refresh their secret keys on their own to avoid any communication overhead with any PKG.

Surprisingly, the design of an fs-HIBE scheme that fulfils the above system requirements turns out to be non-trivial, despite the fact that both HIBE [22] scheme and fs-PKE [13] scheme are known. Intuitive combinations of the two schemes fail to achieve all the desired system features. Next, we explain why this is the case.

1.3 Some Forward-Secure HIBE Attempts

In this section, we make three simple forward-secure HIBE constructions based on HIBE scheme [22] and fs-PKE scheme [13], and explain why these naive schemes do not satisfy the requirements of a practical fs-HIBE scheme.

1.3.1 Scheme I

Consider a scheme based on the HIBE [22] scheme. The user with a given ID tuple $(ID_1, \ldots, ID_h)$ maintains two sub-hierarchies (subtrees): the time subtree that evolves over time for forward security (as in fs-PKE [13]), and the ID subtree to which other nodes are added as children join the hierarchy. To encrypt a message for this user at time t, use the HIBE with identity $(ID_1, \ldots, ID_h, t)$. The user can decrypt this message using HIBE decryption, using the fact that he knows the key from the time subtree. The user's children are added to the hierarchy into the ID subtree.

The problem with this scheme is combining dynamic joins with forward security. Suppose a user never erases the secret key corresponding to the root of his ID subtree. Then should this key ever be exposed, the forward secrecy of his children is compromised. On the other hand, if this secret key is ever erased, then no nodes can be added as children of $(ID_1, \ldots, ID_h)$ in the hierarchy, and so this scheme will not support dynamic joins.

The lesson we learn from this failed scheme is that all keys must be evolved together.

1.3.2 Scheme II

Let us try to repair Scheme I by making sure that the key from which children's keys are derived is also evolving over time. In Scheme II, the public key of a user consists of alternating ID-tuples and time strings, which is referred to as an *ID-time-tuple*. The private key of a user serves three purposes: decryption, generating private keys for new children, and deriving future private keys of the user. The public key of a newly joined child is the parent's ID-time-tuple appended with the child's ID. That key is in turn used for generating keys for lower-level nodes further down the hierarchy. For example, if Alice joins Bob, the root, at time (*January, Week 1*) and Eve joins Alice at time (*January, Week 2*), Eve's public key is (Bob, *January, Week 1*, Alice, *January, Week 2*, Eve). Encrypting a message to Eve requires the sender to know when Eve and all her ancestors joined the system. Therefore Scheme II is not joining-time-oblivious.

The lesson we learn from the failed Scheme II is that the keys must evolve in a way that is transparent to the encryption algorithm.

1.3.3 Scheme III

In our final unsuccessful attempt, Scheme III, a user adds a child to the hierarchy by giving him or her secret keys that depend both on the current time and on the child's position in the hierarchy. This is achieved by requiring that messages may only be decrypted by those who know two keys: one corresponding to the current time and the other corresponding to their positions in the hierarchy. Each user autonomously evolves his time key, and gives his newly joined children his time key in addition to their ID keys.

It is easy to see that this scheme is *n*ot forward-secure. An adversary who joins the hierarchy at the beginning of time can corrupt a user at any future time and obtain his or her ID key. Moreover, this adversary can derive any past time key (because he joined at the beginning of time). Thus, this adversary may decrypt any past message addressed to the exposed user.

For the same reason, the multiple hierarchical identity-based encryption (MHIBE) scheme generalized from Scheme III is not collusion-resistant, where the ciphertext for a user with multiple identities can be decrypted if some other individuals collude. MHIBE scheme is useful for secure communications with entities having multiple identities, and is described in Section 1.4.3 and 5.

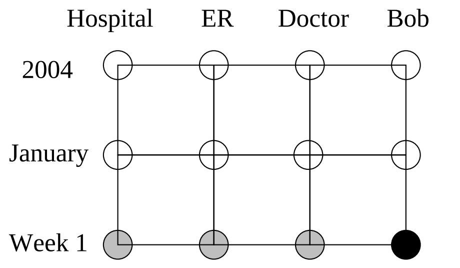
1.3.4 Comparisons

All the above trivial approaches fail. Therefore, constructing a forward-secure hierarchical ID-based encryption scheme that is both secure and scalable is not so straightforward. Our implementation, which is described in next Section, is still based on GS-HIBE [22] scheme and fs-PKE [13] scheme. Yet, it overcomes the problems existing in naive combinations of the two schemes, and satisfies the requirements of supporting dynamic joins, joining-time-obliviousness, forward security, and autonomous key updates.

1.4 Our Contributions

We make several contributions in this paper. First, we present a scalable and joining-time-oblivious forward-secure hierarchical identity-based encryption scheme that allows keys to be updated autonomously. Second, we show how our fs-HIBE scheme can be used to obtain a forward-secure public-key broadcast encryption (fs-BE) scheme. Third, we generalize our fs-HIBE scheme and discuss its application in secure communications with entities having multiple roles in role-based access control (RBAC) [33].

Figure 1: A schematic drawing of keys for ID-tuple (Hospital, ER, Doctor, Bob) at time period (2004, January, Week 1) in a forward-secure HIBE scheme. The ID-tuple (Hospital, ER, Doctor, Bob) of Bob is on x-axis. The tuple representing time period (2004, January, Week 1) is on y-axis. The origin represents the root identity (Hospital) and the highest-level time period (2004). The dark node represents Bob's key at Week 1. The grey nodes correspond to keys of Bob's ancestors at Week 1. Each white node represents an intermediate key. Secret keys at both the grey and white nodes can be used to compute private keys for Bob.



1.4.1 Forward-Secure HIBE Scheme

Our fs-HIBE protocol is based on the HIBE scheme by Gentry and Silverberg [22] and forward-secure public-key encryption (fs-PKE) [13] scheme due to Canetti, Halevi and Katz. It satisfies the requirements of dynamic joins, joining-time-obliviousness, forward security, and autonomous key updates.

A HIBE scheme involves only one hierarchy, whereas an fs-HIBE scheme has two hierarchies: ID and time. Each (ID-tuple, time) pair can be thought of as a point on the two-dimensional grid as follows. On the x-axis, we start with the identity of the root Public Key Generator in the ID hierarchy (e.g. Hospital), then in position (1,0) we have the identity of the first-level PKG (e.g. ER). In position (2,0) there is the identity of the second level PKG (e.g. Doctor), and in position (3,0) there may be another PKG or an individual user (e.g. Bob). Thus the x-axis represents an ID-tuple, for example (Hospital, ER, Doctor, Bob). Similarly, the y-axis represents the time. Divide a duration of time into multiple time periods and arrange them as leaf nodes of a tree. Internal nodes of the tree represent the time spans associated with their child nodes. Then, the origin of the grid corresponds to the root of the time hierarchy (e.g. 2004). In position (0, 1) we have the first level of the time hierarchy (e.g. January), and in position (0, 2) there is the next level of time hierarchy (e.g. Week 1). Thus a time period can be expressed as a tuple on the y-axis, for example (2004, January, Week 1). Figure 1 gives a schematic drawing of the correspondence between the tuples and keys in fs-HIBE.

In an fs-HIBE scheme, the secret key of an (ID-tuple, time) pair is associated with some path on the grid. For each grid point on that path, there is a corresponding element in this secret key. Such a path (secret key) is *not* joining-time-oblivious: it depends on when the user, as well as the nodes higher up, join the system. However, when encrypting, the sender does not have to know the path. What is non-trivial here is that, the path (secret key) and ciphertext of our fs-HIBE scheme are designed in such a way that we do not need to come up with a separate ciphertext for each possible path in order to achieve joining-time-obliviousness.

Our fs-HIBE scheme has collusion resistance and chosen ciphertext security in the random oracle model [5] assuming the difficulty of the bilinear Diffie-Hellman problem [11, 13, 22], provided that the depths of the ID hierarchy and time hierarchy are bounded by constants. The formal definitions and proofs of the scheme are given in the paper. The complexities of various parameters in our fs-HIBE scheme are summarized in Table 1 and are discussed in Section 6.

1.4.2 Forward-Secure Broadcast Encryption Scheme

We show how our fs-HIBE scheme can be used to construct a scalable forward-secure public-key broadcast encryption (fs-BE) scheme, which protects the secrecy of prior transmissions. A broadcast encryption (BE) [15, 16, 23, 24, 28, 31, 32, 35] scheme allows content providers to securely distribute digital contents to a dynamically changing user population. Each active user is issued a distinct secret key when he joins the system, by a trusted center. In comparison with the symmetric-key setting, a public-key BE scheme of [15] has a single public key associated with the system, which allows the distribution of the broadcast workload to untrusted third parties.

In a scalable forward-secure public-key broadcast encryption (fs-BE) scheme, users should be able to update their secret keys autonomously, and the trusted center should allow users to dynamically join the broadcast system at any time while achieving forward security. In addition, each content provider does not need to know when each user joins the system in order to broadcast the encrypted contents. The encryption algorithm of an fs-BE scheme should only depend on the current time and the set of authorized users, and thus be joining-time-oblivious. Applying our fs-HIBE to the public-key BE scheme [15] yields such an fs-BE scheme.

1.4.3 Multiple Hierarchical ID-Based Encryption

We further generalize our forward-secure hierarchical ID-based encryption scheme into a collusion-resistant multiple hierarchical identity-based encryption (MHIBE) scheme, and describe its application in secure communications with individuals who have multiple roles in role-based access control (RBAC) [33]. In large-scale organizations, a user may own multiple identities, each of which is represented by an ID-tuple. In MHIBE, a message can be encrypted under multiple ID-tuples (identities) and can be decrypted only by those who have *all* the required identities. The collusion-resistant property cannot be achieved using separate HIBE schemes. We note that the fs-HIBE scheme is a special case of our MHIBE scheme, in that in fs-HIBE scheme, time can be viewed as another identity of a user. Therefore the identities in MHIBE scheme capture a broad sense of meaning.

1.5 Outline of the Paper

The rest of the paper is organized as follows. In Section 2 we give definitions for fs-HIBE scheme and its security. In Section 3, we first recall the bilinear Diffie-Hellman assumption [11], and then give the construction of an fs-HIBE scheme and analyze the security. In Section 4, we show how an fs-HIBE scheme can be used to add forward secrecy to a public-key broadcast encryption scheme. In Section 5, we describe the multiple hierarchical ID-based encryption scheme. The complexity analysis is given in Section 6. Section 7 is the conclusion.

2 Forward-secure HIBE (fs-HIBE)

This section defines the notion of forward secrecy for HIBE scheme and the related security.

In an fs-HIBE scheme, secret keys associated with an ID-tuple are evolved with time. At any time period i an entity joins the system (hierarchy), its parent node computes its decryption key corresponding to time period i and other values necessary for the entity to compute its own future secret keys. Once the newly joined entity receives this secret information, at the end of each period it updates its secret key and erases the old key. During time period i, a message is encrypted under an ID-tuple and the time i. Decryption requires the secret key of the ID-tuple at time i.

2.1 Notations

Time Period: As usual in forward-secure public-key encryption [13] scheme, we assume for simplicity that the total number of time periods N is a power of 2; that is $N = 2^l$.

ID-tuple: An entity has a position in the hierarchy, defined by its tuple of IDs: $(\mathrm{ID}_1, \ldots, \mathrm{ID}_h)$. The entity's ancestors in the hierarchy are the users / PKGs whose ID-tuples are $\{(\mathrm{ID}_1, \ldots, \mathrm{ID}_i) : 1 \le i < h\}$. ID_1 is the ID for the root PKG.

Keys: There are two types of keys: $sk_{w,(\mathrm{ID}_1,\ldots,\mathrm{ID}_h)}$ and $SK_{i,(\mathrm{ID}_1,\ldots,\mathrm{ID}_h)}$. The node key $sk_{w,(\mathrm{ID}_1,\ldots,\mathrm{ID}_h)}$ is the key associated with some prefix w of the bit representation of a time period i and a tuple $(\mathrm{ID}_1, \ldots, \mathrm{ID}_h)$. $SK_{i,(\mathrm{ID}_1,\ldots,\mathrm{ID}_h)}$ denotes the key associated with time i and an ID-tuple $(\mathrm{ID}_1, \ldots, \mathrm{ID}_h)$. It consists of sk keys as follows: $SK_{i,(\mathrm{ID}_1,\ldots,\mathrm{ID}_h)} = \{sk_{i,(\mathrm{ID}_1,\ldots,\mathrm{ID}_h)}, sk_{w1,(\mathrm{ID}_1,\ldots,\mathrm{ID}_h)} : w0 \text{ is a prefix of } i\}$. When this causes no confusion, we denote the keys as $sk_{w,h}$ and $SK_{i,h}$, respectively.

2.2 fs-HIBE: Syntax

Forward-secure Hierarchical ID-Based Encryption (fs-HIBE) scheme: an fs-HIBE scheme is specified by five algorithms: Root Setup, Lower-level Setup, Update, Encrypt, and Decrypt:

Root Setup: The root PKG takes a security parameter k and the total number of time periods N, and returns *params* (system parameters) and the initial root key $SK_{0,1}$. The system parameters include a description of the message space $\mathcal{M}$ and the ciphertext space $\mathcal{C}$. The system parameters will be publicly available, while only the root PKG knows the initial root key.

Lower-level Setup: This algorithm is run by the parent of a newly joined child at time i to compute the child's private key. During a time period i, a lower-level entity (user or lower-level PKG) joins in the system at level h. Its parent at level $h-1$ computes the entity's key $SK_{i,h}$ associated with time period i. The inputs are the parent's private key $SK_{i,h-1}$, time i, and the ID-tuple of the child. (Note that the functionality of our Lower-level Setup includes the functionality of both the Lower-level Setup and the Extraction algorithm in the HIBE [22] scheme. This simplifies the protocol without any loss of generality.)

Update: During the time period i, an entity (PKG or individual) with ID-tuple $(\mathrm{ID}_1, \ldots, \mathrm{ID}_h)$ uses $SK_{i,h}$ to compute his key $SK_{(i+1),h}$ for the next time period $i+1$, and erases $SK_{i,h}$.

Encrypt: A sender inputs *params*, the index i of the current time period, $M \in \mathcal{M}$ and the ID-tuple of the intended message recipient, and computes a ciphertext $C \in \mathcal{C}$.

Decrypt: During the time period i, a user with the ID-tuple $(\mathrm{ID}_1, \ldots, \mathrm{ID}_h)$ inputs *params*, $C \in \mathcal{C}$, and its secret key $SK_{i,h}$ associated with time period i and the ID-tuple, and returns the message $M \in \mathcal{M}$.

Encryption and decryption must satisfy the standard consistency constraint, namely when $SK_{i,h}$ is the secret key generated by algorithm Lower-level Setup for ID-tuple $(\mathrm{ID}_1, \ldots, \mathrm{ID}_h)$ and time period i, then: $\forall M \in \mathcal{M}$,

$$\textsc{Decrypt}(params, SK_{i,h}, C) = M, \text{ where } C \leftarrow \textsc{Encrypt}(params, i, (\mathrm{ID}_1, \ldots, \mathrm{ID}_h), M)$$

2.3 fs-HIBE: Security

We allow an attacker to make *lower-level setup queries*. Also, we allow the adversary to choose the time period and the identity on which it wishes to be challenged. Notice that an adversary may *choose* the time period and the identity of its targets adaptively or nonadaptively. An adversary that chooses its targets adaptively first makes lower-level setup queries and decryption queries, and then chooses its targets based on the results of these queries. A nonadaptive adversary, on the other hand, chooses its targets independently from the results of the queries he makes. Security against an adaptive-chosen-target adversary, which is captured below, is the stronger notion of security than the non-adaptive one. It is also stronger than the selective-node security defined in the fs-PKE scheme by Canetti *et al.* [13].

Chosen-ciphertext Security (CCA2): We say an fs-HIBE scheme is semantically secure against adaptive chosen ciphertext, time period, and identity attack, if no polynomial time bounded adversary $\mathcal{A}$ has a non-negligible advantage against the challenger in the following game.

Setup: The challenger takes a security parameter k, and runs the ROOT SETUP algorithm. It gives the adversary the resulting system parameters *params*. It keeps the root secrets to itself.

Phase 1: The adversary issues queries $q_1, \ldots, q_m$, where q_i is one of the followings[2]:

1. Lower-level setup query $(t_i, \text{ID-tuple}_i)$: the challenger runs the LOWER-LEVEL SETUP algorithm to generate the private key $SK_{(t_i,\ \text{ID-tuple}_i)}$ corresponding to $(t_i, \text{ID-tuple}_i)$, and sends $SK_{(t_i, \text{ID-tuple}_i)}$ to the adversary.
2. Decryption query $(t_i, \text{ID-tuple}_i, C_i)$: the challenger runs the LOWER-LEVEL SETUP algorithm to generate the private key $SK_{(t_i, \text{ID-tuple}_i)}$ corresponding to the pair $(t_i, \text{ID-tuple}_i)$, runs the Decryption algorithm to decrypt C_i using $SK_{(t_i, \text{ID-tuple}_i)}$, and sends the resulting plaintext to the adversary.

These queries may be asked adaptively. Also, the queried ID-tuple$_i$ may correspond to a position at any level in the ID hierarchy, and the adversary is allowed to query for a future time and then for a past time.

Challenge: Once the adversary decides that **Phase 1** is over, it outputs two equal length plaintexts $M_0, M_1 \in \mathcal{M}$, a time period t^* and an ID-tuple* on which it wishes to be challenged. The constraint is that no lower-level setup query has been issued for ID-tuple* or any of its ancestors for any time $t \leq t^*$.

The challenger picks a random bit $b \in \{0, 1\}$, and sets $C^* = \text{ENCRYPT}(params, t^*, \text{ID-tuple}^*, M_b)$. It sends C^* as a challenge to the adversary.

Phase 2: The adversary issues more queries $q_{m+1}, \ldots, q_n$, where q_i is one of[3]:

1. Lower-level setup query $(t_i, \text{ID-tuple}_i)$, where the time period t_i and ID-tuple$_i$ are under the same restriction as in **Challenge**: the challenger responds as in **Phase 1**.
2. Decryption query $(t_i, \text{ID-tuple}_i, C_i) \neq (t^*, \text{ID-tuple}^*, C^*)$: the challenger responds as in **Phase 1**.

Guess: The adversary outputs a guess $b' \in \{0, 1\}$. The adversary wins the game if $b = b'$. We define its advantage in attacking the scheme to be $|\Pr[b = b'] - \frac{1}{2}|$.

3 A Forward-secure HIBE Scheme

Here, we present a forward-secure hierarchical identity-based encryption scheme. Following the presentation standard in the IBE literature [11, 22], we first present an fs-HIBE with *one-way security*. One-way security is the weakest notion of security. It means that it is hard to recover a

[2]In the random oracle model, the adversary may also issue public key queries. Public key query $(t_i, \text{ID-tuple}_i)$: challenger runs a hash algorithm on $(t_i, \text{ID-tuple}_i)$ to obtain the public key $H(t_i \circ \text{ID-tuple}_i)$ corresponding to $(t_i, \text{ID-tuple}_i)$, where H is a random oracle.

[3]In the random oracle model, the adversary may also issue public key query. Public key query $(t_i, \text{ID-tuple}_i)$: the challenger responds as in **Phase 1**.

plaintext with a passive attack. A standard technique, due to Fujisaki and Okamoto [19], converts one-way security to CCA2 security in the random oracle model. For completeness, we give our definition of one-way security in the Appendix B, and the Fujisaki-Okamoto conversion of the one-way secure fs-HIBE in the Appendix C. Our scheme, which is based on the HIBE scheme of Gentry and Silverberg [22] and the fs-PKE scheme of Canetti, Halevi and Katz [13, 27], overcomes the scalability and security problems that exist when naively combining the two schemes as described in Section 1.3. Next, we first give the number theoretic assumptions needed in our scheme, and then describe the algorithms in our construction. Proofs of security of our fs-HIBE scheme are shown in the Appendix D.

3.1 Assumptions

The security of our fs-HIBE scheme is based on the difficulty of the bilinear Diffie-Hellman (BDH) problem [11]. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two cyclic groups of some large prime order q. We write $\mathbb{G}_1$ additively and $\mathbb{G}_2$ multiplicatively. Our schemes make use of a bilinear pairing.

Admissible pairings: Following Boneh and Franklin [11], we call $\hat{e}$ an admissible pairing if $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ is a map with the following properties:

1. Bilinear: $\hat{e}(aP, bQ) = \hat{e}(P, Q)^{ab}$ for all $P, Q \in \mathbb{G}_1$ and all $a, b \in \mathbb{Z}$.
2. Non-degenerate: The map does not send all pairs in $\mathbb{G}_1 \times \mathbb{G}_1$ to the identity in $\mathbb{G}_2$.
3. Computable: There is an efficient algorithm to compute $\hat{e}(P, Q)$ for any $P, Q \in \mathbb{G}_1$.

We refer the readers to papers by Boneh and Franklin [11] and Boneh and Silverberg [12] for examples and discussions of groups that admit such pairings.

Bilinear Diffie-Hellman (BDH) Parameter Generator: As in IBE [11] scheme, a randomized algorithm $\mathcal{IG}$ is a BDH parameter generator if $\mathcal{IG}$ takes a security parameter $k > 0$, runs in time polynomial in k, and outputs the description of two groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of the same prime order q and the description of an admissible paring $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$.

BDH Problem: As in IBE [11] scheme, given a randomly chosen $P \in \mathbb{G}_1$, as well as aP, bP, and cP (for unknown randomly chosen $a, b, c \in \mathbb{Z}_q$), compute $\hat{e}(P, P)^{abc}$.

For the BDH problem to be hard, $\mathbb{G}_1$ and $\mathbb{G}_2$ must be chosen so that there is no known algorithm for efficiently solving the Diffie-Hellman problem in either $\mathbb{G}_1$ or $\mathbb{G}_2$. Note that if the BDH problem is hard for a paring $\hat{e}$, then it follows that $\hat{e}$ is non-degenerate.

BDH Assumption: As in IBE [11] scheme, we say a BDH parameter generator $\mathcal{IG}$ satisfies the BDH assumption if the following is negligible in k for all PPT algorithm $\mathcal{A}$:

$$\Pr[(\mathbb{G}_1, \mathbb{G}_2, \hat{e}) \leftarrow \mathcal{IG}(1^k); P \leftarrow \mathbb{G}_1; a, b, c \leftarrow \mathbb{Z}_q : \mathcal{A}(\mathbb{G}_1, \mathbb{G}_2, \hat{e}, P, aP, bP, cP) = \hat{e}(P, P)^{abc}]$$

3.2 fs-HIBE: Implementation

For simplicity of description, our fs-HIBE construction makes use of a version of fs-PKE scheme due to Katz [27]. In Katz's scheme, time periods are associated with the *leaf* nodes of a binary tree (Rather than with all tree nodes as in the scheme by Canetti *et al.* [13]. Our fs-HIBE scheme can also be realized based on the fs-PKE scheme by Canetti *et al.*, which will give faster key update time. The complexity discussion of our scheme is in Section 6).

We construct a full binary tree of height l, as in Katz's scheme [27]. The root of this tree is labeled ϵ; all other nodes are recursively labeled as follows: if the label of a node is w, then its left child is labeled $w0$, and its right child is labeled $w1$. This way, for each time period $i \in \{0, N-1\}$, there is a leaf labeled with the binary representation of i. Following the fs-PKE scheme [13], we denote the k-bit prefix of a word $w = w_1 w_2 \dots w_d$ by $w|_k$, that is, $w|_k = w_1 \dots w_k$ for $k \leq d$. The word w is of length d, i.e. $|w| = d$. Let $w|_0 = \epsilon$ and $w = w|_d$.

At the beginning of time, public parameters are generated. At time t, the entity at level h with ID-tuple $(\mathrm{ID}_1, \dots, \mathrm{ID}_h)$ holds a secret key $SK_{t,(\mathrm{ID}_1,\dots,\mathrm{ID}_h)}$. Recall that we denote key $SK_{t,(\mathrm{ID}_1,\dots,\mathrm{ID}_h)}$ as $SK_{t,h}$ when this causes no confusion. $SK_{t,h}$ consists of $(sk_{t,h}, \{sk_{w,h}\})$, where w are all the labels of the right sibling, if one exists, of each ancestor of the node labeled t in the complete binary tree. $sk_{w,h}$ are also called node keys. At the beginning of time, as public parameters are generated, the values $sk_{0,1}$ and $sk_{1,1}$ are created by the root PKG, whose ID is ID_1. Each $sk_{w,h}$, where w is any string, can be used to compute an $sk_{w,u}$, where $u > h$, for tuple $(\mathrm{ID}_1, \dots, \mathrm{ID}_u)$ who is a descendant of $(\mathrm{ID}_1, \dots, \mathrm{ID}_h)$. The algorithm to use is Lower-Level Setup.

Each $sk_{w,h}$, where w is any string, is also used to compute the value $sk_{(w \circ b),h}$, where $b = 0$ or 1. They are the child nodes of w on the binary tree. The algorithm for doing so is Compute Next, which is defined as follows. In Compute Next, an entity (a PKG or a user) with ID-tuple $(\mathrm{ID}_1, \dots, \mathrm{ID}_h)$ uses the value $sk_{w,h}$ associated with a node w to compute values $sk_{(w0),h}$ and $sk_{(w1),h}$ for the two child nodes of w. Compute Next is a helper function and is called by the algorithm Root Setup and Update.

We must delete all information from which $sk_{t',h}$ for $t' < t$ can be inferred. The algorithm for computing the keys for the next time period and erasing old keys is Update.

The public parameters, time t, and ID-tuple $(\mathrm{ID}_1, \dots, \mathrm{ID}_h)$ are all that a sender needs in order to send an encrypted message to ID-tuple $(\mathrm{ID}_1, \dots, \mathrm{ID}_h)$ at time t using algorithm Encrypt.

The value $sk_{t,h}$ is all that the user with tuple $(\mathrm{ID}_1, \dots, \mathrm{ID}_h)$ needs in order to decrypt at time t. The algorithm for doing so is Decrypt.

Let us look at the contents of each $sk_{w,h}$ more closely. It has two components $S_{w,h}$ and $\mathcal{Q}_{w,h}$. $S_{w,h}$ is a point in $\mathbb{G}_1$, and $\mathcal{Q}_{w,h}$ contains a set of Q-values, which will be explained later. If w represents a leaf on the binary tree, $S_{w,h}$ and the Q-values in $\mathcal{Q}_{w,h}$ together are used for decryption for ID-tuple $(\mathrm{ID}_1, \dots, \mathrm{ID}_h)$ at time w. If w is an internal node on the binary tree, these values are used for generating future decryption keys.

Computing $S_{w,h}$ makes use of $|w| \times h$ number of secret values $s_{w|_k,(\mathrm{ID}_1,\dots,\mathrm{ID}_j)}$, where $1 \leq k \leq |w|$ and $1 \leq j \leq h$. The shorthand notation for $s_{w|_k,(\mathrm{ID}_1,\dots,\mathrm{ID}_j)}$ is $s_{k,j}$, when $w|_k$ and $(\mathrm{ID}_1, \dots, \mathrm{ID}_j)$ are clear from the context. Given a node w on the binary tree and an ID-tuple $(\mathrm{ID}_1, \dots, \mathrm{ID}_h)$, there is a secret value $s_{k,j}$ for every (time, ID-tuple) combination, where time corresponds to a node that has some prefix $w|_k$ of node w (including w itself and excluding the root node ϵ) and ID-tuple is some ancestor $(\mathrm{ID}_1, \dots, \mathrm{ID}_j)$ of ID-tuple $(\mathrm{ID}_1, \dots, \mathrm{ID}_h)$ (including itself). Each $s_{k,j}$ is chosen randomly from $\mathbb{Z}_q$.

Each $s_{k,j}$ is also used for computing $Q_{k,j}$, which is called a Q-value. For each $s_{k,j}$ there is a Q-value $Q_{k,j}$, which is an element in $\mathbb{G}_1$. Q-values are used in Decrypt. $\mathcal{Q}_{w,h}$ is a set of Q-values. All $s_{k,j}$ values are erased once $S_{w,h}$ and Q-values are computed. The construction is shown below.

Construction Let $\mathcal{IG}$ be a BDH parameter generator for which the BDH assumption holds.

Root Setup$(1^k, N = 2^l)$: The root PKG with ID_1 does the following:

1. $\mathcal{IG}$ is run to generate groups $\mathbb{G}_1, \mathbb{G}_2$ of order q and bilinear map $\hat{e}$.
2. A random generator $P \leftarrow \mathbb{G}_1$ is selected along with random $s_\epsilon \leftarrow \mathbb{Z}_q$. Set $Q = s_\epsilon P$.
3. Choose a cryptographic hash function $H_1 : \{0,1\}^* \rightarrow \mathbb{G}_1$. Choose a cryptographic hash function $H_2 : \mathbb{G}_2 \rightarrow \{0,1\}^n$ for some n. The security analysis will treat H_1 and H_2 as random oracles [5]. The message space is $\mathcal{M} = \{0,1\}^n$. The ciphertext space is $\mathcal{C} = \mathbb{G}_1^{l \times h} \times \{0,1\}^n$ where h is the level of the recipient. The system parameters are *params* $= (\mathbb{G}_1, \mathbb{G}_2, \hat{e}, P, Q, H_1, H_2)$.

All operations of fs-HIBE are performed under *params*. The master key is $s_\epsilon \in \mathbb{Z}_q$.
The root PKG needs to generate not only the sk key associated with the current time period 0, but also the sk keys corresponding to the internal nodes on the binary tree whose bit representations are all 0 except the last bit. The sk key for time 0 is denoted as $sk_{0^l,1}$. The rest of sk values are used by the root PKG to generate keys for future time periods, and are represented as $\{sk_{1,0}, sk_{(01),1}, \ldots, sk_{(0^{l-1}1),1}\}$. These values are generated recursively as follows.

(a) Set the secret point $S_{0,1}$ to $s_\epsilon H_1(0 \circ \mathrm{ID}_1)$, and $S_{1,1}$ to $s_\epsilon H_1(1 \circ \mathrm{ID}_1)$.
(b) Set secret key $sk_{0,1} = (S_{0,1}, \emptyset)$ and $sk_{1,1} = (S_{1,1}, \emptyset)$. Root PKG uses $sk_{0,1}$ to recursively call algorithm Compute Next (defined below) to generate its secret keys. Let $(sk_{w00,1}, sk_{w01,1}) =$ Compute Next$(sk_{w0,1}, w0, \mathrm{ID}_1)$, for all $1 \leq |w0| \leq l-1$.
(c) Set the root PKG's secret key for time period 0 as $SK_{0,1} = (sk_{0^l,1}, \{sk_{1,1}, sk_{(01),1}, \ldots, sk_{(0^{l-1}1),1}\})$, and erase all other information.

Compute Next$(sk_{w,h}, w, (\mathrm{ID}_1 \ldots \mathrm{ID}_h))$: This is a helper method and is called by the Root Setup and Update algorithms. It takes a secret key $sk_{w,h}$, a node w, and an ID-tuple, and outputs keys $sk_{(w0),h}$, $sk_{(w1),h}$ for time nodes $w0$ and $w1$ of $(\mathrm{ID}_1 \ldots \mathrm{ID}_h)$. (Note that this is similar to the Extraction algorithms in the Gentry-Silverberg HIBE and the Derivation algorithm in the fs-PKE schemes, only here we extract keys corresponding to nodes in the time hierarchy for a given ID-tuple.)

1. Parse w as $w_1 \ldots w_d$, where $|w| = d$. Parse ID-tuple as $\mathrm{ID}_1, \ldots, \mathrm{ID}_h$. Parse $sk_{w,h}$ associated with time node w as $(S_{w,h}, \mathcal{Q}_{w,h})$, where $S_{w,h} \in \mathbb{G}_1$ and $\mathcal{Q}_{w,h} = \{Q_{k,j}\}$ for all $1 \leq k \leq d$ and $1 \leq j \leq h$, except for $k = 1$ and $j = 1$.
2. Choose random $s_{(d+1),j} \in \mathbb{Z}_q$ for all $1 \leq j \leq h$.
3. Set $S_{(w0),h} = S_{w,h} + \sum_{j=1}^{h} s_{(d+1),j} H_1(w0 \circ \mathrm{ID}_1 \ldots \mathrm{ID}_j)$.
4. Set $S_{(w1),h} = S_{w,h} + \sum_{j=1}^{h} s_{(d+1),j} H_1(w1 \circ \mathrm{ID}_1 \ldots \mathrm{ID}_j)$.
5. Set $Q_{(d+1),j} = s_{(d+1),j} P$ for all $j \in [1, h]$.
6. Set $\mathcal{Q}_{(w0),h}$ and $\mathcal{Q}_{(w1),h}$ to be the union of $\mathcal{Q}_{w,h}$ and $Q_{(d+1),j}$ for all $1 \leq j \leq h$.
7. Output $sk_{(w0),h} = (S_{(w0),h}, \mathcal{Q}_{(w0),h})$ and $sk_{(w1),h} = (S_{(w1),h}, \mathcal{Q}_{(w1),h})$.
8. Erase $s_{(d+1),j}$ for all $1 \leq j \leq h$.

Lower-level Setup$(SK_{i,(h-1)}, i, (\mathrm{ID}_1 \ldots \mathrm{ID}_h))$: Let E_h be an entity that joins the hierarchy during the time period $i < N-1$ with ID-tuple $(\mathrm{ID}_1, \ldots, \mathrm{ID}_h)$. E_h's parent generates E_h's key $SK_{i,h}$ using its key $SK_{i,(h-1)}$ as follows:

1. Parse i as $i_1 \ldots i_l$ where $l = \log_2 N$. Parse $SK_{i,(h-1)}$ as $(sk_{i,(h-1)}, \{sk_{(i|_{k-1}1),(h-1)}\}_{i_k=0})$.
2. For each value $sk_{w,(h-1)}$ in $SK_{i,(h-1)}$, E_h's parent does the following to generate E_h's key $sk_{w,h}$:
   (a) Parse w as $w_1 \ldots w_d$, where $d \leq l$, and parse the secret key $sk_{w,(h-1)}$ as $(S_{w,(h-1)}, \mathcal{Q}_{w,(h-1)})$.
   (b) Choose random $s_{k,h} \in \mathbb{Z}_q$ for all $1 \leq k \leq d$. Recall that $s_{k,j}$ is a shorthand for $s_{w|_k,(\mathrm{ID}_1 \ldots \mathrm{ID}_j)}$ associated with time node $w|_k$ and tuple $(\mathrm{ID}_1 \ldots \mathrm{ID}_j)$.
   (c) Set the child entity E_h's secret point
   $S_{w,h} = S_{w,(h-1)} + \sum_{k=1}^{d} s_{k,h} H_1(w|_k \circ \mathrm{ID}_1 \ldots \mathrm{ID}_h)$.

(d) Set $Q_{k,h} = s_{k,h}P$ for all $1 \leq k \leq d$. Let $\mathcal{Q}_{w,h}$ be the union of $\mathcal{Q}_{w,(h-1)}$ and $Q_{k,h}$ for all $1 \leq k \leq d$.

(e) Set $sk_{w,h}$ to be $(S_{w,h}, \mathcal{Q}_{w,h})$.

3. E_h's parent sets E_h's $SK_{i,h} = (sk_{i,h}, \{sk_{(i|_{k-1}1),h}\}_{i_k=0})$, and erases all other information.

Update$(SK_{i,h}, i+1, (\mathrm{ID}_1 \ldots \mathrm{ID}_h))$ (where $i < N-1$): At the end of time i, an entity (PKG or individual) with ID-tuple $(\mathrm{ID}_1, \ldots, \mathrm{ID}_h)$ does the following to compute its private key for time $i+1$, as in the fs-PKE scheme [13, 27].

1. Parse i as $i_1 \ldots i_l$, where $|i| = l$. Parse $SK_{i,h}$ as $(sk_{(i|_l),h}, \{sk_{(i|_{k-1}1),h}\}_{i_k=0})$. Erase $sk_{i|_l,h}$.
2. We distinguish two cases. If $i_l = 0$, simply output the remaining keys as the key $SK_{(i+1),h}$ for the next period for ID-tuple $(\mathrm{ID}_1, \ldots, \mathrm{ID}_h)$. Otherwise, let $\tilde{k}$ be the largest value such that $i_{\tilde{k}} = 0$ (such $\tilde{k}$ must exist since $i < N-1$). Let $i' = i|_{\tilde{k}-1}1$. Using $sk_{i',h}$ (which is included as part of $SK_{i,h}$), recursively apply algorithm Compute Next to generate keys $sk_{(i'0^d1),h}$ for all $0 \leq d \leq l - \tilde{k} - 1$, and $sk_{(i'0^{d-\tilde{k}},h)}$. The key $sk_{(i'0^{d-\tilde{k}},h)}$ will be used for decryption in the next time period $i+1$; the rest of sk keys are for computing future keys. Erase $sk_{i',h}$ and output the remaining keys as $SK_{(i+1),h}$.

Encrypt$(i, (\mathrm{ID}_1, \ldots, \mathrm{ID}_h), M)$ (where $M \in \{0,1\}^n$):

1. Parse i as $i_1 \ldots i_l$. Select random $r \leftarrow \mathbb{Z}_q$.
2. Denote $P_{k,j} = H_1(i|_k \circ \mathrm{ID}_1 \ldots \mathrm{ID}_j)$ for all $1 \leq k \leq l$ and $1 \leq j \leq h$. Output $\langle i, (\mathrm{ID}_1, \ldots, \mathrm{ID}_h), C\rangle$, where $C = (rP, rP_{2,1}, \ldots, rP_{l,1}, rP_{1,2}, \ldots, rP_{l,2}, \ldots, rP_{1,h}, \ldots, rP_{l,h}, M \oplus H_2(\hat{e}(Q, H_1(i_1 \circ \mathrm{ID}_1))^r))$.

Decrypt$(i, (\mathrm{ID}_1, \ldots, \mathrm{ID}_h), SK_{i,h}, C)$:

1. Parse i as $i_1 \ldots i_l$. Parse $SK_{i,h}$ associated with the ID-tuple as $(sk_{i,h}, \{sk_{(i|_{k-1}1),h}\}_{i_k=0})$ and the key $sk_{i,h}$ as $(S_{i,h}, \mathcal{Q}_{i,h})$. Parse $\mathcal{Q}_{i,h}$ as $\{Q_{k,j}\}$ for all $1 \leq k \leq l$ and $1 \leq j \leq h$, except for $k = 1$ and $j = 1$.
2. Parse C as $(U_0, U_{2,1}, \ldots, U_{l,1}, U_{1,2}, \ldots, U_{l,2}, \ldots, U_{1,h}, \ldots, U_{l,h}, V)$.
3. $M = V \oplus H_2(\frac{\hat{e}(U_0, S_{i,h})}{g})$, where g is: $\Pi_{k=1}^{l}\Pi_{j=2}^{h}\hat{e}(Q_{k,j}, U_{k,j})\Pi_{k=2}^{l}\hat{e}(Q_{k,1}, U_{k,1})$.

Verification of the correctness of our Decrypt algorithm is shown in the Appendix A. In Appendix C, using Fujisaki-Okamoto padding [19] and the help of random oracles H_3 and H_4, algorithm Encrypt and Decrypt can be converted into ones with chosen ciphertext security, as in BF-IBE [11] and GS-HIBE [22].

Theorem 3.1. *Suppose there is a nonadaptive adversary $\mathcal{A}$ that has advantage ϵ against the one-way secure fs-HIBE scheme for some* fixed *time t and ID-tuple, and that makes $q_{H_2} > 0$ hash queries to the hash function H_2 and a finite number of lower-level setup queries. If the hash functions H_1, H_2 are random oracles, then there is an algorithm $\mathcal{B}$ that solves the BDH in groups generated by $\mathcal{IG}$ with advantage $(\epsilon - \frac{1}{2^n})/q_{H_2}$ and running time $\mathcal{O}(time(\mathcal{A}))$.*

Theorem 3.2. *Suppose there is an adaptive adversary $\mathcal{A}$ that has advantage ϵ against the one-way secure fs-HIBE scheme targeting some time and some ID-tuple at level h, and that makes $q_{H_2} > 0$ hash queries to the hash function H_2 and at most $q_E > 0$ lower-level setup queries. Let $l = \log_2 N$, where N is the total number of time periods. If the hash functions H_1, H_2 are random oracles, then there is an algorithm $\mathcal{B}$ that solves the BDH in groups generated by $\mathcal{IG}$ with advantage $(\epsilon(\frac{h+l}{e(2lq_E+h+l)})^{(h+l)/2} - \frac{1}{2^n})/q_{H_2}$ and running time $\mathcal{O}(time(\mathcal{A}))$.*

The proofs of the theorems are in the Appendix D.

4 Application: Forward-Secure Broadcast Encryption

In this section, we show how the fs-HIBE scheme can be used to build a scalable forward-secure public-key broadcast encryption (fs-BE) scheme which is joining-time-oblivious. In what follows, N denotes the total number of time periods, $\mathcal{E}$ denotes the universe of users and $E = |\mathcal{E}|$.

4.1 fs-BE: Syntax

Forward-Secure Broadcast Encryption (fs-BE): An fs-BE scheme is specified by five poly-time algorithms KeyGen, Reg, Upd, Enc, Dec:

KeyGen: The *key generation algorithm*, is a probabilistic algorithm run by the center to set up the parameters of the scheme. KeyGen takes as input a security parameter k and possibly $r_{\max}$ (where $r_{\max}$ is a revocation threshold, i.e. the maximum number of users that can be revoked). The input also includes the total number E of users in the system and the total number of time periods N. KeyGen generates the public key PK and the initial master secret key MSK_0.

Reg: The *registration algorithm*, is a probabilistic algorithm run by the center to compute the secret initialization data for a new user. Reg takes as input the master secret key MSK_t at time t, the identity u of the user and the current time period $t < N - 1$ and outputs the new secret key $USK_{t,u}$.

Upd: The *key update algorithm*, is a deterministic algorithm run by an entity (center or user) to update its own secret key of time t into a new secret key valid for the following time period $t + 1$. For a user, Upd takes as input the public key PK, the identity u of a user, the current time period $t < N - 1$, and the user's secret key $USK_{t,u}$, and outputs the new user's secret key $USK_{t+1,u}$. For the center, the algorithm takes as input the public key PK, the current time period $t < N$, and the key MSK_t, and outputs the secret key MSK_{t+1}.

Enc: The *encryption algorithm*, is a probabilistic algorithm that each content provider can use to encrypt messages. Enc takes as input the public key PK, a message M, the current time period t 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 C to be broadcast.

Dec: The *decryption algorithm*, is a deterministic algorithm run by each user to recover the content from the broadcast. Dec takes as input the public key PK, the identity u of a user, a time period $t < N$, the user's secret key $USK_{t,u}$ and a ciphertext C, and returns a message M.

An fs-BE scheme should satisfy the following correctness constraint: for any pair (PK, MSK_t) output by the algorithm $\textsc{KeyGen}(k, r_{\max}, N, E)$, any $t < N$, any $\mathcal{R} \subseteq \mathcal{E}, (|\mathcal{R}| \leq r_{\max})$, any user $u \in \mathcal{E} \setminus \mathcal{R}$ with secret key $USK_{t,u}$ (properly generated for time period t) and any message M, it should hold that:

$$M = \textsc{Dec}(PK, u, t, USK_{t,u}, \textsc{Enc}(PK, M, t, \mathcal{R})).$$

4.2 fs-BE: Security

In fs-BE scheme, if a user leaks his or her secret key and is not revoked by a content provider, the security of subsequent communications broadcasted by such provider is compromised. As a matter of fact, the forward security of broadcast encryption schemes guarantees that this is the *only* case where unauthorized access to the broadcast content may occur. The advantage of the adversary is not significantly improved even if she corrupts multiple users at different time periods. We formalize the security definition of fs-BE below.

Chosen-ciphertext Security: An fs-BE scheme is *forward-secure against chosen-ciphertext attack* if no polynomial time bounded adversary $\mathcal{A}$ has a non-negligible advantage against the challenger in the following game:

Setup: The challenger takes security parameters $k, r_{\max}$, and runs the KeyGen algorithm, for the specified number of users E and time periods N. It gives the adversary the resulting system public key PK and keeps the initial master secret key MSK_0 secret to itself.
Phase 1: The adversary issues, in any adaptively-chosen order, queries $q_1, \ldots, q_m$, where q_i is one of the followings:

1. Corrupt query (u, t): the challenger runs algorithm $\textsc{Reg}(MSK_t, u, t)$ to generate the private key $USK_{t,u}$ corresponding to user u at time t, and sends $USK_{t,u}$ to the adversary.
2. Decryption query (u, t, C): the challenger first runs the $\textsc{Reg}(MSK_t, u, t)$ algorithm to recover private key $USK_{t,u}$ corresponding to user u at time t, and then runs decryption algorithm $\textsc{Dec}(PK, u, t, USK_{t,u}, C)$ to decrypt C, and sends the resulting plaintext to the adversary.

Challenge: Once the adversary decides that **Phase 1** is over, it outputs two equal-length plaintexts $M_0, M_1 \in \mathcal{M}$, and a time period t^* on which it wishes to be challenged. The challenger picks a random bit $b \in \{0, 1\}$, and set $C^* = \textsc{Enc}(PK, M_b, t^*, \mathcal{R}_{t^*})$, where $\mathcal{R}_{t^*} = \{u \mid \mathcal{A}$ asked a query Corrupt(u, t), for some $t \leq t^*\}$. It sends C^* as a challenge to the adversary.
Phase 2: The adversary issues more queries $q_{m+1}, \ldots, q_n$, where q_i is one of:

1. Corrupt query (u, t): the challenger first checks that either $u \in \mathcal{R}_{t^*}$ or $t > t^*$ and if so, it responds as in **Phase 1**. Notice that if a bound r_{max} was specified in KeyGen, then the adversary is restricted to corrupt at most r_{max} distinct users via Corrupt queries.
2. Decryption query (u, t, C): the challenger first checks that either $C \neq C^*$ or $u \in \mathcal{R}_{t^*}$ or $t \neq t^*$ and if so, it responds as in **Phase 1**.

Guess: The adversary outputs a guess $b' \in \{0, 1\}$ and wins the game if $b = b'$. We define its advantage in attacking the scheme to be $|\Pr[b = b'] - \frac{1}{2}|$.

4.3 fs-BE: A Construction Based on fs-HIBE

Here, we show how our fs-HIBE scheme can be applied to the construction of the public-key broadcast encryption of [15] to obtain a forward-secure public-key BE scheme. Dodis and Fazio [15] provided a construction that extends the symmetric-key broadcast encryption scheme of Naor *et al.* [31] to the public-key setting, based on any secure HIBE scheme: for ease of reference, we include it in Appendix E.1. The construction of [15] also applies to the scheme of Halevy and Shamir [24], that improves upon the work of [31]. The symmetric-key BE scheme of Halevy and Shamir is an instance of the *Subset Cover Framework* [31]. The main idea of the framework is to define a family $\mathcal{S}$ of subsets of the universe $\mathcal{E}$ of users in the system, and to associate each subset with a key, which is made available to all the users belonging to the given subset. To broadcast a message to all the subscribers except those in some set $\mathcal{R}$, a content provider first *covers* the set of privileged users using subsets from the family $\mathcal{S}$. This is done by identifying a partition of $\mathcal{E} \setminus \mathcal{R}$, where all the subsets are elements of $\mathcal{S}$. Then, the provider encrypts the message for all the subsets in that partition. To decrypt, a user $u \notin \mathcal{R}$ first identifies the subset in the partition of $\mathcal{E} \setminus \mathcal{R}$ to which he belongs, and then recovers the corresponding secret keys from his secret information.

In the public-key BE scheme [15], the subsets containing a given user are organized into groups, and a special secret key, *protokey*, is associated with each of these groups. A user only needs to store these protokeys, from which he can derive the actual decryption keys corresponding to all the subsets in the group. Such an organization of the subsets of the family $\mathcal{S}$ produces a hierarchy, in which the leaves are elements of $\mathcal{S}$ and each internal node corresponds to a group of subsets. Using HIBE, a secret key can be associated with each internal node in the hierarchy, and constitutes the protokey for the group corresponding to that internal node.

In order to add forward secrecy in the public-key BE scheme, we essentially apply the fs-HIBE scheme to the above hierarchy. In fs-BE scheme, a protokey is associated with not only a node in the hierarchy, but also with a time period t. In the KeyGen operation, the center runs the Root Setup algorithm of fs-HIBE to compute its master secret $SK_{0,1}$. This key evolves with time, and is used by the center to compute protokeys for users. In the Reg operation, a user joins the broadcast at some time t, and the center uses its current master secret key $SK_{t,1}$ to derive protokeys for the user by running the Lower-level Setup of fs-HIBE. The center and users evolve their secret keys with time autonomously by calling the Update algorithm of fs-HIBE. In the Enc algorithm, a content provider uses the Encrypt algorithm of fs-HIBE to encrypt the message not only with respect to the nodes in the hierarchy that represents the subsets in the partition of $\mathcal{E} \setminus \mathcal{R}$, but also to the current time t. In the Dec operation, the user first runs the Lower-Level Setup algorithm of fs-HIBE to derive the current secret keys from his protokey at time t. These secret keys are used for decryption. The construction of our fs-BE scheme is given in Appendix E. We analyze the complexity of fs-BE operations in Section 6.

5 Multiple Hierarchical Identity-Based Encryption Scheme

ID-based cryptographic schemes have been used in complex access control scenarios [25, 34]. In this paper, we generalize the fs-HIBE into a collusion resistant multiple hierarchical ID-based encryption (MHIBE) scheme, where a message can be encrypted under multiple ID-tuples. The applications of MHIBE scheme include secure communications with users having multiple identities.

Motivations for MHIBE In role-based access control systems (RBAC) [33], individuals are assigned roles according to their qualifications, and access decisions are based on roles. The use of roles to control access is proven to be an effective means for streamlining the security management process [33]. Communications to a specific role may need to be protected so that messages can be read only by members of that role. This can be done using a shared key approach, which can be realized by an HIBE scheme. Members of a role are given a secret group key that is used for decrypting messages encrypted with the group public key of that role, which is an ID-tuple in HIBE. For example, the public key of the role *doctor* in the Emergency Room at a hospital is the ID-tuple (Hospital, ER, doctor), and members of the role *doctor* are given the corresponding private key in HIBE. The hierarchical structure of public keys in HIBE makes it particularly suitable for managing role communications in large organizations. This group key approach is efficient and scalable compared to encrypting the message with individual recipients' personal public keys, because a message is encrypted only once (under the public key of the role).

A user may have multiple roles. Some messages are intended to be read only by those who have multiple roles, and should not be recovered by collusions among role members. For example, the intended message recipients are those who must take on both role *doctor* in ER and role *research manager* at the affiliated medical school of the hospital. In healthcare systems, medical data such as patient records are extremely sensitive, therefore, achieving this type of secure communications is important. However, the GS-HIBE [22] scheme provides cryptographic operations only if the message is encrypted under one identity (ID-tuple). It cannot be used for communications to an *intersection* of identities. Note that the Dual-Identity-Based Encryption scheme by Gentry and Silverberg [22] is different from what we want to achieve here. The word "dual" in their scheme [22] refers that the identities of both the sender and the recipient, rather than just the recipient, are required as input into the encryption and decryption algorithms.

To solve the problem of secure communications to members having multiple roles, we develop a multiple hierarchical identity-based encryption (MHIBE) scheme, where encryption is under multiple ID-tuples. In addition, it can be used for authenticating multiple hierarchical identities in the hidden credential protocol [25], where the success of authentication of identities is implied if one can correctly decrypt the message encrypted with the required identities of the intended recipients. What makes the problem interesting is that the *intersection* of identities is different from the *union* of identities, which implies that a proper scheme should be collusion-resistant: secure even if adversaries with partial roles collude. In other words, it requires that compromising the private keys of individual identities does not compromise the messages encrypted with the intersection of identities. This property cannot be achieved by the broken Scheme III described in Section 1.3, where two separate HIBE schemes are used, as it is not collusion-resistant.

Next we use an example to describe the MHIBE scheme, including key acquisition, encryption, and the properties of MHIBE implementation generalized from our fs-HIBE scheme.

5.1 Identity-set and Joining-path-obliviousness

In MHIBE, we define an *identity-set* as the set of identities that a user has, each represented as an ID-tuple. For example, Bob's identity-set is {(Hospital, ER, Doctor), (Hospital, School, Manager)}. An *ancestor* E' of a node E has the same number of ID-tuples in its identity-set as that of E, and for each ID-tuple T in the identity-set of E, there is an ID-tuple in the identity-set of E' such that it is either the ancestor of T in HIBE or the same as T. In addition, the ancestor E' of the node E cannot be E. All ancestors of node E are capable of generating secret keys for E.

In an MHIBE scheme, Bob may obtain his key directly from either of the two ancestor entities. One is the entity whose identity-set is {(Hospital, ER), (Hospital, School, Manager)}. And the other has the identity-set {(Hospital, ER, Doctor), (Hospital, School)}. Bob's parents obtain their keys from their parents in the same way. The highest-level ancestor in this example is the hospital and has the identity-set {Hospital, Hospital} (not {Hospital}). The root secret s_ϵ used for computing the private key for identity-set {Hospital, Hospital} may be the same as the root secret used in regular HIBE scheme [22]. The private key is set to $s_\epsilon H_1(\text{Hospital} \circ \text{Hospital})$. Bob's key can be computed only by his ancestors in the MHIBE scheme. An MHIBE scheme needs to be *joining-path-oblivious*. This means that encryption should be oblivious of the path from which the receiver and his ancestors acquire their private keys. Having the receiver's identity-set is sufficient to encrypt a message. For example, the sender does not need to know whether Bob obtains his keys from entity {(Hospital, ER), (Hospital, School, Manager)} or from entity {(Hospital, ER, Doctor), (Hospital, School)}.

5.2 Properties of Our MHIBE Implementation

Our fs-HIBE scheme naturally gives rise to an MHIBE scheme. In fs-HIBE, a message is encrypted under both an ID-tuple and the current time. This can be viewed as the encryption under two tuples, one being the current time. Therefore, the identities in MHIBE scheme capture a broader sense of meaning. The MHIBE scheme generalized from our fs-HIBE scheme supports dynamic joins and joining-path-oblivious encryption. More importantly, it is collusion-resistant, which cannot be achieved by using multiple separate HIBE [22] schemes. In our MHIBE implementation, a message encrypted under {(Hospital, ER, Doctor), (Hospital, School, Manager)} or {(Hospital, School, Manager), (Hospital, ER, Doctor)} requires different decryption keys. We note that in this scheme, the fact that a user holds the private key corresponding to multiple identities does not imply that he or she has the private key to any subset of identities.

We omit the details of MHIBE scheme (definition of security, description of scheme, and proof of security), as this is a direct generalization of fs-HIBE scheme. The complexities of various parameters of our MHIBE scheme are shown in Table 1 in Section 6.

6 Discussions

We analyze the complexity of our fs-HIBE scheme, the generalized MHIBE scheme, and the fs-BE scheme in Table 1 showing running time complexities and key sizes. Key generation time of fs-HIBE and MHIBE is the time to generate secret keys for a child node by the parent. Key generation time of fs-BE scheme is the running time of Reg algorithm. In our fs-HIBE scheme, the time periods correspond to leaf nodes of a binary tree, and the key update time is $\mathcal{O}(h \log N)$, where N is the total number of time periods and h is the length of an ID-tuple. Because of the node arrangement, the key generation time and key update time of our fs-HIBE scheme grows logarithmically with the total number of time periods N. Faster key update time ($\mathcal{O}(h)$) can be achieved, if the time periods are associated with *all* the nodes of the tree in a pre-order traversal, as in the fs-PKE scheme by Canetti *et al.* [13]. Because the realization of such an fs-HIBE scheme can be easily derived from the construction in Section 3.2, it is omitted in this paper. We show the optimized running time in Table 1.

Even dropping the joining-time-obliviousness requirement (as in the naive Scheme II of Section 1.3), our implementation cannot achieve a ciphertext with linear length $\mathcal{O}(h+\log N)$. It remains an interesting open question whether a general fs-HIBE scheme with linear complexity can be realized.

Parameters	fs-HIBE	MHIBE	fs-BE
Key generation time	$\mathcal{O}(h \log N)$	$\mathcal{O}(h^m)$	$\mathcal{O}(\log^3 E \log N)$
Encryption time	$\mathcal{O}(h \log N)$	$\mathcal{O}(h^m)$	$\mathcal{O}(r \log E \log N)$
Decryption time	$\mathcal{O}(h \log N)$	$\mathcal{O}(h^m)$	$\mathcal{O}(r + \log E \log N))$
Key update time	$\mathcal{O}(h)$	N/A	$\mathcal{O}(\log^3 E)$
Ciphertext length	$\mathcal{O}(h \log N)$	$\mathcal{O}(h^m)$	$\mathcal{O}(r \log E \log N)$
Public key size	$\mathcal{O}(h + \log N)$	$\mathcal{O}(hm)$	$\mathcal{O}(r \log E + \log N)$
Secret key size	$\mathcal{O}(h \log N)$	$\mathcal{O}(h^m)$	$\mathcal{O}(\log^3 E \log N)$

Table 1: Dependency of parameters of our fs-HIBE, MHIBE, and fs-BE schemes on the total number N of time periods, the length h of an ID-tuple, the number m of ID-tuples in an identity-set in MHIBE, the total number E of fs-BE users and the number r of actual revoked users in fs-BE scheme. Key generation time of fs-HIBE and MHIBE is the time to generate secret keys for a child node by the parent. Key generation time of fs-BE scheme is the running time of Reg algorithm.

7 Conclusion

The Multiple Hierarchical Identity-Based Encryption (MHIBE) scheme is an ID-Based encryption scheme for complex hierarchies. The generalization of a collusion-resistant MHIBE scheme from the Hierarchical Identity-Based Encryption (HIBE) scheme is significant, because MHIBE scheme conveniently lends itself to a wide range of applications that cannot be accomplished using HIBE schemes. To demonstrate this, we presented in detail a forward-secure HIBE scheme and a forward-secure Broadcast Encryption scheme. We also described the application of MHIBE in the access control paradigm. The forward-secure applications derived from our MHIBE scheme are joining-time-oblivious and support dynamic joins, which make them scalable.

8 Acknowledgements

We are grateful to Shai Halevi and Jonathan Katz for helpful discussions. The first author is thankful to Seth Proctor at Sun Microsystems Lab for his helpful comments.

References

[1] M. Abdalla, S. K. Miner, and C. Namprempre. Forward-secure threshold signature schemes. In *Topics in Cryptography — CT-RSA '01*, volume 2020 of *LNCS*, pages 441–456. Springer-Verlag, 2001.

[2] J. An, Y. Dodis, and T. Rabin. On the security of joint signature and encryption. In *Advances in Cryptology — Eurocrypt '02*, volume 2332 of *LNCS*, pages 83–107. Springer-Verlag, 2002.

[3] R. Anderson. Two remarks on public-key cryptology. Invited lecture, *4th ACM Conference on Computer and Communications Security*, 1997. Available at `http://www.cl.cam.ac.uk/ftp/users/rja14/`.

[4] M. Bellare and S. K. Miner. A forward-secure digital signature scheme. In *Advances in Cryptology — Crypto '99*, volume 1666 of *LNCS*, pages 431–448. Springer-Verlag, 1999.

[5] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 62–73. ACM, 1993.

[6] M. Bellare and B. Yee. Forward security in private-key cryptography. In *CT-RSA*, volume 2612 of *LNCS*, pages 1–18. Springer-Verlag, 2003.

[7] S. Berkovits. How to broadcast a secret. In *Advances in Cryptology — Eurocrypt '91*, volume 547 of *LNCS*, pages 535–541. Springer-Verlag, 1991.

[8] D. Boneh and X. Boyen. Secure identity based encryption without random oracles. To appear at Crypto '04. Available at `http://eprint.iacr.org/2004/173/`.

[9] D. Boneh and X. Boyen. Efficient selective-ID secure identity-based encryption without random oracles. In *Advances in Cryptology — Eurocrypt '04*, volume 3027 of *LNCS*, pages 223–238. Springer-Verlag, 2004.

[10] D. Boneh and M. K. Franklin. Identity-based encryption from the Weil pairing. Extended version of [11], `http://www.cs.stanford.edu/~dabo/papers/ibe.pdf`.

[11] D. Boneh and M. K. Franklin. Identity-based encryption from the Weil pairing. In *Advances in Cryptology — Crypto '01*, volume 2139 of *LNCS*, pages 213–229. Springer-Verlag, 2001.

[12] D. Boneh and A. Silverberg. Applications of multilinear forms to cryptography. *Contemporary Mathematics*, 324:71–90, 2003.

[13] R. Canetti, S. Halevi, and J. Katz. A forward-secure public-key encryption scheme. In *Advances in Cryptology — Eurocrypt '03*, volume 2656 of *LNCS*, pages 255–271. Springer-Verlag, 2003.

[14] W. Diffie, P. van Oorschot, and W. Wiener. Authentication and authenticated key exchanges. In *Designs, Codes and Cryptography*, volume 2, pages 107–125, 1992.

[15] Y. Dodis and N. Fazio. Public-key broadcast encryption for stateless receivers. In *Digital Rights Management — DRM '02*, volume 2696 of *LNCS*, pages 61–80. Springer, 2002.

[16] Y. Dodis and N. Fazio. Public-key trace and revoke scheme secure against adaptive chosen ciphertext attack. In *Public Key Cryptography — PKC '03*, volume 2567 of *LNCS*, pages 100–115. Springer-Verlag, 2003.

[17] Y. Dodis and J. Katz. Chosen ciphertext security of multiple encryption. Manuscript. Rump Session at Crypto 2003, 2003.

[18] A. Fiat and M. Naor. Broadcast encryption. In *Advances in Cryptology — Crypto '93*, volume 773 of *LNCS*, pages 480–491. Springer-Verlag, 1993.

[19] E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *Advances in Cryptology — Crypto '99*, volume 1666 of *LNCS*, pages 537–554. Springer-Verlag, 1999.

[20] A. Garay, J. Staddon, and A. Wool. Long-lived broadcast encryption. In *Advances in Cryptology — Crypto 2000*, volume 1880 of *LNCS*, pages 333–352. Springer-Verlag, 2000.

[21] C. Günther. An identity-based key exchange protocol. In *Advances in Cryptology — Eurocrypt '89*, volume 434 of *LNCS*, pages 29–37. Springer-Verlag, 1989.

[22] C. Gentry and A. Silverberg. Hierarchical ID-based cryptography. In *Advances in Cryptology — Asiacrypt '02*, volume 2501 of *LNCS*, pages 548–566. Springer-Verlag, 2002.

[23] M. T. Goodrich, J. Z. Sun, and R. Tamassia. Efficient tree-based revocation in groups of low-state devices. In *Advances in Cryptology - Crypto '04*, LNCS.

[24] D. Halevy and A. Shamir. The LSD broadcast encryption scheme. In *Advances in Cryptology — Crypto '02*, volume 2442 of *LNCS*, pages 47–60. Springer-Verlag, 2002.

[25] J. Holt, R. Bradshaw, K. E. Seamons, and H. Orman. Hidden credentials. In *Proceedings of the 2nd ACM Workshop on Privacy in the Electronic Society*, pages 1–8, October 2003.

[26] J. Horwitz and B. Lynn. Toward hierarchical identity-based encryption. In *Advances in Cryptology — Eurocrypt '02*, volume 2332 of *LNCS*, pages 466–481. Springer-Verlag, 2002.

[27] J. Katz. A forward-secure public-key encryption scheme. Cryptology ePrint Archive, Report 2002/060, 2002. `http://eprint.iacr.org/`.

[28] C. Kim, Y. Hwang, and P. Lee. An efficient public key trace and revoke scheme secure against adaptive chosen ciphertext attack. In *Advances in Cryptology — Asiacrypt 2003*, volume 2894 of *LNCS*, pages 359–373. Springer-Verlag, 2003.

[29] M. Luby and J. Staddon. Combinatorial bounds for broadcast encryption. In *Advances in Cryptology — Eurocrypt '98*, volume 1403 of *LNCS*, pages 512–526. Springer-Verlag, 1998.

[30] T. Malkin, D. Micciancio, and S. K. Miner. Efficient generic forward-secure signatures with an unbounded number of time periods. In *Advances in Cryptology — Eurocrypt '02*, volume 2332 of *LNCS*, pages 400–417. Springer-Verlag, 2002.

[31] D. Naor, M. Naor, and J. Lotspiech. Revocation and tracing schemes for stateless receivers. In *Advances in Cryptology — Crypto '01*, volume 2139 of *LNCS*, pages 41–62. Springer-Verlag, 2001.

[32] M. Naor and B. Pinkas. Efficient trace and revoke schemes. In *Financial Cryptography — FC '00*, volume 1962 of *LNCS*, pages 1–20. Springer-Verlag, 2000.

[33] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29, Number 2:38–47, 1996.

[34] R. Tamassia, D. Yao, and W. H. Winsborough. Role-based cascaded delegation. In *Proceedings of the ACM Symposium on Access Control Models and Technologies (SACMAT '04)*, pages 146 – 155. ACM Press, June 2004.

[35] W. Tzeng and Z. Tzeng. A public-key traitor tracing scheme with revocation using dynamics shares. In *Public Key Cryptography — PKC '01*, volume 1992 of *LNCS*, pages 207–224.

Springer-Verlag, 2001.

[36] B. R. Waters. Efficient identity-based encryption without random oracles. Cryptology ePrint Archive, Report 2004/180, 2004. http://eprint.iacr.org/.

[37] C. Wong, M. Gouda, and S. Lam. Secure group communications using key graphs. In *Proceedings of the ACM SIGCOMM '98*, pages 68 – 79. ACM Press, 1998.

A fs-HIBE: Verification of Correctness

We now verify that decryption of fs-HIBE in Section 3.2 is performed correctly. When encrypting, we have $\hat{e}(Q, H_1(i_1 \circ \mathrm{ID}_1))^r = \hat{e}(P, H_1(i_1 \circ \mathrm{ID}_1))^{rs_\epsilon}$. Recall $U_{kj} = rP_{kj} = rH_1(i|_k \circ \mathrm{ID}_1 \ldots \mathrm{ID}_j)$.

$$\begin{aligned}
\frac{\hat{e}(U_0, S_{i,h})}{\Pi_{k=1}^{l}\Pi_{j=2}^{h}\hat{e}(Q_{k,j}, U_{k,j})\Pi_{k=2}^{l}\hat{e}(Q_{k,1}, U_{k,1})} &= \frac{\hat{e}(rP, s_\epsilon P_{1,1} + \sum_{k=1}^{l}\sum_{j=2}^{h} s_{k,j}P_{k,j} + \sum_{k=2}^{l} s_{k,1}P_{k,1})}{\Pi_{k=1}^{l}\Pi_{j=2}^{h}\hat{e}(s_{k,j}P, U_{k,j})\Pi_{k=2}^{l}\hat{e}(s_{k,1}P, U_{k,1})} \\
&= \frac{\hat{e}(P, P_{1,1})^{rs_\epsilon}\Pi_{k=1}^{l}\Pi_{j=2}^{h}\hat{e}(rP, s_{k,j}P_{k,j})\Pi_{k=2}^{l}\hat{e}(rP, s_{k,1}P_{k,1})}{\Pi_{k=1}^{l}\Pi_{j=2}^{h}\hat{e}(s_{k,j}P, rP_{k,j})\Pi_{k=2}^{l}\hat{e}(s_{k,1}P, rP_{k,j})} \\
&= \hat{e}(P, P_{1,1})^{rs_\epsilon} = \hat{e}(P, H_1(i_1 \circ \mathrm{ID}_1))^{rs_\epsilon}
\end{aligned}$$

B fs-HIBE: One-way Identity-Based Encryption

As in HIBE [22] schemes, the proof of security of our fs-HIBE system makes use of a weaker notion of security, which is one-way encryption (OWE). One-way security differs from chosen-ciphertext security in that decryption queries are not allowed and the challenge is guessing the message, as opposed to distinguishing ciphertexts of two messages. We say an fs-HIBE scheme is one-way if no polynomial time adversary has a non-negligible advantage against the challenger in the following game. (Phase 1 is omitted for nonadaptive adversary.)

Setup: The challenger takes a security parameter k and runs the Root Setup algorithm. It gives the adversary the resulting system parameters *params*. It keeps the root keys to itself.

Phase 1: The adversary makes lower-level setup queries as in the Phase 1 of chosen-ciphertext security in Section 2.3[4].

Challenge: Once the adversary decides that Phase 1 is over, it outputs a new time period t and a new ID-tuple on which it wishes to be challenged. The challenger picks a random $M \in \mathcal{M}$ and sets $C = \textsc{Encrypt}(params, t, \text{ID-tuple}, M)$. It sends C as a challenge to the adversary.

Phase 2: The adversary issues more lower-level setup queries. The restrictions are the same as in the queries at Phase 2 of chosen-ciphertext security[5]. The challenger responds as in Phase 1.

Guess: The adversary outputs a guess $M' \in \mathcal{M}$. The adversary wins the game if $M = M'$. We define the adversary's advantage in attacking the scheme to be $\Pr[M = M']$.

C fs-HIBE with CCA2 Security

Fujisaki-Okamoto padding [19] is used to convert the fs-HIBE scheme with one-way security to an fs-HIBE scheme that is chosen ciphertext secure in the random oracle model as follows. The basic fs-HIBE scheme refers to the scheme described in Section 3.2.

Root Setup: As in the basic fs-HIBE scheme, but in addition choose hash function $H_3 : \{0,1\}^n \times \{0,1\}^n \to \mathbb{Z}_q$ and $H_4 : \{0,1\} \to \{0,1\}^n$.
Compute Next: As in the basic fs-HIBE scheme.
Lower-level Setup: As in the basic fs-HIBE scheme.
Encrypt(i, ($\mathrm{ID}_1, \ldots, \mathrm{ID}_h$), M): To encrypt $M \in \mathcal{M}$ with the time i and ID-tuple $(\mathrm{ID}_1, \ldots, \mathrm{ID}_h)$, do the following:

[4] In the random oracle model, the adversary also makes public key queries.
[5] More public key queries are made in the random oracle model.

1. Compute $P_{k,j} = H_1(i|_k \circ \mathrm{ID}_1 \dots \mathrm{ID}_j)$, for $1 \le k \le l$, and $1 \le k \le h$;
2. Choose a random $\sigma \in \{0,1\}^n$;
3. Set $r = H_3(\sigma, M)$;
4. Set the ciphertext C to be: $(rP, rP_{2,1}, \dots, rP_{l,1}, rP_{1,2}, \dots, rP_{l,2}, \dots rP_{1,h}, \dots, rP_{l,h}, \sigma \oplus H_2(g^r), M \oplus H_4(\sigma))$ where $g = \hat{e}(Q, P_{1,1}) \in G_2$.

Decrypt(i, $(\mathrm{ID}_1, \dots, \mathrm{ID}_h), C$): Let $C \in \mathcal{C}$ be the ciphertext encrypted using the time i and the ID-tuple.

1. Parse C as $(U_0, U_{2,1}, \dots, U_{l,1}, U_{1,2}, \dots, U_{l,2}, \dots, U_{1,h}, \dots, U_{l,h}, V, W)$.
2. If $(U_0, U_{2,1}, \dots, U_{l,1}, U_{1,2}, \dots, U_{l,2}, \dots, U_{1,h}, \dots, U_{l,h}) \notin G_1^{l \times h}$, reject the ciphertext. Decryption of C proceeds as follows:
   (a) Compute $V \oplus H_2(\frac{\hat{e}(U_0, S_{i,h})}{g}) = \sigma$, where g is: $\Pi_{k=1}^{l} \Pi_{j=2}^{h} \hat{e}(Q_{k,j}, U_{k,j}) \Pi_{k=2}^{l} \hat{e}(Q_{k,1}, U_{k,1})$.
   (b) Compute $W \oplus H_4(\sigma) = M$;
   (c) Set $r = H_3(\sigma, M)$ and test if $(U_0, U_{2,1}, \dots, U_{l,1}, U_{1,2}, \dots, U_{l,2}, \dots, U_{1,h}, \dots, U_{l,h}, V)$ is a basic fs-HIBE encryption of M using r, time i, and $(\mathrm{ID}_1, \dots, \mathrm{ID}_h)$; if not, it rejects the ciphertext;
   (d) Output M as the decryption of C.

D fs-HIBE: Proof of Security

The proof of one-way security against nonadaptive chosen time and target adversary is given in Section D.2. **Phase 1** is omitted for the nonadaptive adversary in the one-way security definition. The proof of one-way security against adaptive chosen-target adversary is given in Section D.3, which may be converted to a proof of chosen-ciphertext security using similar proof techniques as shown in IBE [10] scheme. In our proofs, we use similar arguments to the ones in the proofs of IBE [11] and HIBE [22] scheme to show that the adversary $\mathcal{B}$ who uses an fs-HIBE adversary $\mathcal{A}$ to break BasicPub (shown below) does not abort with non-negligible probability.

D.1 BasicPub

To analyze the security of fs-HIBE scheme, we make use of a public-key encryption scheme called BasicPub [11], which was presented by Boneh and Franklin in the proof of their IBE scheme. Boneh and Franklin showed that BasicPub is one-way secure in the random oracle model under the BDH assumption [11]. To show that our fs-HIBE scheme is secure, we give a reduction from breaking fs-HIBE scheme to BasicPub. We use the techniques of Gentry and Silverberg in the proof of HIBE [22] scheme.

BasicPub is a public-key encryption scheme, specified by three algorithms: Generate Key, Encrypt and Decrypt.

Generate Key:

1. Run $\mathcal{IG}$ on input k to generate two groups $\mathbb{G}_1, \mathbb{G}_2$ of the same prime order q and a bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_2$. Choose an arbitrary generator $P \in \mathbb{G}_1$.
2. Pick random $s_\epsilon \in \mathbb{Z}_q$ and set $Q = s_\epsilon P$.
3. Pick a random point $P_1 \in \mathbb{G}_1$.
4. Choose a cryptographic hash function $H_2 : \mathbb{G}_2 \to \{0,1\}^n$.

The message space is $\mathcal{M} = \{0,1\}^n$. The ciphertext space is $\mathcal{C} = \mathbb{G}_1 \times \{0,1\}^n$. The public key is $(\mathbb{G}_1, \mathbb{G}_2, \hat{e}, P, Q, P_1, H_2)$. The private key is $S_1 = s_\epsilon P_1$.

ENCRYPT: To encrypt $M \in \mathcal{M}$, do the following:

1. Choose a random $r \in \mathbb{Z}_q$.
2. Set the ciphertext to be: $C = (rP, M \oplus H_2(g^r))$ where $g = \hat{e}(Q, P_1) \in \mathbb{G}_2$.

DECRYPT: Let $C = (U, V) \in \mathcal{C}$ be the ciphertext. To decrypt C, compute: $V \oplus H_2(\hat{e}(U, S_1)) = M$.

D.2 fs-HIBE scheme – Targeting a specific ID-tuple and time period

Lemma D.1. *Let H_1 be a random oracle from $\{0,1\}^*$ to $\mathbb{G}_1$. Let $\mathcal{A}$ be a nonadaptive adversary against the one-way secure fs-HIBE scheme that makes a finite number of lower-level setup queries and has advantage ϵ against fs-HIBE for some fixed time t_0 and ID-tuple$_0$. Then there is an OWE adversary $\mathcal{B}$ that has advantage at least ϵ against BasicPub. Its running time is $\mathcal{O}(time(\mathcal{A}))$.*

Proof: Let us construct an OWE adversary $\mathcal{B}$ that uses $\mathcal{A}$ to gain advantage ϵ against BasicPub. The game between the challenger and the adversary $\mathcal{B}$ starts with the challenger first generating a random public key by running the key generation algorithm of BasicPub. The result is a public key $K_{pub} = (\mathbb{G}_1, \mathbb{G}_2, \hat{e}, P, Q, P_1, H_2)$, with $Q = s_\epsilon P$, and a private key $S_1 = s_\epsilon P_1$. The challenger then picks a random plaintext $M \in \mathcal{M}$ and encrypts M using the encryption algorithm of BasicPub. It gives K_{pub} and the resulting ciphertext $C = (U, V)$ to algorithm $\mathcal{B}$. Algorithm $\mathcal{B}$ is supposed to output a guess for M.

Let the ID-tuple$_0$ and time period t_0 be the ID-tuple and the time period for which $\mathcal{A}$ has an advantage against fs-HIBE scheme. Then $\mathcal{A}$ and $\mathcal{B}$ agree to use the ID-tuple and the time period.

Let us first give an overview of what follows. $\mathcal{B}$ has essentially two goals to accomplish in order to succeed in this game. One is to be able to answer H_1 queries and lower-level setup queries made by $\mathcal{A}$, without knowing s_ϵ or $s_\epsilon P_1$. Recall that SK_{t_i,h_i} denotes the private key associated with time t_i and ID-tuple $(\text{ID}_1 \ldots \text{ID}_{h_i})$. It consists of a series of $(sk_{t_i,h_i}, \{sk_{w,h_i}\})$ values, each of which contains an S_{w,h_i} and $\mathcal{Q}_{w,h_i} = \{Q_{k,j}$ for all $1 \le k \le |w|$ and $1 \le j \le h_i\}$. A private key SK_{t_i,h_i} is *valid* if the following holds. Each $sk_{w,h}$ in SK_{t_i,h_i}, where w is any string, has to have the correct distribution, that is, $S_{w,h_i} = S_{w,(h_i-1)} + \sum_{k=1}^{|w|} s_{k,h_i} H_1(w|_k \circ \text{ID}_1 \ldots \text{ID}_{h_i}) = S_{(w|_{|w|-1}),h_i} + \sum_{j=1}^{h_i} s_{|w|,j} H_1(w \circ \text{ID}_1 \ldots \text{ID}_j) = \sum_{k=1}^{|w|} \sum_{j=1}^{h_i} s_{k,j} H_1(w|_k \circ \text{ID}_1 \ldots \text{ID}_j)$, and $Q_{k,j} = s_{k,j} P$ for some $s_{k,j} \in \mathbb{Z}_q$, for all $1 \le k \le |w|$ and $1 \le j \le h_i$. In the above expression, we let $S_{0,j}$ and $S_{k,0}$ be the identity element in $\mathbb{G}_1$, for all j and k. Recall $w|_k$ denotes the first k-bit prefix of w, $w_1 \ldots w_k$.

The other goal of $\mathcal{B}$ in this proof is to convert his challenge ciphertext C into an fs-HIBE ciphertext C', so that it can be correctly decrypted by a private key of associated with the target t_0 and ID-tuple$_0$. Adversary $\mathcal{B}$ interacts with $\mathcal{A}$ as follows:

Setup: $\mathcal{B}$ gives $\mathcal{A}$ the fs-HIBE system parameters $params = (\mathbb{G}_1, \mathbb{G}_2, \hat{e}, P, Q, H_1, H_2)$. H_1 is a random oracle controlled by $\mathcal{B}$. Let $N = 2^l$ be the total number of time periods. The target time period t_0 may be represented as $t_{0,1} \ldots t_{0,l}$.

H_1-**queries:** At any time, algorithm $\mathcal{A}$ can query the random oracle H_1. This oracle will be used to store the Point-tuple$_i = \{T_{i,k,j} = H_1(t_i|_k \circ \text{ID}_{i,1} \ldots \text{ID}_{i,j}) : 1 \le k \le l, 1 \le j \le h_i\}$, Secret-tuple$_i$ $\{s_{i,k,j} : 1 \le k \le l, 1 \le j \le h_i\}$, and Scalar-tuple$_i$ $\{b_{i,k,j} : 1 \le k \le l, 1 \le j \le h_i\}$ corresponding to Time-ID-pair$_i$ $(t_i, (\text{ID}_{i,1} \ldots \text{ID}_{i,h_i}))$. In responding to these queries, algorithm $\mathcal{B}$ maintains a list H_1^{list} containing tuples of the form (Time-ID-pair$_i$, Point-tuple$_i$, Scalar-tuple$_i$, Secret-tuple$_i$). This list is initially empty.

The time period t_0 together with the ID-tuple$_0$ creates a Time-ID-pair, which are denoted as Time-ID-pair$_0$. Algorithm $\mathcal{B}$ adds Time-ID-pair$_0$ with to H_1^{list} as follows:

1. sets $s_{0,1,1} = 1$;
2. picks a random $s_{0,k,j} \in \mathbb{Z}_q$, for $1 \le k \le l$ and $1 \le j \le h_0$ except for $k = 1$ and $j = 1$;
3. picks a random $b_{0,k,j} \in \mathbb{Z}_q$, for $1 \le k \le l$ and $1 \le j \le h_0$;
4. sets $T_{0,1,1} = b_{0,1,1}P_1$, and $T_{0,k,j} = b_{0,k,j}P$ for $1 \le k \le l$ and $1 \le j \le h_0$ except for $k = 1$ and $j = 1$.

Algorithm $\mathcal{B}$ then puts the followings in H_1^{list}: $((t_{0,1}, \dots, t_{0,l}), (\mathrm{ID}_{0,1}, \dots, \mathrm{ID}_{0,h_0}))$, $(T_{0,1,1}, \dots, T_{0,1,h}, \dots, T_{0,l,1}, \dots, T_{0,l,h_0})$, $(b_{0,1,1}, \dots, b_{0,1,h_0}, \dots, b_{0,l,1}, \dots, b_{0,l,h_0})$, $(s_{0,1,1}, \dots, s_{0,1,h_0}, \dots, s_{0,l,1}, \dots, s_{0,l,h_0})$.

Once that is done, $\mathcal{A}$ queries H_1 about Time-ID-pair$_i$ $(t_i, (\mathrm{ID}_{i,1} \dots \mathrm{ID}_{i,h_i}))$. Let us first briefly describe how $\mathcal{B}$ answers this query. $\mathcal{B}$ has to be ready in case $\mathcal{A}$ makes a lower-level setup query for this Time-ID-pair$_i$. So an appropriate SK_{t_i,h_i} must be concocted. Recall that SK_{t_i,h_i} associated with time t_i and ID-tuple$_i$ $(\mathrm{ID}_{i,1}, \dots, \mathrm{ID}_{i,h_i})$ consists of a series of $(sk_{t_i,h_i}, \{sk_{w,h_i}\})$ values, each of which contains an S_{w,h_i} and $\mathcal{Q}_{w,h_i} = \{Q_{k,j}$ for all $1 \le k \le |w|$ and $1 \le j \le h_i\}$.

For Time-ID-pair$_i$ $(t_i, (\mathrm{ID}_{i,1} \dots \mathrm{ID}_{i,h_i}))$, $\mathcal{B}$ answers H_1 query by computing the point-tuple values $T_{i,k,j} = H_1(t_i|_k \circ \mathrm{ID}_{i,1} \dots \mathrm{ID}_{i,j}) : 1 \le k \le l, 1 \le j \le h_i\}$. $\mathcal{B}$ also chooses Scalar-tuple$_i$ and Secret-tuple$_i$ for Time-ID-pair$_i$. Scalar-tuple$_i$, $\{b_{i,j,k}\}$ for all $1 \le k \le l, 1 \le j \le h_i$, is a set of values chosen from $\mathbb{Z}_q$. Secret-tuple$_i$, $\{s_{i,j,k}\}$ for all $1 \le k \le l, 1 \le j \le h_i$, is another set of values chosen from $\mathbb{Z}_q$. $s_{i,1,1} = 1$ for all i. Some of the values in Scalar-tuple$_i$ and Secret-tuple$_i$ for Time-ID-pair$_i$ are inherited from some existing Time-ID-pair in H_1^{list}, and others are freshly generated by $\mathcal{B}$.

At lower-level setup queries $\mathcal{B}$ uses these values to generate the key SK for some Time-ID-pair. With in key SK, the value S_{t_i,h_i} is computed as the sum of the products of the values in the Secret-tuple and Point-tuple. Because of possible overlaps among the Time-ID-pairs, some of the values in some existing Point-tuple, Scalar-tuple and Secret-tuple associated with Time-ID-pair$_i$ are used for Time-ID-pair$_{i'}$, where $i' > i$.

In order for $\mathcal{B}$ to generate valid S values without knowing s_ϵ or $s_\epsilon P_1$, $\mathcal{B}$ does a trick at choosing one of the T values for Time-ID-pair$_i$, so that this special T has a term with the negative sign that contains s_ϵ. These two terms with s_ϵ are conveniently canceled out with each other when computing S_{t_i,h_i}. $\mathcal{B}$ has to be careful at choosing this special T, which contains s_ϵ with the negative sign, since only one such T is needed.

However, for a Time-ID-pair $(t, (\mathrm{ID}_1, \dots, \mathrm{ID}_h))$, cancellations may occur twice. This is the case when the adversary $\mathcal{A}$ first queries $(t|_1, (\mathrm{ID}_1, \dots, \mathrm{ID}_h))$, where $t|_1$ is the first bit of a time period t. Then $\mathcal{A}$ queries $((t|_1, \dots, t|_l), \mathrm{ID}_1)$, where $l = |t|$, followed by the query $((t|_1, \dots, t|_l), (\mathrm{ID}_1, \dots, \mathrm{ID}_h))$. Note that the adversary $\mathcal{A}$ may not be able to directly query about $t|_1$, but in order to answer her (other) queries, $\mathcal{B}$ may need to compute the key associated with $(t|_1, (\mathrm{ID}_1, \dots, \mathrm{ID}_h))$ as the node key $sk_{1,h}$, which is for deriving future secret keys in fs-HIBE. Therefore, $\mathcal{B}$ has to handle the case when the cancellation occurs twice. This is done by choosing a T value that contains a positive s_ϵ, and placing it at the position where both cancellations occur.

In order to see if the trick of choosing special T values is necessary for some Time-ID-pair$_i$, $\mathcal{B}$ compares Time-ID-pair$_i$ to both the target Time-ID-pair$_0$ and the existing ones in H_1^{list} as follows.

For each Time-ID-pair$_g$ in H_1^{list}, let v be the maximal such that $(t_{i,1} \circ \mathrm{ID}_{i,1}, \dots, t_{i,v} \circ \mathrm{ID}_{i,1}) = (t_{g,1} \circ \mathrm{ID}_{g,1}, \dots, t_{g,v} \circ \mathrm{ID}_{g,1})$, where $t_{i,1}$ denotes the first bit of t_i, etc. And let w be the maximal such that $(\mathrm{ID}_{i,1} \circ t_{i,1}, \dots, \mathrm{ID}_{i,w} \circ t_{i,1}) = (\mathrm{ID}_{g,1} \circ t_{g,1}, \dots, \mathrm{ID}_{g,w} \circ t_{g,1})$. For $1 \le k \le v$ and $1 \le j \le w$, if $T_{i,v,w}$ is null, set $T_{i,k,j} = T_{g,k,j}$, $b_{i,k,j} = b_{g,k,j}$, and $s_{i,k,j} = s_{g,k,j}$.

Once the above comparison is done, let v be the maximal such that $T_{i,v,1}$ is not null ($T_{i,v,1}$ is given a value from the above), and let w be the maximal such that $T_{i,1,w}$ is not null.

Let $x \leq v$ be the maximal such that $(t_{i,1} \circ \mathrm{ID}_{i,1}, \ldots, t_{i,x} \circ \mathrm{ID}_{i,1}) = (t_{0,1} \circ \mathrm{ID}_{0,1}, \ldots, t_{0,x} \circ \mathrm{ID}_{0,1})$, and $y \leq w$ be the maximal such that $(\mathrm{ID}_{i,1} \circ t_{i,1}, \ldots, \mathrm{ID}_{i,y} \circ t_{i,1}) = (\mathrm{ID}_{0,1} \circ t_{0,1}, \ldots, \mathrm{ID}_{0,y} \circ t_{0,1})$.

The purpose of these comparisons is to see if the trick of choosing the special T is needed. The details are as follows.

1. If $0 < x < v$ and $0 < y < w$ and $T_{i,v,w}$ is null, algorithm $\mathcal{B}$ picks a random $b_{i,v,w} \in \mathbb{Z}_q$, sets $s_{i,v,w} = b_{i,v,w}^{-1}$ and $T_{i,v,w} = b_{i,v,w} b_{i,1,1} P_1$.
2. If $0 < v = x < l$ (implying that $T_{i,v+1,1}$ is null), algorithm $\mathcal{B}$ picks a random $b_{i,v+1,1} \in \mathbb{Z}_q$ and a random $s_{i,v+1,1} \in \mathbb{Z}_q$, sets $T_{i,v+1,1} = b_{i,v+1,1} P - s_{i,v+1,1}^{-1} b_{i,1,1} P_1$.
3. If $0 < w = y < h_i$ (implying that $T_{i,1,w+1}$ is null), algorithm $\mathcal{B}$ picks a random $b_{i,1,w+1} \in \mathbb{Z}_q$ and a random $s_{i,1,w+1} \in \mathbb{Z}_q$, sets $T_{i,1,w+1} = b_{i,1,w+1} P - s_{i,1,w+1}^{-1} b_{i,1,1} P_1$.
4. If $1 < v = x < l$ and $1 < w = y < h_i$, (implying that $T_{i,v+1,w+1}$ is null), algorithm $\mathcal{B}$ picks a random $b_{i,v+1,w+1} \in \mathbb{Z}_q$, sets $s_{i,v+1,w+1} = b_{i,v+1,w+1}^{-1}$, sets $T_{i,v+1,w+1} = b_{i,v+1,w+1} b_{i,1,1} P_1$.
5. For $1 \leq k \leq l$ and $1 \leq j \leq h_i$ and $T_{i,k,j}$ is null, algorithm $\mathcal{B}$ picks a random $b_{i,k,j} \in \mathbb{Z}_q$ and a random $s_{i,k,j} \in \mathbb{Z}_q$ except for $s_{i,1,1}$ which is set to 1, and sets $T_{i,k,j} = b_{i,k,j} P$.

Algorithm $\mathcal{B}$ then puts the following in H_1^{list}: $((t_{i,1}, \ldots, t_{i,l}), (\mathrm{ID}_{i,1}, \ldots, \mathrm{ID}_{i,h_i}))$, $(T_{i,1,1}, \ldots, T_{i,l,h_i})$, $(b_{i,1,1}, \ldots, b_{i,l,h_i})$, $(s_{i,1,1}, \ldots, s_{i,l,h_i})$. $\mathcal{B}$ returns $(T_{i,1,1}, \ldots, T_{i,l,h_i})$ to $\mathcal{A}$.

Note that $T_{i,k,j}$ is always chosen uniformly in $\mathbb{G}_1$ and is independent of $\mathcal{A}$'s view as required.

Using above methods, $\mathcal{B}$ also stores the corresponding tuples that are necessary to compute the node keys at lower-level setup queries. Recall for time t_i and ID-tuple$_i$ $(\mathrm{ID}_1, \ldots, \mathrm{ID}_{h_i})$, node keys are denoted as sk_{w,h_i}, where w are all the labels of the right sibling, if one exists, of each ancestor of the node labeled t_i in the complete binary tree. They can also be written as $\{sk_{(t_i|_{k-1}1),h_i}\}_{t_{i,k}=0}$. For each time node w, $\mathcal{B}$ compares $(w, \text{ID-tuple}_i)$ with existing tuples in H_1^{list}, the same way as comparing $(t_i, \text{ID-tuple}_i)$. Node keys together with the decryption key sk_{t_i,h_i} are what $\mathcal{B}$ needs to return in a lower-level setup query. We omit the details of how $\mathcal{B}$ prepares for node keys in this proof, as it is essentially the same as preparing the decryption key shown above.

Lower-Level Setup Queries: At any time, $\mathcal{A}$ may make a lower-level setup query on any time period t_i and ID-tuple$_i$, given $t_i > t_0$, or ID-tuple$_i \neq$ ID-tuple$_0$ and its ancestor. $\mathcal{B}$ responds to this query as follows:

1. Run the above algorithm for responding to H_1-queries to obtain the appropriate tuple (Time-ID-pair$_i$, Point-tuple$_i$, Scalar-tuple$_i$, Secret-tuple$_i$) in H_1^{list}.
2. Define $S_{i,t_i,h_i} = \sum_{k=1}^{l} \sum_{j=1}^{h_i} s_{i,k,j} s_\epsilon T_{i,k,j}$ where $s_{i,1,1} := 1$ for all i. $\mathcal{B}$ also gives $\mathcal{A}$ the corresponding $\{Q_{i,k,j} = s_{i,k,j} Q : 1 \leq k \leq l, 1 \leq j \leq h_i\}$.
   Let $sk_{i,t_i,h_i} = (S_{i,t_i,h_i}, \mathcal{Q}_{i,t_i,h_i})$, where $\mathcal{Q}_{i,t_i,h_i} = \{Q_{i,k,j} = s_{i,k,j} Q : 1 \leq k \leq l, 1 \leq j \leq h_i\}$. sk_{i,t_i,h_i} is what ID-tuple$_i$ needs to decrypt messages at t_i.
3. Using similar method for computing sk_{i,t_i,h_i}, $\mathcal{B}$ also computes the node keys $\{sk_{i,(t_i|_{k-1}1),h_i}\}_{t_i|_k=0}$ corresponding to time t_i and ID-tuple$_i$. It can be verified that those node keys have the correct distribution. Using these node keys $\mathcal{B}$ can derive keys SK that are associated with any time $t > t_i$ or descendants of ID-tuple$_i$ with the correct distribution.

We leave to the readers the verification that shows the private key, in particular S_{t_i,h_i} for Time-ID-pair$_i$ is computable by $\mathcal{B}$ and has the correct distribution. Note that $\mathcal{B}$ does not know s_ϵ nor $s_\epsilon P_1$.

Challenge: At any time, $\mathcal{A}$ may request a challenge ciphertext from $\mathcal{B}$ on the ID-tuple$_0$ and time period t_0. Let $C = (U, V)$ be the challenge ciphertext given to algorithm $\mathcal{B}$. $\mathcal{B}$ sets the fs-HIBE ciphertext C' to be

$$\begin{gathered}(b_{0,1,1}^{-1}U, b_{0,1,1}^{-1}b_{0,2,1}U, \ldots, b_{0,1,1}^{-1}b_{0,l,1}U,\\ b_{0,1,1}^{-1}b_{0,1,2}U, \ldots, b_{0,1,1}^{-1}b_{0,l,2}U,\\ \ldots\\ b_{0,1,1}^{-1}b_{0,1,h}U, \ldots, b_{0,1,1}^{-1}b_{0,l,h_0}U, V)\end{gathered}$$

$\mathcal{B}$ responds to $\mathcal{A}$ with the challenge C'. Note that C' is an fs-HIBE encryption of M under ID-tuple$_0$ and t_0 as required. To verify this, first observe that the decryption key corresponding to ID-tuple$_0$ and t_0 is $S' = s_\epsilon T_{0,1,1} + \sum_{j=2}^{h_0}\sum_{k=1}^{l} s'_{k,j}T_{0,k,j} + \sum_{k=2}^{l} s'_{k,1}T_{0,k,1}$ along with the additional information $\{s'_{k,j}P : 1 \le k \le l \text{ and } 1 \le j \le h_0\}$ (except $s'_{1,1}$) for some set $\{s'_{k,j} : 1 \le k \le l$ and $1 \le j \le h_0\}$ (except $s'_{1,1}$).

Second, observe that :

$$\begin{aligned}\frac{\hat{e}(b_{0,1,1}^{-1}U, S')}{\Pi_{k=1}^{l}\Pi_{j=2}^{h_0}\hat{e}(b_{0,1,1}^{-1}b_{0,k,j}U, s'_{k,j}P)\Pi_{k=2}^{l}\hat{e}(b_{0,1,1}^{-1}b_{0,k,1}U, s'_{k,1}P)} &= \hat{e}(b_{0,1,1}^{-1}U, s_\epsilon T_{0,1,1})\\ &= \hat{e}(b_{0,1,1}^{-1}U, s_\epsilon T_{0,1,1})\\ &= \hat{e}(b_{0,1,1}^{-1}U, s_\epsilon b_{0,1,1}P_1)\\ &= \hat{e}(U, s_\epsilon P_1)\end{aligned}$$

The correct decryption of C' is M.

Guess: Eventually, $\mathcal{A}$ outputs a guess M'. $\mathcal{B}$ outputs M' as its guess for the decryption of C.

Claim: $\mathcal{A}$'s view is identical to its view in the real attack. Furthermore, $\Pr[M = M'] \ge \epsilon$. The probability is over the random bits used by $\mathcal{A}$, $\mathcal{B}$ and the challenger.

Proof of Claim: All responses to H_1-queries are as in the real attack since each response is uniformly and independently distributed in $\mathbb{G}_1$. All responses to lower-level setup queries are valid. Finally, the challenge ciphertext C' given to $\mathcal{A}$ is the fs-HIBE encryption of the random plaintext M under the ID-tuple$_0$ and time period t_0 chosen by $\mathcal{A}$. Therefore, by the definition of the algorithm $\mathcal{A}$, it will output $M' = M$ with probability at least ϵ.

D.3 fs-HIBE scheme – Adaptively chosen target ID-tuple and time period

Lemma D.2. *Let H_1 be a random oracle from $\{0,1\}^*$ to $\mathbb{G}_1$. Let $\mathcal{A}$ be an adaptive adversary against the one-way secure fs-HIBE scheme that makes a finite number of lower-level setup queries and has advantage ϵ of targeting an fs-HIBE time and ID-tuple. Let h be the level of the ID-tuple and $l = \log_2 N$ where N is the total number of time periods. Then there is an OWE adversary $\mathcal{B}$ that has advantage at least $\epsilon((h+l)/e(2lq_E + h + l))^{(h+l)/2}$ against BasicPub and running time is $\mathcal{O}(time(\mathcal{A}))$.*

Proof. We show how to construct an OWE adversary $\mathcal{B}$ that uses $\mathcal{A}$ to gain advantage against BasicPub. The game between the challenger and the adversary $\mathcal{B}$ starts with the challenger first generating a random public key by running the key generation algorithm of BasicPub. The result is a public key $K_{pub} = (\mathbb{G}_1, \mathbb{G}_2, \hat{e}, P, Q, P_1, H_2)$, with $Q = s_\epsilon P$, and a private key $S_1 = s_\epsilon P_1$. The challenger then picks a random plaintext $M \in \mathcal{M}$ and encrypts M using the encryption algorithm of BasicPub. It gives K_{pub} and the resulting ciphertext $C = (U, V)$ to algorithm $\mathcal{B}$. Algorithm $\mathcal{B}$ is supposed to output a guess for M.

As in the proof for Lemma D.1, in **Phase 1** $\mathcal{B}$ answers lower-level setup queries for some time t_i and ID-tuple$_i$ made by $\mathcal{A}$ making use of coin flips. In computing private keys, $\mathcal{B}$ uses coin flips to guess if an ID-tuple and a time period will be chosen by $\mathcal{A}$ as the target. At **Challenge**, $\mathcal{A}$ picks a target time and ID-tuple. $\mathcal{B}$ runs lower-level setup for the targets, the coin flips of which determine whether $\mathcal{B}$ can continue the game or not. If $\mathcal{B}$ can continue to play, it converts his challenge ciphertext C into an fs-HIBE ciphertext and gives it to $\mathcal{A}$. In **Phase 2**, adversary $\mathcal{B}$ answers more lower-level setup queries the way it did in Phase 1. Eventually, $\mathcal{A}$ outputs a guess, which is used as the guess by $\mathcal{B}$.

Similar to the proof for nonadaptive adversary in Section D.2, in order to answer Lower-level Setup queries, $\mathcal{B}$ does a trick to cancel the term with the unknown s_ϵ in the private key. Only here, $\mathcal{B}$ decides whether to do the trick or not by flipping coins. For the case when cancellations occur twice, $\mathcal{B}$ puts a positive term that contains s_ϵ at a proper position.

Adversary $\mathcal{B}$ interacts with $\mathcal{A}$ as follows:

Setup: $\mathcal{B}$ gives $\mathcal{A}$ the fs-HIBE system parameters $params = (\mathbb{G}_1, \mathbb{G}_2, \hat{e}, P, Q, H_1, H_2)$. H_1 is a random oracle controlled by $\mathcal{B}$. Let $N = 2^l$ be the total number of time periods.
H_1**-queries:** At any time, algorithm $\mathcal{A}$ can query the random oracle H_1. Essentially, this oracle will be used to store the Point-tuple$_i$ = $\{T_{i,k,j} = H_1(t_i|_k \circ \mathrm{ID}_{i,1} \dots \mathrm{ID}_{i,j}) : 1 \le k \le l, 1 \le j \le h_i\}$ corresponding to Time-ID-pair$_i$ $(t_i, (\mathrm{ID}_{i,1} \dots \mathrm{ID}_{i,h_i}))$. In responding to these queries, algorithm $\mathcal{B}$ maintains a list H_1^{list} containing tuples of the form (Time-ID-pair$_i$, Point-tuple$_i$, Scalar-tuple$_i$, Secret-tuple$_i$, ID-Coin-tuple$_i$, Time-Coin-tuple$_i$). For the i-th query, Time-ID-pair$_i$, Point-tuple$_i$, Scalar-tuple$_i$, and Secret-tuple$_i$ may be matrices, whereas ID-Coin-tuple$_i$ is an array of length h_i and Time-Coin-tuple$_i$ is an array of length l. The sizes of the Point-tuple, secret-tuple, and Scalar-tuple are the length of ID-tuple times l. H_1^{list} list is initially empty.

$\mathcal{A}$ queries H_1^{list} about Time-ID-pair$_i$ $(t_i, (\mathrm{ID}_{i,1} \dots \mathrm{ID}_{i,h_i}))$. For the overlapping portion of the tuples, the corresponding items in H_1^{list} are copied as shown next. For each Time-ID-pair$_g$ in H_1^{list}, let v be the maximal such that $(t_{i,1} \circ \mathrm{ID}_{i,1}, \dots, t_{i,v} \circ \mathrm{ID}_{i,1}) = (t_{g,1} \circ \mathrm{ID}_{g,1}, \dots, t_{g,v} \circ \mathrm{ID}_{g,1})$, and let w be the maximal such that $(\mathrm{ID}_{i,1} \circ t_{i,1}, \dots, \mathrm{ID}_{i,w} \circ t_{i,1}) = (\mathrm{ID}_{g,1} \circ t_{g,1}, \dots, \mathrm{ID}_{g,w} \circ t_{g,1})$. For $1 \le k \le v$ and $1 \le j \le w$, if $T_{i,v,w}$ is null, set Point-tuple $T_{i,k,j} = T_{g,k,j}$, Scalar-tuple $b_{i,k,j} = b_{g,k,j}$, Secret-tuple $s_{i,k,j} = s_{g,k,j}$, ID-Coin-tuple $c_{i,j} = c_{g,j}$, and Time-Coin-tuple $c'_{i,k} = c'_{g,k}$.

Once the above comparison is done, let v be the maximal such that $T_{i,v,1}$ is not null ($T_{i,v,1}$ is given a value from the above), and let w be the maximal such that $T_{i,1,w}$ is not null.

$\mathcal{B}$ computes the values for the rest part as follows.

1. If $0 \le v < l$ (implying that $T_{i,v+1,1}$ is null), for $k \in (v, l]$, algorithm $\mathcal{B}$ does the follows:
   (a) Pick a random $b_{i,k,1} \in \mathbb{Z}_q$;
   (b) If $k = 1$, generate a random time coin $c'_{i,1} \in \{0, 1\}$ so that $\Pr[c'_{i,1} = 1] = \delta$ for some δ that will be determined later. If $c'_{i,1} = 1$, set $T_{i,1,1} = b_{i,1,1}P_1$; else set $T_{i,1,1} = b_{i,1,1}P$. Set $s_{i,1,1} = s_\epsilon$.

(c) Else if $c'_{i,k-1} = 1$, generate a random time coin $c'_{i,k}$. If $c'_{i,k} = 1$, pick a random $s_{i,k,1} \in \mathbb{Z}_q$ and set $T_{i,k,1} = b_{i,k,1}P$; else choose a random $p_{i,k,1} \in \mathbb{Z}_q$, set $T_{i,k,1} = b_{i,k,1}P - p_{i,k,1}^{-1} b_{i,1,1}P_1$, and define $s_{i,k,1} = p_{i,k,1}s_\epsilon$ ($\mathcal{B}$ does not have to compute $s_{i,k,1}$).

(d) Else if $c'_{i,k-1} = 0$, set $c'_{i,k} = 0$, pick a random $s_{i,k,1} \in \mathbb{Z}_q$ and set $T_{i,k,1} = b_{i,k,1}P$.

2. If $1 \leq w < h_i$ (implying that $T_{i,1,w+1}$ is null), for $j \in (w, h_i]$, algorithm $\mathcal{B}$ does the follows:
   (a) Pick a random $b_{i,1,j} \in \mathbb{Z}_q$;
   (b) Set $c_{i,1} = c'_{i,1}$. If $c_{i,j-1} = 1$, generate a random coin $c_{i,j}$ as in the above step 1b. If $c_{i,j} = 1$, pick a random $s_{i,1,j} \in \mathbb{Z}_q$ and set $T_{i,1,j} = b_{i,1,j}P$; else choose a random $p_{i,1,j} \in \mathbb{Z}_q$, set $T_{i,1,j} = b_{i,1,j}P - p_{i,1,j}^{-1} b_{i,1,1}P_1$, and define $s_{i,1,j} = p_{i,1,j}s_\epsilon$ ($\mathcal{B}$ does not have to compute $s_{i,1,j}$).
   (c) If $c_{i,j-1} = 0$, set $c_{i,j} = 0$, pick a random $s_{i,1,j} \in \mathbb{Z}_q$ and set $T_{i,1,j} = b_{i,1,j}P$.
3. If $c_{i,1} = 1$ (i.e. $c'_{i,1} = 1$), let $1 < x \leq l$ be the minimal such that $c'_{i,x} = 0$, and let $1 < y \leq h_i$ be the minimal such that $c_{i,y} = 0$. If such x and y both exist and $T_{i,x,y}$ is null, algorithm $\mathcal{B}$ does the follows:
   (a) Pick a random $b_{i,x,y} \in \mathbb{Z}_q$;
   (b) Define $s_{i,x,y} = s_\epsilon b_{i,x,y}^{-1}$ ($\mathcal{B}$ does not have to compute $s_{i,x,y}$);
   (c) Set $T_{i,x,y} = b_{i,x,y}T_{i,1,1}$;
4. For $2 \leq k \leq l$ and $2 \leq j \leq h_i$ and $T_{i,k,j}$ is null, algorithm $\mathcal{B}$ does the follows:
   (a) Pick a random $b_{i,k,j} \in \mathbb{Z}_q$;
   (b) Pick a random $s_{i,k,j} \in \mathbb{Z}_q$;
   (c) Set $T_{i,k,j} = b_{i,k,j}P$;

Algorithm $\mathcal{B}$ then puts the following in H_1^{list}: $((t_{i,1}, \ldots, t_{i,l}), (\mathrm{ID}_{i,1}, \ldots, \mathrm{ID}_{i,h_i}))$, $(T_{i,1,1}, \ldots, T_{i,l,h_i})$, $(b_{i,1,1}, \ldots, b_{i,l,h_i})$, $(s_{i,1,1}, \ldots, s_{i,l,h_i})$, $(c_{i,1}, \ldots, c_{i,h_i})$, and $(c'_{i,1}, \ldots, c'_{i,l})$. $\mathcal{B}$ returns $(T_{i,1,1}, \ldots, T_{i,l,h_i})$ to $\mathcal{A}$.

Note that $T_{i,k,j}$ is always chosen uniformly in $\mathbb{G}_1$ and is independent of $\mathcal{A}$'s view as required.

Using above methods, $\mathcal{B}$ also stores the corresponding tuples that are necessary to compute the node keys at lower-level setup queries. Recall for time t_i and ID-tuple$_i$ $(\mathrm{ID}_1, \ldots, \mathrm{ID}_{h_i})$, node keys are denoted as sk_{w,h_i}, where w are all the labels of the right sibling, if one exists, of each ancestor of the node labeled t_i in the complete binary tree. They can also be written as $\{sk_{(t_i|_{k-1}1),h_i}\}_{t_{i,k}=0}$. For each time node w, $\mathcal{B}$ compares $(w, \text{ID-tuple}_i)$ with existing tuples in H_1^{list}, the same way as comparing $(t_i, \text{ID-tuple}_i)$ pair. Node keys together with the decryption key sk_{t_i,h_i} are what $\mathcal{B}$ needs to return in a lower-level setup query. We omit the details of how $\mathcal{B}$ prepares for node keys in this proof, as it is essentially the same as preparing the decryption key shown above.

Lower-Level Setup Queries: At any time, $\mathcal{A}$ may make a lower-level setup query on any time period t_i and ID-tuple$_i$. $\mathcal{B}$ responds to this query as follows:

1. Run the above algorithm for responding to H_1-queries to obtain the appropriate tuple (Time-ID-pair$_i$, Point-tuple$_i$, Scalar-tuple$_i$, Secret-tuple$_i$, ID-Coin-tuple$_i$, Time-Coin-tuple$_i$) in H_1^{list}.
2. If $c_{i,h_i} = 1$ and $c'_{i,l} = 1$, then $\mathcal{B}$ aborts.
3. Define $S_{i,t_i,h_i} = \sum_{k=1}^{l} \sum_{j=1}^{h_i} s_{i,k,j}T_{i,k,j}$. $\mathcal{B}$ also gives $\mathcal{A}$ the corresponding $\{Q_{i,k,j} = s_{i,k,j}P : 1 \leq k \leq l, 1 \leq j \leq h_i\}$.
   Let $sk_{i,t_i,h_i} = (S_{i,t_i,h_i}, \mathcal{Q}_{i,t_i,h_i})$, where $\mathcal{Q}_{i,t_i,h_i} = \{Q_{i,k,j} : 1 \leq k \leq l, 1 \leq j \leq h_i\}$. sk_{i,t_i,h_i} is what ID-tuple$_i$ needs to decrypt messages at t_i.
4. Using similar method for computing sk_{i,t_i,h_i}, $\mathcal{B}$ also computes the node keys $\{sk_{i,(t_i|_{k-1}1),h_i}\}_{t_i|_k=0}$

corresponding to time t_i and ID-tuple$_i$. It can be verified that those node keys, if $\mathcal{B}$ does not aborts, have the correct distribution. Using these node keys $\mathcal{B}$ can derive keys SK that are associated with any time $t > t_i$ or descendants of ID-tuple$_i$ with the correct distribution.

Readers can verify that each sk key is always computable by $\mathcal{B}$, if $\mathcal{B}$ does not abort. Note that $\mathcal{B}$ does not know s_ϵ nor $s_\epsilon P_1$, but does know $Q = s_\epsilon P$. The definition of $T_{i,k,j}$ when $c_{i,j} = 0$ and $c_{i,(j-1)} = 1$ is designed to cause certain cancellations to occur in the computation of $S_{i,k,j}$, so that this point can be computed by $\mathcal{B}$ even though $S_{i,(k-1),j}$ or $S_{i,k,(j-1)}$ may not. Similarly, cancellations occur when $c'_{i,k} = 0$ and $c'_{i,(k-1)} = 1$. The definition of $T_{i,k,j}$ when $c_{i,j} = 0$ and $c'_{i,k} = 0$ ($j > 1$ and $k > 1$) handles the case when both of the above described cancellations happen.

Challenge: Once adversary $\mathcal{A}$ decides that Phase 1 is over, it outputs an ID-tuple $(\mathrm{ID}_1, \ldots, \mathrm{ID}_h)$ and time period t on which it wishes to be challenged. Adversary $\mathcal{B}$ responds as follows:

1. Run the H_1-query to obtain the Point-tuple $(T_{1,1}, \ldots, T_{l,h})$, Secret-tuple $(s_{1,1}, \ldots, s_{l,h})$, Scalar-tuple $(b_{1,1}, \ldots, b_{l,h})$, ID-Coin-tuple $(c_1, \ldots, c_h)$, and Time-Coin-tuple $(c'_1, \ldots, c'_l)$. If $c_j = 0$ for some $1 \le j \le h$ or $c'_k = 0$ for some $1 \le k \le l$, then $\mathcal{B}$ reports failure and terminates. The attack on BasicPub failed.
2. Otherwise, we know $T_{1,1} = b_{1,1}P_1$ and $T_{k,j} = b_{k,j}P$ for all $1 \le k \le l$ and $1 \le j \le h$ except for $k = 1$ and $j = 1$. Let $C = (U, V)$ be the challenge ciphertext given to algorithm $\mathcal{B}$. $\mathcal{B}$ sets the fs-HIBE ciphertext C' to be

$$\begin{gathered}(b_{1,1}^{-1}U, b_{1,1}^{-1}b_{2,1}U, \ldots, b_{1,1}^{-1}b_{l,1}U,\\ b_{1,1}^{-1}b_{1,2}U, \ldots, b_{1,1}^{-1}b_{l,2}U,\\ \ldots\\ b_{1,1}^{-1}b_{1,h}U, \ldots, b_{1,1}^{-1}b_{l,h}U, V)\end{gathered}$$

$\mathcal{B}$ responds to $\mathcal{A}$ with the challenge C'. Note that C' is an fs-HIBE encryption of M under the target ID-tuple and t as required. To verify this, first observe that the decryption key corresponding to target ID-tuple and t is $S' = s_\epsilon T_{1,1} + \sum_{j=2}^{h}\sum_{k=1}^{l} s'_{k,j}T_{k,j} + \sum_{k=2}^{l} s'_{k,1}T_{k,1}$ along with the additional information $\{s'_{k,j}P : 1 \le k \le l \text{ and } 1 \le j \le h\}$ (except $s'_{1,1}$) for some set $\{s'_{k,j} : 1 \le k \le l \text{ and } 1 \le j \le h\}$ (except $s'_{1,1}$).
Second, observe that :

$$\begin{aligned}\frac{\hat{e}(b_{1,1}^{-1}U, S)}{\Pi_{k=1}^{l}\Pi_{j=2}^{h}\hat{e}(b_{1,1}^{-1}b_{k,j}U, s'_{k,j}P)\Pi_{k=2}^{l}\hat{e}(b_{1,1}^{-1}b_{k,1}U, s'_{k,1}P)} &= \hat{e}(b_{1,1}^{-1}U, s_\epsilon T_{1,1})\\ &= \hat{e}(b_{1,1}^{-1}U, s_\epsilon T_{1,1})\\ &= \hat{e}(b_{1,1}^{-1}U, s_\epsilon b_{1,1}P_1)\\ &= \hat{e}(U, s_\epsilon P_1)\end{aligned}$$

The correct decryption of C' is M.

Phase 2: Adversary $\mathcal{B}$ responds to more lower-level setup queries the way it did in Phase 1.

Guess: Eventually, $\mathcal{A}$ outputs a guess M'. $\mathcal{B}$ outputs M' as its guess for the decryption of C.

Claim: $\mathcal{A}$'s view is identical to its view in the real attack. Furthermore, $\Pr[M = M'] \ge \epsilon$. The probability is over the random bits used by $\mathcal{A}$, $\mathcal{B}$ and the challenger.

Proof of Claim: All responses to H_1-queries are as in the real attack since each response is uniformly and independently distributed in $\mathbb{G}_1$. All responses to lower-level setup queries are valid. Finally, the challenge ciphertext C' given to $\mathcal{A}$ is the fs-HIBE encryption of the random plaintext M under the ID-tuple and time period chosen by $\mathcal{A}$. Therefore, by the definition of the algorithm $\mathcal{A}$, it will output $M' = M$ with probability at least ϵ.
Probability: Suppose $\mathcal{A}$ makes a total of q_E lower-level setup queries. For each lower-level setup query, the adversary $\mathcal{B}$ needs to compute $\mathcal{O}(l)$ number of sk keys, including the sk key for the queried time and sk keys for computing future secret keys. The probability that $\mathcal{B}$ aborts when answering any one of the sk keys is $\leq \delta^2$. The probability that $\mathcal{B}$ does not abort in any one lower-level setup query is $\geq (1-\delta^2)^l$. Therefore, the probability that $\mathcal{B}$ does not abort in Phase 1 or 2 is at least $(1-\delta^2)^{l \times q_E}$.

The probability that it does not abort during the challenge step is δ^{h+l-1}. The overall probability that $\mathcal{B}$ does not abort is $\geq (1-\delta^2)^{lq_E}\delta^{h+l}$. The latter value is maximized at $\delta_{opt} = \sqrt{(h+l)/(2lq_E+h+l)}$. Using δ_{opt}, the probability that $\mathcal{B}$ does not abort is at least
$((h+l)/e(2lq_E+h+l))^{(h+l)/2}$. $\square$

If h and l are $O(1)$ and q_E is polynomial in the security parameter, then $((h+l)/e(2lq_E + h+l))^{(h+l)/2}$ is non-negligible in the security parameter, and we have a polynomial reduction from BasicPub to fs-HIBE for adaptive chosen-ciphertext-target adversaries.

D.4 Security Proof for fs-HIBE scheme

To complete the proof for Theorem 3.1 fs-HIBE scheme, we need the following result, which is Lemma 4.3 of [10], says that solving BDH reduces to breaking BasicPub.

Lemma D.3. *Suppose that $\mathcal{A}$ is an OWE adversary with advantage ϵ against BasicPub that makes a total of q_{H_2} queries to the hash function H_2, and suppose that H_2 is a random oracle. Then there is an algorithm $\mathcal{B}$ that solves the BDH problem for $\mathcal{IG}$ with advantage at least $(\epsilon - \frac{1}{2^n})/q_{H_2}$ and running time $O(time(\mathcal{A}))$.*

Theorem 3.1 for fs-HIBE scheme follows by combining Lemma D.1 and Lemma D.3. Theorem 3.2 for fs-HIBE scheme follows by combining Lemma D.2 and Lemma D.3.

E Construction of fs-BE scheme

We divide our construction of the fs-BE scheme into two parts. First, we describe how the Subset Difference method is extended to the public-key setting in the BE scheme [15]. At high level, this is done by constructing a tree $\mathcal{T}'_{SD}$ in such a way that its leaves correspond to the subsets of the cover family (described below), and is used for computing secret keys of users. Then, we apply the fs-HIBE scheme to the tree $\mathcal{T}'_{SD}$ to obtain an fs-BE scheme in Section E.2. Detailed description of each algorithm in fs-BE scheme is also given.
Notations in this section: In what follows, S denotes a subset, not a secret key. $MSK_{t,1}$ is the secret key of the center at time t, and USK is the secret key of a user. $SK_{t,\text{ID-tuple}}$ is the secret key in fs-HIBE that is associated with a time period t and an ID-tuple.

E.1 The public-key Broadcast Encryption scheme

Here, we describe the necessary knowledge of the public-key broadcast encryption [15] needed in order to understand the fs-BE scheme. We refer readers to broadcast encryption papers [15, 31] for a complete description of the subset-cover framework and the public-key broadcast encryption scheme. The construction of the public-key broadcast encryption [15] scheme is based on the *Subset-Cover Framework* [31], in particular, on the *Subset Difference* (SD) method. In the SD method, users are associated with the leaves of a complete binary tree $\mathcal{T}_{SD}$ of height $n = \log_2 E$, where E is the total number of users in the system. The generic subset S_{ij} is defined in terms of two nodes $u_i, u_j \in \mathcal{T}_{SD}$, and consists of all the leaves of the subtree rooted at u_i except those in the subtree rooted at u_j, where u_i is an ancestor of u_j. Node u_i is called the *primary root*, and u_j is called the *secondary root* of S_{ij}.

In the public-key broadcast encryption [15] scheme, a labeled tree $\mathcal{T}'_{SD}$ is defined, whose leaves correspond to all the subsets S_{ij} in the family $\mathcal{S}$. Nodes in the tree $\mathcal{T}'_{SD}$ are called *vertices*, in order to distinguish them from nodes in the tree $\mathcal{T}_{SD}$. The tree $\mathcal{T}'_{SD}$ is constructed from the tree $\mathcal{T}_{SD}$ as follows:

1. For each internal node u in $\mathcal{T}_{SD}$, a vertex w is added as the child to the root of $\mathcal{T}'_{SD}$. Therefore, at $Level_1$ of the tree $\mathcal{T}'_{SD}$ there are exactly $E-1$ vertices. These vertices correspond to all the possible *primary roots* u_i of a generic subset S_{ij}.
2. To each vertex w at $Level_1$ of $\mathcal{T}'_{SD}$ (corresponding, by Step 1, to an internal node u in $\mathcal{T}_{SD}$), attach two children w^ℓ and w^r corresponding respectively to the left and right children of node u in $\mathcal{T}_{SD}$.
3. For each vertex w at $Level_2$ (and recursively at lower levels):
   (a) if w is labeled with the symbol $\perp$, then w is a leaf in $\mathcal{T}'_{SD}$;
   (b) if w corresponds to a leaf u in $\mathcal{T}_{SD}$, then w is given a single child labeled $\perp$;
   (c) otherwise, w must correspond to an internal node u in $\mathcal{T}_{SD}$. Then, w is given three children labeled w^ℓ, w^r, and $\perp$, where w^ℓ and w^r correspond respectively to the left and right children of node u in $\mathcal{T}_{SD}$ (as in Step 2).

Each leaf vertex v that is labeled with the symbol $\perp$ in $\mathcal{T}'_{SD}$ corresponds to a subset of the cover family $\mathcal{S}$ as follows. Consider the path p from the root of $\mathcal{T}'_{SD}$ down to such a leaf v. Let w_i be the second vertex in the path p (i.e., w_i is a vertex at $Level_1$), and w_j be the second-to-last vertex in p (i.e. w_j is the parent of v). Then, associated with v is the subset S_{ij}, whose primary root is the node u_i corresponding to vertex w_i, and whose secondary root is the node u_j corresponding to vertex w_j.

E.2 Construction of a Forward-Secure Broadcast Encryption scheme using fs-HIBE

The central idea of constructing an fs-BE scheme is to use fs-HIBE over the tree $\mathcal{T}'_{SD}$ that is defined in Appendix E.1. Each vertex in $\mathcal{T}'_{SD}$ is given an ID, which is unique among its siblings. Then, a vertex w in $\mathcal{T}'_{SD}$ can be associated with an ID-tuple (denoted ID-tuple$_w$) by simply considering the sequence of IDs corresponding to each of w's ancestors (including vertex w itself). For a vertex w in $\mathcal{T}'_{SD}$ and a time period t, there is a secret key $SK_{t,\text{ID-tuple}_w}$, which can be computed using the Lower-Level Setup algorithm of fs-HIBE. Therefore, at any time period t, each vertex w in $\mathcal{T}'_{SD}$ is associated with an ID-tuple$_w$ and a secret key $SK_{t,\text{ID-tuple}_w}$.

Construction:

- KeyGen(k, r_{max}, E, N): Run algorithm Root Setup(k, N) of fs-HIBE. Set $PK = params$ and $MSK_0 = SK_{0,1}$.
- Reg(MSK_t, u, t): Each user u is represented as a leaf in tree $\mathcal{T}_{SD}$. Consider the path from the root of $\mathcal{T}_{SD}$ to u; let $u_0, u_1, \ldots, u_n \equiv u$ be all the ancestors of u, and denote by v_h the sibling of $u_h, h = 1, \ldots, n$. For $1 \le i < j \le n$ consider the subset difference S_{u_i,v_j} and the associated

leaf l_{u_i,v_j} in $\mathcal{T}'_{SD}$; let w_{u_i,v_j} be the parent of l_{u_i,v_j}. Then, starting from $MSK_t = SK_{t,1}$, recursively apply the LOWER-LEVEL SETUP algorithm of fs-HIBE to derive the secret key $SK_{t,\text{ID-tuple}_{i,j}}$ corresponding to ID-tuple$_{i,j}$ associated to vertex w_{u_i,v_j}. Set the user's secret key $USK_{t,u} = \{\langle \text{ID-tuple}_{i,j}, SK_{t,\text{ID-tuple}_{i,j}} \rangle \mid 1 \leq i < j \leq n\}$.

- UPD($PK, u, t, USK_{t,u}$): For each $\langle \text{ID-tuple}_{i,j}, SK_{t,\text{ID-tuple}_{i,j}} \rangle \in USK_{t,u}, 1 \leq i < j \leq n$, run algorithm UPDATE($t$, ID-tuple$_{i,j}$, $SK_{t,\text{ID-tuple}_{i,j}}$) of fs-HIBE to compute $SK_{t+1,\text{ID-tuple}_{i,j}}$ and erase $SK_{t,\text{ID-tuple}_{i,j}}$. Return $USK_{t+1,u} = \{\langle \text{ID-tuple}_{i,j}, SK_{t+1,\text{ID-tuple}_{i,j}} \rangle \mid 1 \leq i < j \leq n\}$. For the center, call the UPDATE(t, MSK_t) of fs-HIBE to compute the master secret key MSK_{t+1} for time period $t+1$.
- ENC($PK, M, t, \mathcal{R}$): First run the *Cover* algorithm (cf. Naor *et al.* [31]) to partition the set $\mathcal{E} \setminus \mathcal{R}$ into the subsets $\{S_{i_1,j_1}, \ldots, S_{i_z,j_z}\}$. (The *Cover* algorithm guarantees that $z < 2|\mathcal{R}|$.) Recall that each subset $S_{i_h,j_h} (h = 1, \ldots, z)$, corresponds to a leaf l_{u_h,v_h} in the tree $\mathcal{T}'_{SD}$: let ID-tuple$_h$ be the ID-tuple associated to the leaf l_{u_h,v_h}. For each ID-tuple$_h$, $(h = 1, \ldots, z)$, run algorithm ENCRYPT($params, t$, ID-tuple$_h$, M) of fs-HIBE to compute ciphertext C_h. Return $C = \langle \text{ID-tuple}_1, \ldots, \text{ID-tuple}_z, C_1, \ldots, C_z \rangle$.[6]
- DEC($PK, u, t, USK_{t,u}, C$): By definition of the REG algorithm, if the ciphertext $C = \langle \text{ID-tuple}_1, \ldots, \text{ID-tuple}_z, C_1, \ldots, C_z \rangle$ was constructed for a subset $\mathcal{R}$ of revoked users which does not include u, then among the ID-tuples in C, there exists one (say, ID-tuple$_h$) which is a descendent of one of the ID-tuples (say, ID-tuple$_{i,j}$) for which user u received the corresponding secret key when he joined the system. Let $SK_{t,\text{ID-tuple}_{i,j}}$ be the value of such a secret key at time period t. Starting from $SK_{t,\text{ID-tuple}_{i,j}}$, recursively apply LOWER-LEVEL SETUP of fs-HIBE to derive the secret key $SK_{t,\text{ID-tuple}_h}$ corresponding to ID-tuple$_h$. Then, run algorithm DECRYPT($params, t$, ID-tuple$_h$, $SK_{t,\text{ID-tuple}_h}, C_h$) of fs-HIBE to obtain the corresponding message M.

[6] In fact, this only achieves gCCA security [2]. In order to achieve CCA2 security using "parallel encryption", it is necessary to resort to the techniques demonstrated in [17]. We omit the details.