Broadcast Encryption with Fine-grained Delegation and its Application to IoT

Yuncong Hu

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2020-52
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-52.html

May 20, 2020

Broadcast Encryption with Fine-grained Delegation and its Application to IoT

by Yuncong Hu

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor Raluca Ada Popa
Research Advisor

Professor Alessandro Chiesa
Second Reader

Broadcast Encryption with Fine-grained Delegation and its Application to IoT

Yuncong Hu
University of California, Berkeley

Abstract

Broadcast encryption schemes allow senders to distribute data to selected receivers securely. Broadcast encryption schemes have been widely used in designing revocation protocols in publish-subscribe systems. However, existing broadcast encryption schemes do not support delegation, which is essential for secure communication in IoT systems. We extend tree-based broadcast encryption schemes to support delegation, which allows subscribers to delegate their keys to other subscribers in a fine-grained, distributed way. We design a revocation protocol for IoT systems based on the delegable broadcast encryption scheme. By incorporating our revocation protocols and other designs, we propose JEDI [49] (**J**oining **E**ncryption and **D**elegation for **I**oT), a many-to-many end-to-end encryption protocol for IoT. JEDI encrypts and signs messages end-to-end while conforming to the decoupled communication model typical of IoT systems. This report only focuses on the part of the JEDI design that includes the delegable broadcast encryption and the revocation protocol. Please refer to the JEDI paper [49] for more details.

Contents

1 Introduction

Broadcast encryption scheme [36, 59, 32, 20, 21, 50, 22, 23] allows a publisher to encrypt and distribute messages to a subset of subscribers who are listening on an insecure broadcast channel, and only subscribers within this subset can decrypt the broadcast. Even if subscribers outside of the subset collude, they cannot obtain any information about the contents of the broadcast.

Broadcast encryption scheme has seen widespread usage in designing revocation protocols. For example, the publisher maintains a revocation list that stores information about all the revoked subscribers. During the broadcast, the publisher encrypts the message so that only subscribers outside of this revocation list can decrypt and read. Such kinds of revocation protocols can be used for some specific applications, including access control in digital content management systems [56, 76, 45].

Recently, as the Internet of Things (IoT) has emerged over the past decade, smart devices have become increasingly common. This trend is only expected to continue, with tens of billions of new IoT devices deployed over the next few years [27]. The IoT vision requires these devices to communicate to discover and use the resources and data provided by one another. However, these devices collect privacy-sensitive information about users. A natural step to secure privacy-sensitive data is to use *end-to-end encryption* to protect it during transit.

Specialized end-to-end encryption protocols [60, 66, 49] have been proposed to enable efficient end-to-end encryption in large-scale industrial IoT scenarios. However, existing broadcast encryption schemes [36, 59, 32, 20, 21, 50, 22, 23] appear not to be a good fit for IoT systems. We investigate existing IoT systems to understand the requirements of the revocation protocols. We identify three central requirements, which we treat in turn below:

▷ **Stateless subscribers.** Some revocation protocols might require all subscribers periodically refreshing decryption keys, and revoked subscribers are not able to update their decryption keys. However, IoT-scale systems could consist of thousands of principals, making it expensive for each subscriber to keep decryption keys up to date. Moreover, some decryption keys are hard-coded in smart devices. It is infeasible to reinstall all the devices and refresh hard-coded decryption keys.

▷ **Immediate revocation.** A simple solution for revocation is to rely on expiration. In this solution, all keys are time-limited, and subscribers with expired keys cannot decrypt new messages. This solution is sufficient for applications where subscribers do not need to obtain a new decryption key after expiry. Otherwise, key updates might be expensive, as discussed above. Furthermore, because revocation only takes effect when the decryption key expires, revoked subscribers can still learn secret before the expiry. It is desirable to have immediate revocation.

▷ **Decentralized delegation.** Access control in IoT needs to be fine-grained. For example, if Bob has an app that needs access to temperature readings from a single sensor, that app should receive the decryption key for only that one sensor, even if Bob has keys for all sensors in the entire room. In an IoT-scale system, it is not scalable for a central authority to individually give fine-grained decryption keys to each person's devices. Moreover, such an approach would pose increased security and privacy risks. Instead, Bob, who has access to readings for the entire room, should be able to delegate temperature-readings access to the app. Generally, a principal with access to a set of resources can give another principal access to a subset of those resources, as shown in Fig. 1.

Vanadium [68] and bw2 [6] introduced *decentralized delegation* (SPKI/SDSI [28] and Macaroons [14]) in the smart buildings space. Since then, decentralized delegation has become the state-of-the-art for access control in smart buildings, especially those geared toward large-scale commercial buildings or organizations [37, 44]. In these systems, a principal can access a resource if there exists a *chain* of delegations, from the owner of the resource to that principal, granting access. At each link in the chain, the extent of

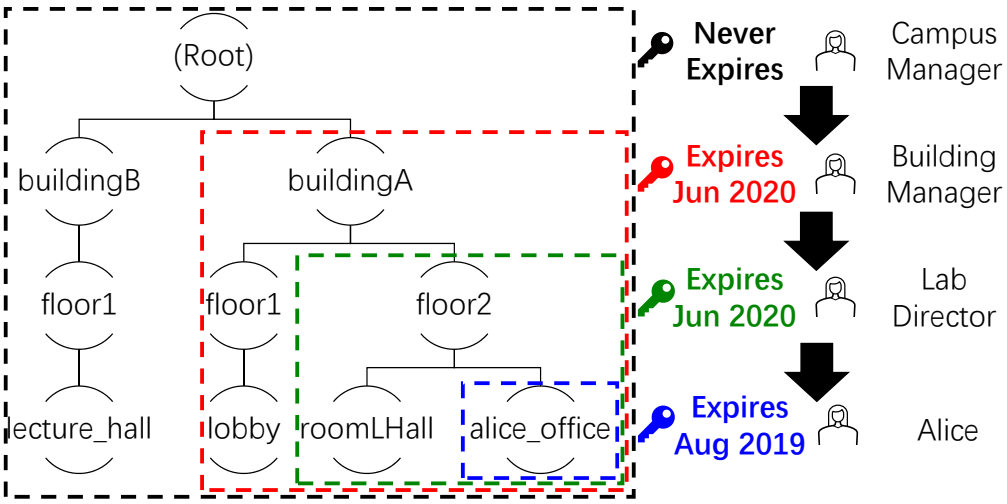

Figure 1: JEDI keys can be qualified and delegated, supporting decentralized, cryptographically-enforced access control via key delegation. Each person has a decryption key for the indicated resource subtree that is valid until the indicated expiry time. Black arrows denote delegation.

access may be qualified by *caveats*, which add restrictions to which resources can be accessed and when. While these systems provide delegation of permissions, they do not provide protocols for encrypting and decrypting messages end-to-end.

Although fine-grained decentralized delegation is essential in IoT systems, it makes revocation much harder. As shown in Fig. 1, the lab director gives Alice access to her office. Later, when the lab director gets fired, both decryption keys of the lab director and Alice must be revoked. Otherwise, the lab director may use Alice's decryption key to access her office. Sometimes this "delegation tree" is so big that revoking the decryption key of each one by one becomes infeasible. We want a revocation scheme such that after a decryption key gets revoked, all of its derivatives get revoked automatically.

A naive solution is to share decryption keys. For example, the lab director shares the decryption key of the designate office with Alice. However, this solution is still not fine-grained. When Alice gets fired, the lab director will also lose access to the office.

In summary, revocation in IoT systems demands much complex semantics, while we are unaware of any broadcast encryption schemes or revocation protocols that fulfill all of those requirements. Our contributions are the following:

- We first present a new broadcast encryption schemes to support delegation. Our new broadcast encryption schemes are based on tree-based broadcast encryption schemes [32, 59]. However, naive adaption results in a huge decryption key size. We show how to compress the decryption key by leveraging hierarchical structures and non-trivially utilizing identity-based encryption with wildcard key derivation (WKD-IBE) [1].
- We show how to design revocation protocols for IoT systems based on our new broadcast encryption scheme. We identify that both resources in IoT and our new broadcast encryption scheme can be expressed in hierarchies. We again leverage WKD-IBE to combine resource hierarchy with broadcast encryption hierarchy.
- We evaluate our broadcast encryption schemes and revocation protocols on both normal devices and low-power devices. The result shows that it is feasible to run our scheme in IoT systems.

By incorporating our revocation protocols and other designs, we propose JEDI [49], a many-to-many end-to-end encryption and key delegation scheme for IoT systems. This report focuses only on broadcast encryption and revocation protocol designs in JEDI [49]. Please refer to JEDI [49] for more details.

2 Preliminaries

2.1 Identity-based encryption with wildcard key derivation

We begin by formally defining identity-based encryption with wildcard key derivation [1], a class of identity-based encryption that allows more general key delegation patterns. This notion was proposed to enhance the concept of hierarchical identity-based encryption (HIBE) [39].

In our work, we observe the ability of WKD-IBE to support delegation with multiple hierarchies while HIBE cannot. This enables us to design efficient broadcast encryption and revocation protocol. Although other encryption schemes like attribute Attribute-Based Encryption (ABE) [41, 13] can also support multiple hierarchies like WKD-IBE, they are too expensive to be used in the context of resource constraints of IoT devices.

In WKD-IBE, messages are encrypted with *patterns*, and keys also correspond to patterns. A pattern is a list of values: $P = (\mathbb{Z}_p^* \cup \{\perp\})^\ell$. The notation $P(i)$ denotes the ith component of P, 1-indexed. A pattern P_1 *matches* a pattern P_2 if, for all $i \in [1, \ell]$, either $P_1(i) = \perp$ or $P_1(i) = P_2(i)$. In other words, if P_1 specifies a value for an index i, P_2 must match it at i. Note that the "matches" operation is not commutative; "P_1 matches P_2" does not imply "P_2 matches P_1".

We refer to a component of a pattern containing an element of $\mathbb{Z}_p^*$ as *fixed*, and to a component that contains $\perp$ as *free*. To aid our presentation, we define the following sets:

Definition 1. *For a pattern S, we define:*

$$\mathit{fixed}(S) = \{(i, S(i)) \mid S(i) \neq \perp\}$$
$$\mathit{free}(S) = \{i \mid S(i) = \perp\}$$

A key for pattern P_1 can decrypt a message encrypted with pattern P_2 if $P_1 = P_2$. Furthermore, a key for pattern P_1 can be used to derive a key for pattern P_2, as long as P_1 matches P_2. In summary, the following is the syntax for WKD-IBE.

- $\mathsf{WIBE.Setup}(1^\kappa, 1^\ell) \to \mathsf{ibpp}, \mathsf{ibmk}$. On input a security parameter κ, and a maximum slot number $\ell \in \mathbb{N}$, WIBE.Setup outputs the public parameters ibpp and a master key ibmk.
- $\mathsf{WIBE.KeyGen}(\mathsf{ibpp}, \mathsf{ibmk}, \mathsf{P}) \to \mathsf{SK}_\mathsf{P}$. On input ibpp, the master key ibmk and the pattern P, WIBE.KeyGen outputs the secret key SK_P.
- $\mathsf{WIBE.KeyDer}(\mathsf{ibpp}, \mathsf{SK}_{\mathsf{P}_1}, \mathsf{P}_2) \to \mathsf{SK}_{\mathsf{P}_2}$. On input ibpp, the secret key $\mathsf{SK}_{\mathsf{P}_1}$, and the pattern P_2, WIBE.KeyDer outputs the secret key $\mathsf{SK}_{\mathsf{P}_2}$. Note that the pattern P_1 must match the pattern P_2. Otherwise WIBE.KeyDer will abort.
- $\mathsf{WIBE.Enc}(\mathsf{ibpp}, \mathsf{P}, m) \to \mathsf{Ciphertext}_{\mathsf{P},m}$. On input ibpp, the pattern P, and the message m, WIBE.Enc outputs the ciphertext $\mathsf{Ciphertext}_{\mathsf{P},m}$ of the message m under the pattern P.
- $\mathsf{WIBE.Dec}(\mathsf{SK}_\mathsf{P}, \mathsf{Ciphertext}_{\mathsf{P},m}) \to m$. On input the secret key SK_P of the pattern P, and the ciphertext $\mathsf{Ciphertext}_{\mathsf{P},m}$, WIBE.Dec outputs the message m.

The security definition of WKD-IBE is shown below.

Definition 2 (IND-sWKID-CPA [17, 1])**.** *A WKD-IBE is said to be Selective-ID CPA-secure (IND-sWKID-CPA) if there exists no probabilisitc polynomial-time adversary $\mathcal{A}$ who can win the following security game against a challenger $\mathcal{C}$ with more than negligible advantage:*
Initialization. *$\mathcal{A}$ selects a set of patterns $\mathcal{P}$ to attack.*
Setup. *$\mathcal{C}$ gives $\mathcal{A}$ the public parameters of the WKD-IBE instance.*
Phase 1. *$\mathcal{A}$ can ask $\mathcal{C}$ to run* WIBE.KeyGen *on any pattern P', as long as $\mathsf{P}' \notin \mathcal{P}$ and there is no pattern*

$\mathsf{P} \in \mathcal{P}$ *and* P' *matches* P. $\mathcal{A}$ *can also ask* $\mathcal{C}$ *to run* WIBE.KeyDer *to derive decryption keys.*
Challenge. *When* $\mathcal{A}$ *chooses to end Phase 1, it sends* $\mathcal{C}$ *two messages,* m_0 *and* m_1*, of the same length. Then* $\mathcal{C}$ *chooses a random bit* $b \in \{0,1\}$*, encrypts* m_b *under the challenge pattern set* $\mathcal{P}$*, and gives* $\mathcal{A}$ *all the ciphertexts.*
Phase 2. $\mathcal{A}$ *can make additional queries as in Phase 1.*
Guess. $\mathcal{A}$ *outputs* $b' \in \{0,1\}$*, and wins the game if* $b = b'$*. The advantage of an adversary* $\mathcal{A}$ *is* $\left|\Pr[\mathcal{A} \textit{ wins}] - \frac{1}{2}\right|$.

Definition 3 (History-Independence [49])**.** *A WKD-IBE construction is said to be history-independent if, for every pattern* S*, and for any two well-formed keys* k_1 *(corresponding to pattern* P_1*) and* k_2 *(corresponding to pattern* P_2*) in the same WKD-IBE system such that* P_1 *matches* S *and* P_2 *matches* S*, it holds that*

$$\{\mathbf{KeyDer}(k_1, S)\} = \{\mathbf{KeyDer}(k_2, S)\}$$

where the distributions are over the randomness sampled internally by $\mathbf{KeyDer}$*. If* k_1 *(respectively,* k_2*) is the master key, the pattern* P_1 *(respectively,* P_2*) is one where all slots are free.*

We use the WKD-IBE construction in §3.2 of [1], based on BBG HIBE [18]. Like the BBG construction, it has constant-size ciphertexts, but requires the maximum pattern length ℓ to be known at Setup time. In this WKD-IBE construction, patterns containing $\perp$ can only be used in $\mathbf{KeyDer}$, not in $\mathbf{Encrypt}$; we extend it to support encryption with patterns containing $\perp$. We include the WKD-IBE construction with our optimizations and proofs of security in [49].

2.2 Tree-based broadcast encryption

Broadcast encryption scheme was first explored by Fiat and Naor [36]. Since its introduction, many different broadcast encryption schemes have been studied [59, 32, 20, 21, 50, 22, 23]. In particular, we focus on the tree-based broadcast encryption scheme [59, 32] because we identify its ability to support delegation.

Tree-based broadcast encryption leverages the idea of *Subset-Cover* method in which the key generator generates many unrelated decryption keys, and each subscriber holds a different subset of those keys. During the broadcast, the publisher encrypts the message with a small subset of encryption keys so that only privileged subscribers hold corresponding decryption keys. Because the message is encrypted for each one of those chosen encryption keys, the size of ciphertext is linear to the number of chosen encryption keys. Also, subscribers may have limited storage, for example, resource-constrained devices, so that they cannot hold too many decryption keys.

To achieve *Subset-Cover* efficiently, tree-based broadcast encryption organizes subscribers in a tree structure, and offers two subset-cover frameworks:

- **Complete Subtree (CS) method.** The ciphertext consists of $O(r \log(\frac{n}{r}))$ encryptions, and each subscirber holds $O(\log(n))$ decryption keys.
- **Subset Difference (SD) method.** The ciphertext consists of $2r - 1$ encryptions, and each subscriber holds $O(\log^2(n))$ decryption keys.

in which n is the total number of subscribers, and r is the number of revoked subscribers.

Both of these works leverage identity-based encryption [19] or hierarchical identity-based encryption [39] to obtain fixed *constant size* public keys. In our work, we modify both CS and SD methods to support delegation.

2.2.1 Complete Subtree method

In the CS method, each subscriber is associated with a leaf of a complete binary tree $\mathcal{T}$, which is called *broadcast tree*. For simplicity, we assume that there are n subscribers in the system, where n is a power of 2 (for example, $n = 2^\ell$, for some integer ℓ). The subset-cover family $\mathcal{S}(\mathcal{T})$ is set to be the collection of all the full subtrees of $\mathcal{T}$. Precisely, for each $\mathcal{S}_i \in \mathcal{S}(\mathcal{T})$, we denote $\mathcal{S}_i$ the set of all leaves in the full subtree rooted at v_i.

Each subset $\mathcal{S}_i \in \mathcal{S}(\mathcal{T})$ is assigned a key pair $(\mathsf{PK}_i, \mathsf{SK}_i)$. Each subscriber holds the secret keys corresponding to all the subsets it belongs to. For example, a subsecriber associated with the leaf node v_j holds the secret key SK_i if the node v_j belongs to the set $\mathcal{S}_i$.

During encryption, the publisher first finds out the minimal number of subsets covering only all the leaves corresponding to unrevoked users. Then the publisher encrypts the message using the public key associated with those subsets. The number of encryptions and ciphertexts is just $O(r \log(\frac{n}{r}))$ where n is the total number of subscribers, and r is the number of revoked subscribers in the system. We notice that the storage requirement on each subscriber is just $O(\log(n))$ since each leaf is in only $O(\log(n))$ subtrees. As for the storage requirement of the publisher, we also notice that each node in the tree can be identified by the path to the root so that we can leverage identity-based encryption [19] that provides a fixed constant size public key.

2.2.2 Subset Difference method

One problem with the Complete Subtree method is that its ciphertext size depends on the total number of all subscribers in the system. To solve this problem, the SD method leverages a more complicated subset-cover family $\mathcal{S}(\mathcal{T})$: each leaf node belongs to more subsets so that there is more freedom in the choice of the covering subsets.

In the SD method, we define the set $\mathcal{S}_{ij} \in \mathcal{S}(\mathcal{T})$ in terms of two nodes $v_i, v_j \in \mathcal{T}$ where v_j is a descendant of v_i. For each $\mathcal{S}_{ij} \in \mathcal{S}(\mathcal{T})$, it consists of all the leaves v_k such that the node v_k is in the subtree rooted at v_i but not in the subtree rooted at v_j. We noticed that each leaf node belongs to $\sum_{\ell=1}^{\log(n)}(2^\ell - \ell)$ different subsets, which means each subscriber must store $O(n)$ private keys. To avoid that, the SD method [32] makes use of hierarchical identity-based encryption [39] and each subsriber only needs $O(\log^2(n))$ private keys.

As for encryption, only $O(r)$ sets are enough to cover leaves corresponding to all of the unrevoked subscribers. Similar to the CS method, publishers hold a fixed constant size public key by using HIBE.

3 Delegable broadcast encryption

The key difference between our broadcast encryption and existing works [59, 32] is that our broadcast encryption supports fine-grained decentralized delegation, as discussed in Section 1.

We observe that in the tree-based broadcast encryption, each subscriber is associated with a single leaf node. In order to support delegation, we instead associate each subscriber with a set of leaves. During the delegation, the subscriber can simply give private keys corresponding to a subset of leaves to others. When a subscriber gets revoked, all leaves owned by this revoked subscriber will be considered as revoked. Therefore, once a delegator is revoked, all the following delegatee will also be revoked automatically. On the other hand, even of all delegatees are revoked, as long as there exists at least one unrevoked leaf associated with the delegator, the delegator can decrypt the message.

This design seems to be a natural extension of existing tree-based broadcast encryption schemes. However, it is still not clear how to instantiate this idea efficiently. For example, in the CS method, each subscriber holds $O(\log(n))$ private keys, where n is the number of leaves in the broadcast tree. When we extend it to support delegation directly, each subscriber must hold $O(k + \log(n))$ private keys where k is the number of leaves corresponding to the subscriber. Moreover, the performance in the SD method will become even worse due to the complicated subset family, in which each subscriber may hold $O(k \log^2(n))$ private keys even if HIBE is used as in [32]. This is not desirable for many applications. For example, IoT devices may have limited memory to store private keys. To solve this, we associate each subscriber with a set of **consecutive** leaves and design an algorithm to compress private keys by leveraging the WKD-IBE scheme.

The rest of this section is organized as follows. In Section 3.1, we present the definition that we propose for delegable broadcast encryption schemes. In Section 3.2 and Section 3.3, we extend the Complete Subtree method and the Subset Difference method to support delegation and present how to compress the storage for secret keys by using WKD-IBE. We formalize our construction in Section 3.4 and prove the security in Section 3.5.

3.1 Definition

We begin by assigning each leaf in the broadcast tree a unique identifier to identify its position in the tree. For simplicity, we assume the identifier for a leaf is the number of leaves on the left. In our broadcast encryption, each decryption key is associated with a set of identifiers, which must be consecutive. The publisher now can specify a set of revoked identifiers and encrypt the message so that only decryption keys, which are associated with at least one unrevoked identifier, can decrypt the message.

The delegable broadcast encryption scheme is a tuple of algorithms

$$\mathsf{BE} = (\mathsf{BE.Setup}, \mathsf{BE.KeyGen}, \mathsf{BE.KeyDer}, \mathsf{BE.Enc}, \mathsf{BE.Dec})$$

with the following syntax.

- $\mathsf{BE.Setup}(1^\kappa, n) \to \mathsf{bpp}, \mathsf{bmk}$. On input the security parameter κ and the total number n of subscribers in the system, BE.Setup outputs the public parameter bpp and the master key bmk.
- $\mathsf{BE.KeyGen}(\mathsf{bpp}, \mathsf{bmk}, \mathsf{IDs}) \to \mathsf{Key}_{\mathsf{IDs}}$. On input the public parameter bpp, the master key bmk, and the identifier set IDs, BE.KeyGen outputs the decryption key $\mathsf{Key}_{\mathsf{IDs}}$ for the identifier set IDs. IDs must contain consecutive identifiers, otherwise BE.KeyGen aborts.
- $\mathsf{BE.KeyDer}(\mathsf{bpp}, \mathsf{Key}_{\mathsf{IDs}}, \mathsf{IDs}') \to \mathsf{Key}_{\mathsf{IDs}'}$. On input the public parameter bpp, the decryption key $\mathsf{Key}_{\mathsf{IDs}}$ for identifier set IDs, and another identifier set IDs', BE.KeyDer outputs another decryption key $\mathsf{Key}_{\mathsf{IDs}'}$ for the identifier set IDs'. Note that both IDs and IDs' contain consecutive identifiers, and IDs' must be the subset of IDs. Otherwise BE.KeyDer aborts.

- $\mathsf{BE.Enc}(\mathsf{bpp}, \mathsf{rIDs}, m) \rightarrow \mathsf{Ciphertext}_{\mathsf{rIDs},m}$; On input the public parameter bpp, the revoked identifier set rIDs, and the message m, BE.Enc outputs the ciphertext $\mathsf{Ciphertext}_{\mathsf{rIDs},m}$ of the message m.
- $\mathsf{BE.Dec}(\mathsf{bpp}, \mathsf{Key}_{\mathsf{IDs}}, \mathsf{rIDs}, \mathsf{Ciphertext}_{\mathsf{rIDs},m}) \rightarrow m$. On input the public parameter bpp, the decryption key $\mathsf{Key}_{\mathsf{IDs}}$, the revoked identifier set rIDs, and the ciphertext $\mathsf{Ciphertext}_{\mathsf{IDs},m}$, If there exist $\mathsf{ID} \in \mathsf{IDs}$ and $\mathsf{ID} \notin \mathsf{rIDs}$, BE.Dec outputs the message m. Otherwise, BE.Dec aborts.

We define the security of delegable broadcast encryption schemes in a way that is similar to the case of WKD-IBE scheme, but the adversary can choose identifiers to attack. We formalize the security definition below.

Definition 4. *A delegable broadcast encryption scheme is said to be Selective-ID CPA-secure if there exists no probabilisitc polynomial-time adversary $\mathcal{A}$ who can win the following security game against a challenger $\mathcal{C}$ with more than negligible advantage:*
Initialization. *$\mathcal{A}$ selects a challenge list $\mathcal{L}$ of identifiers to attack.*
Setup. *$\mathcal{C}$ gives $\mathcal{A}$ the public parameters of the* BE *instance.*
Phase 1. *$\mathcal{A}$ can make queries to $\mathcal{C}$. In the chosen-plaintext attack, $\mathcal{A}$ can ask $\mathcal{C}$ to run* BE.KeyGen *on any identifier set* IDs*, where identifiers in* IDs *are not in the challenge list $\mathcal{L}$. $\mathcal{A}$ can also ask $\mathcal{C}$ to run* BE.KeyDer *to derive decryption keys.*
Challenge. *When $\mathcal{A}$ chooses to end Phase 1, it sends $\mathcal{C}$ two messages, m_0 and m_1, of the same length. Then $\mathcal{C}$ chooses a random bit $b \in \{0, 1\}$, encrypts m_b with the challenge identifiers list $\mathcal{L}$, and gives $\mathcal{A}$ all of the ciphertexts.*
Phase 2. *$\mathcal{A}$ can make additional queries as in Phase 1.*
Guess. *$\mathcal{A}$ outputs $b' \in \{0, 1\}$, and wins the game if $b = b'$. The advantage of an adversary $\mathcal{A}$ is $\left|\Pr[\mathcal{A}\ \textit{wins}] - \frac{1}{2}\right|$.*

3.2 Delegable Complete Subtree method

In this section, we present how to extend the Complete Subtree method to support delegation.

Key Assignment. In the CS method, each user is assigned to exactly one leaf node and holds $O(\log(n))$ private keys associated with the ancestor nodes of that leaf node. In order to support delegation, the user may hold private keys corresponding to multiple leaves. A strawman solution is that the user holds the union set of secret keys corresponding to all the leaves. However, we observe that the straightforward extension results in linear storage costs. For example, in Fig. 2, if Alice is assigned to four leaves ([0:3]), there will be nine secret keys (in green and blue) corresponding to those leaves. In particular, if the user is assigned to k consecutive leaves, it may hold $O(k + \log(n))$ secret keys, where n is the total number of leaves in the tree. Note that by leveraging Identity-Based Encryption (IBE) [19], the public key for the whole system will be of constant size. However, IBE cannot save the storage for secret keys.

We observe that the user's secret keys are arranged in a tree structure. In particular, a set of consecutive leaves can be partitioned into $O(\log(k))$ complete subtrees. If we can find a way to derive secret keys in the tree from the secret key in the root node, each user only needs a logarithmic number of secret keys. A central question is: how to derive the secret key?

We answer this question by using the WKD-IBE. Note that we can also use Hierarchical Identity-Based Encryption (HIBE) [39] for the CS method. However, HIBE does not work for the SD method. To begin with, we introduce a node label for each node in the broadcast tree. As shown in Fig. 2, we label each edge with 0 or 1 depending on which child the edge connects to. The node label is the bitstring by reading all the labels in the path from root down to the node. And the root label is the empty string.

Now we use WKD-IBE to generate key pairs for each node in the broadcast tree. For a node v, we denote $\mathsf{Dep}(v)$ the depth of the node v in the broadcast tree. We produce a pattern P_v for the node v as follows:

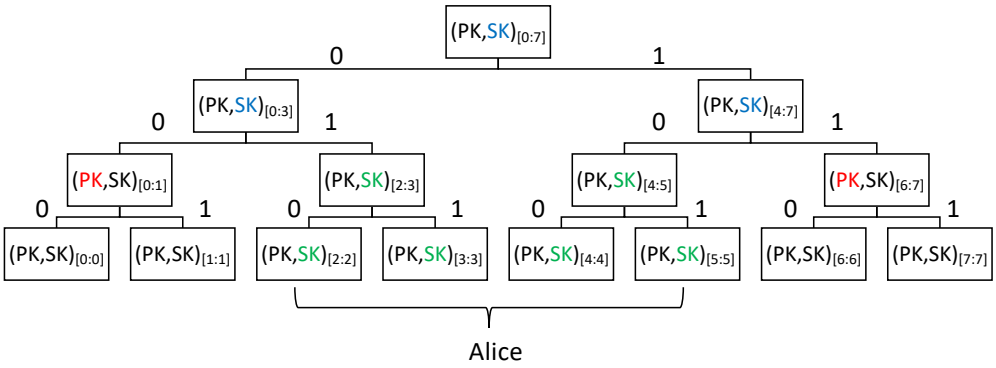

Figure 2: Key management of the delegable CS method. Each node in the broadcast tree is associated with a key pair. Alice is assigned to a range of leaves [2:5] and holds the secret keys in green and blue. If Alice is revoked, the publisher should use public keys in red to encrypt the message.

1. for $0 \leq i < \mathsf{Dep}(v)$, $P_v(i) = \mathsf{label}_v(i)$;
2. for $i = \mathsf{Dep}(v)$, $P_v(i) = 2$;
3. for $i > \mathsf{Dep}(v)$, $P_v(i) = \perp$.

And the secret key for the node v is:

$$\mathsf{SK}_v \leftarrow \mathsf{WIBE.KeyGen}(\mathsf{ibpp}, \mathsf{ibmk}, P_v)$$

As we can see, some nodes may share the prefix in the patterns if they are in the same subtree. Therefore, we can find a secret key with the pattern, which can be used to derive all the secret keys in the subtree. Given a range of leaves corresponding to the user, we can figure out $O(\log(k))$ complete subtrees covering all those leaves. For each root node v of those complete subtrees, we produce a "proto-pattern" PP_v for that node v as follows:

1. for $0 \leq i < \mathsf{Dep}(v)$, $PP_v(i) = \mathsf{label}_v(i)$;
2. for $i \geq \mathsf{Dep}(v)$, $PP_v(i) = \perp$.

And we produce a "proto-key" for the node v as follows:

$$\mathsf{PSK}_v \leftarrow \mathsf{WIBE.KeyGen}(\mathsf{ibpp}, \mathsf{ibmk}, PP_v)$$

Note that the difference between PP_v and P_v above is that $PP_v(\mathsf{Dep}(v)) = \perp$ while $P_v(\mathsf{Dep}(v)) = 2$. This difference allows the user to use PSK_v to derive secret keys associated with any nodes inside the subtree rooted at v. For example, if the node u is one of the nodes inside the subtree rooted at v, we can derive the secret key SK_u as follows.

$$\mathsf{SK}_u \leftarrow \mathsf{WIBE.KeyDer}(\mathsf{ibpp}, \mathsf{PSK}_v, P_u)$$

This is feasible because the proto-pattern PP_v is the prefix of the pattern P_u (and thus matches P_u). Therefore, the user only needs to store $O(\log(k))$ proto-keys for roots of those complete subtrees, and all the secret keys associated with nodes inside the complete subtrees can be derived. Note that the user still needs to hold secret keys associated with nodes outside those complete subtrees, but we observe that there are $O(\log(n))$ such nodes.

For example, in Fig. 2, instead of storing all the secret keys in green, Alice only needs to store the proto-keys $\mathsf{PSK}_{[2:3]}$ and $\mathsf{PSK}_{[4:5]}$. However, Alice still needs to store the secret keys in blue ($\mathsf{SK}_{[0:7]}, \mathsf{SK}_{[0:3]}, \mathsf{SK}_{[4:7]}$).

Key Delegation. Once we have the key assignment algorithm, the key delegation is straightforward. The user specifies a sub-range of leaves for delegation. Then the user can produce the corresponding secret keys for those leaves. For example, in Fig. 2, suppose Alice wants to delegate leaves in the range [2:3] to Bob. Then Alice simply provide Bob with keys $\mathsf{SK}_{[0:7]}, \mathsf{SK}_{[0:3]}, \mathsf{PSK}_{[2:3]}$. Later, if Bob wants to delegate leaves [2:2] to Carlo, then Bob derives the secret key $\mathsf{SK}_{[2:3]}$ and $\mathsf{SK}_{[2:2]}$ by using $\mathsf{PSK}_{[2:3]}$ and provides Carlo with keys $\mathsf{SK}_{[0:7]}, \mathsf{SK}_{[0:3]}, \mathsf{SK}_{[2:3]}, \mathsf{SK}_{[2:2]}$.

Encryption. We use the same algorithm in the normal CS method [32] to encrypt the message. Given a set of leaves associated with all revoked users, the publisher figures out a set of complete subtrees covering all the non-revoked leaves and uses the pattern of root nodes of those complete subtrees to encrypt the message. For example, in Fig. 2, if Alice is revoked, the publisher encrypts the message msg as follows.

$$\mathsf{Ciphertext}_0 \leftarrow \mathsf{WIBE.Enc}(\mathsf{ibpp}, P_{[0:1]}, \mathsf{msg})$$
$$\mathsf{Ciphertext}_1 \leftarrow \mathsf{WIBE.Enc}(\mathsf{ibpp}, P_{[6:7]}, \mathsf{msg})$$

We can see that only unrevoked users have secret keys for [0:1] and [6:7] to encrypt the message.

Complexity Analysis. Now we analyze the storage cost for the delegable CS method. As we discussed, each user only holds $O(\log(k))$ proto-keys, since a range of leaves can be partitioned into $O(\log(k))$ complete subtrees. Also, there are $O(\log(n))$ ancestor nodes for those complete subtrees in total, and the user needs to hold secret keys for each of those ancestor nodes. Therefore, the overall storage cost for private keys is $O(\log(k) + \log(n))$, where k is the number leaves assigned to the user, and n is the total number of leaves. For encryption, as shown in the original CS method [32], the ciphertext is of size $O(r \log(\frac{n}{r}))$ where r is the number of revoked leaves.

3.3 Delegable Subset Difference method

In this section, we present how to extend the Subset Difference method to support delegation. We will follow the same idea used in the CS method to extend the SD method.

Key Assignment. Similarly, we associate each user with multiple leaves in the broadcast tree and let the user hold the union set of secret keys corresponding to each leaf. However, we need to design a new algorithm to compress secret key storage since the key assignment in the SD method is much more complicated than that in the CS method. In particular, we can no longer use HIBE to compress secret key storage for the delegable SD method.

Recall that in the SD method, we have the covering subset family $\mathcal{S}$, where for each $S_{ij} \in \mathcal{S}$, it contains all the leaves in the subtree rooted at the node v_i, but not the subtree rooted at the node v_j. We call v_i the *primary root*and v_j the *secondary root* of S_{ij}. Note that the secondary root must be inside the subtree rooted at the primary root. We assign secret keys for each covering subset, and each leaf is corresponding to those secret keys associated with the subsets, which cover that leaf. We denote D the maximum depth of the broadcast tree. We define the pattern P_{ij} for a subset S_{ij} as follows:

1. for $0 \leq k < \mathsf{Dep}(v_i)$, $P_{ij}(k) = \mathsf{label}_{v_i}(k)$;
2. for $k = \mathsf{Dep}(v_i)$, $P_{ij}(k) = 2$;
3. for $\mathsf{Dep}(v_i) < k \leq D$, $P_{ij}(k) = \perp$;
4. for $D < k < D + \mathsf{Dep}(v_j)$, $P_{ij}(k) = \mathsf{label}_{v_j}(k)$;
5. for $k = D + \mathsf{Dep}(v_j)$, $P_{ij}(k) = 2$;
6. for $k > D + \mathsf{Dep}(v_j)$, $P_{ij}(k) = \perp$.

And we generate the secret key for the subset S_{ij} as follows:

$$\mathsf{SK}_{ij} \leftarrow \mathsf{WIBE.KeyGen}(\mathsf{ibpp}, \mathsf{ibmk}, P_{ij})$$

For any single leaf, it may be covered by $O(n)$ subsets. Previous works [32] make use of HIBE so that each user associated with a single leaf only requires $O(\log^2(n))$ secret keys. The idea is that if the leaf node is covered by both S_{ij} and S_{ik}, and v_j is the ancestor node of v_k, then we can find a proto-key from which the secret key for S_{ik} can be derived.

However, in order to support delegation, each user may be associated with multiple leaves, which means the user must hold $O(k \log^2(n))$ secret keys, where k is the number leaves associated with the user. We note that HIBE cannot further compress the secret keys in this case because HIBE only supports a single hierarchy. In contrast, we make use of WKD-IBE to support two hierarchies at the same time.

Again, we can find $O(\log(k))$ complete subtrees covering all the leaves corresponding to the user. For each root node v_i of those subtrees, we define a proto-pattern PP_i as follows:

1. for $0 \leq k < \mathsf{Dep}(v_i)$, $PP_i(k) = \mathsf{label}_{v_i}(k)$;
2. for $\mathsf{Dep}(v_i) \leq k \leq D$, $PP_i(k) = \perp$;
3. for $D < k < D + \mathsf{Dep}(v_i)$, $PP_i(k) = \mathsf{label}_{v_i}(k)$;
4. for $k \geq D + \mathsf{Dep}(v_i)$, $PP_i(k) = \perp$.

Also, we generate the proto-key for v_i as follows:

$$\mathsf{PSK}_i \leftarrow \mathsf{WIBE.KeyGen}(\mathsf{ibpp}, \mathsf{ibmk}, PP_i)$$

As we can see, the secret key for any subset S_{i*}, in which the primary root is v_i, can be derived from this proto-key. Although the user may use this proto-key to derive secret keys for the invalid subset, in which the secondary root is not inside the subtree rooted at the primary root, we will see that this does not affect the security of the encryption scheme.

Finally, the user still needs to store secret keys for S_{ij} where v_i is the ancestor node of those complete subtrees, and there exists a leaf node that belongs to the user but is not inside the subtree rooted at v_j. We denote v_L the leftmost leaf and v_R the rightmost leaf of all the leaves that belong to the user. We denote $\mathsf{copath}(v)$ the co-path of the node v. For an ancestor node v_i, we compute the proto-key for the set S_{ij} if v_j is inside the subtree rooted at v_i and one of the following two conditions hold:

- Both the left subtree and right subtree of v_i contains leaves associated with the user, and v_j is the left or right child node of v_i;
- Either the left subtree or the right subtree of v_i contains no leaf associated with the user, and v_j is inside $\mathsf{copath}(v_L)$ or $\mathsf{copath}(v_R)$.

Then, we compute the proto-pattern PP_{ij} for the set S_{ij} as follows:

1. for $0 \leq k < \mathsf{Dep}(v_i)$, $PP_{ij}(k) = \mathsf{label}_{v_i}(k)$;
2. for $k = \mathsf{Dep}(v_i)$, $PP_{ij}(k) = 2$;
3. for $\mathsf{Dep}(v_i) < k \leq D$, $PP_{ij}(k) = \perp$;
4. for $D < k < D + \mathsf{Dep}(v_j)$, $PP_{ij}(k) = \mathsf{label}_{v_j}(k)$;
5. for $k \geq D + \mathsf{Dep}(v_j)$, $P_{ij}(k) = \perp$.

This proto-pattern ensures that the user can only extend the second hierarchy to derive secret keys. We

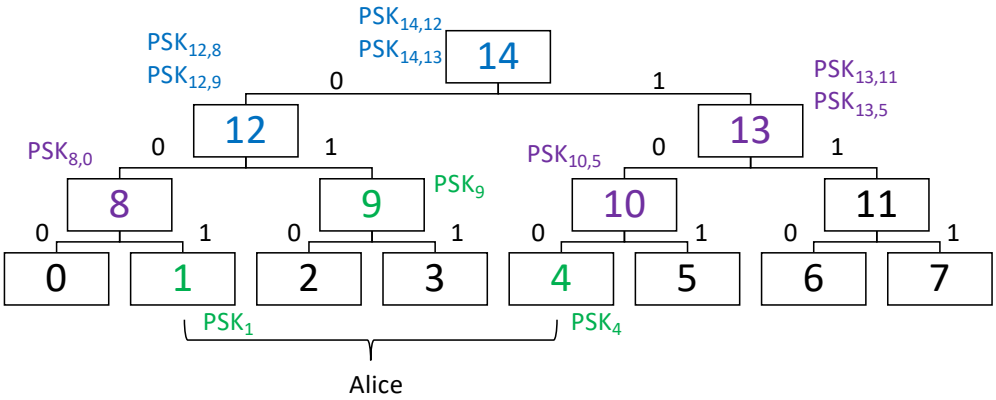

Figure 3: Key management of the delegable SD method. Alice is assigned to a range of leaves [1:4] and is supposed to hold the proto-keys in purple, green, and blue.

Here we show an example of using the delegable SD method. As shown in Fig. 3, Alice is assigned to leaves [1:4]. We can find complete subtrees rooted at v_1, v_9, v_4 to cover leaves [0:2], so Alice holds proto-keys $\mathsf{PSK}_1, \mathsf{PSK}_9, \mathsf{PSK}_4$. The ancestor nodes of those complete subtrees are $v_8, v_{12}, v_{14}, v_{13}, v_{10}$. The left most and right most leaf are v_1 and v_4 respectively. We can find the co-path $\mathsf{copath}(v_1) = \{v_0, v_9, v_{13}\}$ and $\mathsf{copath}(v_4) = \{v_5, v_1 1, v_{12}\}$. Because both the left subtree and the right subtree of v_{12}, v_{14} contain at least one of leaves in [1:4], Alice holds those proto-keys in blue. Because either the left subtree or the right subtree of v_8, v_{10}, v_{13} contains no leaf in [1:4], Alice holds proto-keys in purple. Finally, we can check that Alice can produce secret keys for any leaf associated with her.

Key Delegation. Similarly, the user specifies a sub-range of leaves for delegation and produces the secret keys associated with those leaves as specified in the key assignment algorithm.

Encryption. We use the same algorithm in the SD method [32] to figure out a subset covering $\mathcal{S}$. For any $S_{ij} \in \mathcal{S}$, we encrypts the message using its pattern P_{ij}.

Complexity Analysis. Now we analyze the proto-key storage cost. we can find $O(\log(k))$ complete subtrees covering a range of k leaves. And there are at most $O(\log(n))$ ancestor nodes, and each ancestor node is assigned to $O(\log(n))$ proto-keys because the co-path is of size $O(\log(n))$. Overall, the proto-key storage is of size $O(\log(k) + \log^2(n))$. And the ciphertext is of size $O(r)$ as shown in the original SD method [32], where r is the number of revoked leaves.

3.4 Formal Construction

In this section, we formalize the construction for both delegable CS method and SD method shown in Section 3.2 and Section 3.3. We can see that the key difference between those two methods is the key assignment algorithm So we denote $\mathsf{CS}(\cdot)$ and $\mathsf{SD}(\cdot)$ the key assignment algorithm of delegable CS method and SD method respectively. These function takes in a range of leaf ID and outputs a set of patterns used for key assignment.

Setup. On input the security parameter κ and the total number n of subscribers in the system, we obtain the public parameters and the master key $(\mathsf{ibpp}, \mathsf{ibmk})$ from $\mathsf{WIBE.Setup}(1^\kappa, 1^\ell)$ where $\ell = \log(n) + 1$ if we using CS method, otherwise $\ell = 2 * (\log(n) + 1)$ if we are using SD method. We set $\mathsf{bpp} := \mathsf{ibpp}$ and $\mathsf{bmk} := \mathsf{ibmk}$.

KeyGen. On input the public parameter bpp, the master key bmk, and the identifier set IDs, we first obtain the pattern set $\mathcal{P}$ by using $\mathsf{CS}(\mathsf{IDs})$ or $\mathsf{SD}(\mathsf{IDs})$. For each pattern P inside the pattern set $\mathcal{P}$, we obtain the key as follows:

$$\mathsf{SK}_P \leftarrow \mathsf{WIBE.KeyGen}(\mathsf{bpp}, \mathsf{bmk}, P)$$

And we output the secret key for the identifier set $\mathsf{Key}_{\mathsf{IDs}} := \{\mathsf{SK}_P\}_{P \in \mathcal{P}}$.

KeyDer. On input the public parameter bpp, the decryption key $\mathsf{Key}_{\mathsf{IDs}}$ for identifier set IDs, and another identifier set IDs', we obtain the pattern set $\mathcal{P}$ and $\mathcal{P}'$ for IDs and IDs' respectively by using $\mathsf{CS}(\cdot)$ or $\mathsf{SD}(\cdot)$. For each $P' \in \mathcal{P}'$, we can find $P \in \mathcal{P}$ such that P matches P' And we compute the key as follows:

$$\mathsf{SK}_{P'} \leftarrow \mathsf{WIBE.KeyDer}(\mathsf{bpp}, \mathsf{SK}_P, P')$$

Finally we output the secret key for the identifier set $\mathsf{Key}_{\mathsf{IDs}'} := \{\mathsf{SK}_{P'}\}_{P' \in \mathcal{P}'}$.

Encrypt. On input the public parameter bpp, the revoked identifier set rIDs, and the message m, we figure out a covering set $\mathcal{S}$ by using the encryption algorithm in the original CS or SD method [32]. For each subset $S \in \mathcal{S}$, we produce the pattern P and compute the ciphertext:

$$\mathsf{Ciphertext}_S \leftarrow \mathsf{WIBE.Enc}(\mathsf{bpp}, P, m)$$

We output the ciphertext $\mathsf{Ciphertext}_{\mathsf{rIDs},m} := \{\mathsf{Ciphertext}_S\}_{S \in \mathcal{S}}$.

Decrypt. On input the public parameter bpp, the decryption key $\mathsf{Key}_{\mathsf{IDs}}$, the revoked identifier set rIDs and the ciphertext $\mathsf{Ciphertext}_{\mathsf{rIDs},m}$, we can find an identifier ID such that $\mathsf{ID} \in \mathsf{IDs}$ and $\mathsf{ID} \notin \mathsf{rIDs}$. Again, we figure out a covering set $\mathcal{S}$ by using the encryption algorithm in the original CS or SD method. We can find a subset $S \in \mathcal{S}$ that covers ID. We produce the pattern P for this subset S, and find a key $\mathsf{SK}_{P'} \in \mathsf{Key}_{\mathsf{IDs}}$ such that P' matches P. We obtain the secret key for P as follows:

$$\mathsf{SK}_P \leftarrow \mathsf{WIBE.KeyDer}(\mathsf{bpp}, \mathsf{SK}_{P'}, P)$$

Finally we obtain the message m as follows:

$$m \leftarrow \mathsf{WIBE.Dec}(\mathsf{SK}_P, \mathsf{Ciphertext}_S)$$

3.5 Proof of security

In this section, we prove the security of the delegable broadcast encryption scheme in Section 3.4. We have the following theorem.

Theorem 1. *If the delegable broadcast encryption constructed in Section 3.4 is instantiated with a Selective-ID CPA-secure (Definition 2) WKD-IBE scheme, then it achieves Selective-ID CPA-security (Definition 4).*

Proof. We show that, given an adversary $\mathcal{A}$ which can win the security game in Definition 4 with non-negligible advantage, one can contruct an algorithm $\mathcal{B}$ with non-negligible advantage in the IND-sWKID-CPA security game shown in Definition 2.

We denote $\mathcal{C}$ the challenger in the IND-sWKID-CPA security game. We construct the algorithm $\mathcal{B}$ against the IND-sWKID-CPA security game as follows.

Initialization. $\mathcal{B}$ plays the security game in Definition 4 with $\mathcal{A}$ and acts the challenger. $\mathcal{A}$ gives $\mathcal{B}$ a list $\mathcal{L}$ of challenge identifiers to attack. $\mathcal{B}$ figures out a subset covering $\mathcal{S}$ over the identifier list $\mathcal{L}$ by using the encryption algorithm in the CS or SD method [32]. $\mathcal{B}$ generates the pattern set $\mathcal{P}$ for the subset covering $\mathcal{S}$ as described in Section 3.2 and Section 3.3. $\mathcal{B}$ gives $\mathcal{P}$ to $\mathcal{C}$ to attack.

Setup. $\mathcal{B}$ receives the public parameter ibpp of the WKD-IBEinstance from $\mathcal{C}$. $\mathcal{B}$ set $\mathsf{bpp} = \mathsf{ibpp}$ and sends bpp to $\mathcal{A}$ as the public parameters of the BE instance.

Phase 1. $\mathcal{A}$ may asks $\mathcal{B}$ to provide secret keys for identifier set IDs. Similarly, $\mathcal{B}$ can figure out a pattern set $\mathcal{P}'$ for leaves inside IDs. Because all the identifier $\mathsf{ID} \in \mathsf{IDs}$ may not be in the challenge lists $\mathcal{L}$, the pattern $P \in \mathcal{P}'$ is a also not in the challenge pattern set $\mathcal{P}$. $\mathcal{B}$ asks $\mathcal{C}$ to provide keys for $\mathcal{P}'$, and sends those secret keys to $\mathcal{A}$.

Challenge. Upon receiving the challenge ciphertext from $\mathcal{C}$, $\mathcal{B}$ simply forwards it to $\mathcal{A}$.

Phase 2. $\mathcal{B}$ acts as in the phase 1 to generate secret keys for $\mathcal{A}$.

Guess. $\mathcal{B}$ receives the result bit b from $\mathcal{A}$, and forwards it to $\mathcal{C}$.

Now we can see that the probability that $\mathcal{A}$ wins the security game with $\mathcal{B}$ is equal to the probability that $\mathcal{B}$ wins the security game with $\mathcal{C}$. Therefore, if there exists an adversary $\mathcal{A}$, which can win the security game in Definition 4 with a non-negligible advantage, then we also have the algorithm $\mathcal{B}$ which can win the security game in Definition 2 with non-negligible advantage. □

4 Immediate revocation in JEDI

In this section, we present how to design the immediate revocation protocol in JEDI [49] by leveraging the delegable broadcast encryption scheme in Section 3. JEDI is a many-to-many end-to-end encryption protocol for IoT. JEDI encrypts and signs messages end-to-end while conforming to the decoupled communication model typical of IoT systems. Our revocation protocol must also fit the paradigm of IoT systems. In this section, we first summarize JEDI protocol design in Section 4.1. Please refer to [49] for more details. Then we introduce the revocation protocol of JEDI in Section 4.2.

4.1 Overview of JEDI

JEDI is a many-to-many end-to-end encryption protocol compatible with the paradigm of IoT systems. JEDI fulfills the following three central requirements for IoT:

Decoupled senders and receivers. IoT-scale systems could consist of thousands of principals, making it infeasible for consumers of data (e.g., applications) to maintain a separate session with each producer of data (e.g., sensors). Instead, senders are typically **decoupled** from receivers. Such decoupling is common in *publish-subscribe* systems for IoT, such as MQTT, AMQP, XMPP, and Solace [67]. Senders publish messages by addressing them to *resources* and sending them to a *router*. Recipients *subscribe* to a resource by asking the router to send them messages addressed to that resource.

Many systems for smart buildings/cities, like sMAP [31], SensorAct [8], bw2 [6], VOLTTRON [71], and BAS [48], organize resources as a **hierarchy**. In particular, we represent each resource—a leaf in the hierarchy—as a Uniform Resource Indicator (**URI**), which is like a file path.

Decentralized delegation. Access control in IoT needs to be fine-grained. For example, if Bob has an app that needs access to temperature readings from a single sensor, that app should receive the decryption key for only that one URI, even if Bob has keys for the entire room. In an IoT-scale system, it is not scalable for a central authority to individually give fine-grained decryption keys to each person's devices. Instead, a principal with access to a set of resources can give another principal access to a subset of those resources without contacting a centralized authority.

Resource constraints. IoT devices vary greatly in their capabilities, as shown in Fig. 4. This includes devices constrained in CPU, memory, and energy, such as wearable devices and low-cost environmental sensors.

Figure 4: IoT comprises a diverse set of devices, which span more than four orders of magnitude of computing power (estimated in Dhrystone MIPS).[1]

[1]Image credits: https://tweakers.net/pricewatch/1275475/asus-f540la-dm1201t.html, https://www.lg.com/uk/mobile-phones/lg-H791, https://www.bestbuy.com/site/nest-learning-thermostat-3rd-generation-stainless-steel/4346501.p?skuId=4346501, https://www.macys.com/shop/product/fitbit-charge-2-heart-rate-fitness-wristband?ID=2999458

To achieve this price/power point, sensor platforms are heavily resource-constrained, with mere *kilobytes* of memory (farthest right in Fig. 4) [42, 61, 35, 51, 25, 5, 4]. The *power consumption* of encryption is a serious challenge, even more so than its latency on a slower CPU; the CPU and radio must be used sparingly to avoid consuming energy too quickly [79, 47].

JEDI encrypts messages end-to-end for confidentiality, signs them for integrity while preserving anonymity, and supports delegation with caveats, all while allowing senders and receivers to be decoupled via a resource hierarchy. JEDI is built on top of WKD-IBE, which makes it feasible to deploy our delegable broadcast encryption scheme.

4.1.1 JEDI's System Model (Section 2 in [49])

Participants in JEDI are called *principals*. Any principal can create a **resource hierarchy** to represent some resources that it owns. Because that principal owns all of the resources in the hierarchy, it is called the *authority* of that hierarchy.

Due to the setting of **decoupled senders and receivers**, the sender can no longer encrypt messages with the receiver's public key, as in traditional end-to-end encryption. Instead, JEDI models principals as interacting with resources, rather than with other principals. Herein lies the key difference between JEDI's model and other end-to-end encryption protocols: the publisher of a message encrypts it according to the URI to which it is published, not the recipients subscribed to that URI. Only principals permitted to subscribe to a URI are given keys that can decrypt messages published to that URI.

IoT systems that support **decentralized delegation** (Vanadium, bw2), as well as related non-IoT authorization systems (e.g., SPKI/SDSI [28] and Macaroons [14]) provide principals with tokens (e.g., certificate chains) that they can present to prove they have access to a certain resource. Providing tokens, however, is not enough for end-to-end encryption; unlike these systems, JEDI allows *decryption keys* to be distributed via chains of delegations. Furthermore, the URI prefix and expiry time associated with each JEDI key can be restricted at each delegation. For example, as shown in Fig. 1, suppose Alice, who works in a research lab, needs access to sensor readings in her office. In the past, the campus facilities manager, who is the authority for the hierarchy, granted a key for `buildingA/*` to the building manager, who granted a key for `buildingA/floor2/*` to the lab director. Now, Alice can obtain the key for `buildingA/floor2/alice_office/*` directly from her local authority (the lab director).

4.1.2 Encryption with URIs and Expiry (Section 3 in [49])

JEDI supports *decoupled* communication. The resource to which a message is published acts as a *rendezvous point* between the senders and receivers, used by the underlying system to route messages. Central to JEDI is the challenge of finding an analogous *cryptographic rendezvous point* that senders can use to encrypt messages without knowledge of receivers. A number of IoT systems [66, 62] use only simple cryptography like AES, SHA2, and ECDSA, but these primitives are not expressive enough to encode JEDI's rendezvous point, which must support hierarchically-structured resources, non-interactive expiry, and decentralized delegation.

Existing systems [72, 73, 74] with similar expressivity to JEDI use Attribute-Based Encryption (ABE) [41, 13]. Unfortunately, ABE is not suitable for JEDI because it is too expensive, especially in the context of **resource constraints** of IoT devices. Some IoT systems rule it out due to its latency alone [66]. In the context of low-power devices, encryption with ABE would also consume too much power. JEDI circumvents the problem of using ABE or basic cryptography with two insights: (1) Even though ABE is too heavy for low-power devices, this does not mean that we must resort to only symmetric-key techniques. We show that certain IBE schemes [1] can be made practical for such devices. (2) **Time is another resource hierarchy**: a timestamp can be expressed as `year/month/day/hour`, and in this hierarchical representation, any time range can be represented efficiently as a logarithmic number of subtrees. With this insight, we can simultaneously support URIs and expiry via a nonstandard use of a certain type of IBE scheme: WKD-IBE [1]. Like ABE,

WKD-IBE is based on bilinear groups (pairings), but it is an order-of-magnitude less expensive than ABE as used in JEDI. To make JEDI practical on low-power devices, we design it to invoke WKD-IBE *rarely*, while relying on AES most of the time, much like session keys. Thus, JEDI achieves expressivity commensurate to IoT systems that do not encrypt data—significantly more expressive than AES-only solutions—while allowing several years of battery life for low-power low-cost IoT devices.

4.1.3 Integrity and Anonymity (Section 4 in [49])

In addition to being encrypted, messages should be signed so that the recipient of a message can be sure it was not sent by an attacker. This can be achieved via a certificate chain, as in SPKI/SDSI or bw2. Certificates can be distributed in a decentralized manner, just like encryption keys in Fig. 1.

Certificate chains, however, are insufficient if anonymity is required. For example, consider an office space with an occupancy sensor in each office, each publishing to the same URI `buildingA/occupancy`. In aggregate, the occupancy sensors could be useful to inform, e.g., heating/cooling in the building, but individually, the readings for each room could be considered privacy-sensitive. The occupancy sensors in different rooms could use different certificate chains, if they were authorized/installed by different people. This could be used to deanonymize occupancy readings. To address this challenge, we adapt the WKD-IBE scheme that we use for end-to-end encryption to achieve an *anonymous* signature scheme that can encode the URI and expiry and support decentralized delegation. Using this technique, anonymous signatures are practical even on low-power embedded IoT devices.

4.2 Immediate revocation protocol design

In this section, we present how to achieve revocation in WKD-IBE. A simple solution for revocation is to rely on expiration. In this solution, all keys are time-limited, and delegations are periodically refreshed, according to a higher layer protocol, by granting a new key with a later expiry time. In this setup, the principal who granted a key can easily revoke it by not refreshing that delegation when the key expires. We expect this solution to be sufficient for many applications of JEDI.

Some disadvantages of expiration are that (1) principals must periodically come online to refresh delegations, and (2) revocation only takes effect when the delegated key expires. We would like a solution without these disadvantages.

However, any revocation scheme that does not wait for keys to expire is subject to a set of *inherent* limitations. The recipient of the revoked delegation still has the revoked decryption key, so it can still decrypt messages encrypted in the same way. This means that we must either (1) rely on intermediate parties to modify ciphertexts so that revoked keys cannot decrypt them, or (2) require senders to be aware of the revocation, and encrypt messages in a different way so that revoked keys cannot decrypt them. Neither solution is ideal: (1) makes assumptions about how messages are delivered, which we have avoided thus far, and requires trust in an intermediary to modify ciphertexts, and (2) weakens the decoupling of senders and receivers. We adopt the second compromise: while senders will not need to know who are the receivers, they will need to know who has been revoked.

As discussed in Section 4.1.2, JEDI uses WKD-IBE in a way that provides concurrent hierarchies for both the URI hierarchy and time hierarchy. As we have shown in Section 3, the key idea behind delegable broadcast encryption is to use WKD-IBE in a way that organizes users' secret keys in a hierarchy. Therefore, we can instantiate a third hierarchy in JEDI and use it for revocation.

Each secret key in JEDI will be assigned a tuple $(\mathsf{URI}, \mathsf{Time}, \mathsf{IDs})$, which indicates the resource URI, the time interval, and identifiers of leaves in the broadcast tree respectively. Let $\ell = \ell_1 + \ell_2 + \ell_3$ be the pattern length in the hierarchy's WKD-IBE system. We use the first ℓ_1 slots to encode the URI, and the second ℓ_2 slots to encode the Time (see details in Section 3.4 of [49]). Similarly, we use the last ℓ_3 slots to instantiate the delegable broadcast encryption schemes in Section 3. In particular, if we are using CS method in Section 3.2,

then $\ell_3 = \log(n) + 1$, otherwise $\ell_3 = 2 * (\log(n) + 1)$ for SD method in Section 3.3.

During the delegation, the user should produce secret keys with more restricted tuple $(\mathsf{URI}', \mathsf{Time}', \mathsf{IDs}')$, in which the resource with URI' is under the hierarchy of the resource with URI; the time interval Time' is inside the time interval Time; the leaves for IDs' is a sub-range of leaves for IDs. When encrypting a message, senders use the same encryption protocol from Section 4.1.2 for the first $\ell_1 + \ell_2$ slots, and repeat the process, filling in the remaining ℓ_3 slots with the ID used for broadcast encryption as described in Section 3.

Now we analyze the cost complexity. Let r be the number of revoked keys. The CS method has $O(r \log \frac{n}{r})$-size ciphertexts and the SD method has $O(r)$-size ciphertexts, so JEDI ciphertexts grow to these sizes when revocation is used. If we are using the delegable CS method, the JEDI keys require a total of $O((\log k + \log n) \cdot \log T)$ WKD-IBE keys, where T is the length of the time range for expiry and k is the number of node identifiers. As for the delegable SD method, the JEDI keys require a total of $O((\log k + \log^2 n) \cdot \log T)$ WKD-IBE keys.

Although we can instantiate the revocation protocol with either the CS or SD method, we notice that the SD method requires a larger secret key size. In IoT devices, the storage is usually limited. Therefore, we choose to use the CS method in JEDI.

The construction in this section works to revoke decryption keys, but cannot be used with anonymous signatures (Section 4.1.3). Extensions of tree-based broadcast encryption to signatures exist [52, 53], and we expect them to be useful to develop a construction for anonymous signatures.

How can JEDI inform publishers which leaves are revoked? One simple option is to have a global revocation list, which principals can append to. However, storing this information in a single list becomes a central point of attack, which we have avoided in our system. To avoid this, one can store the revocation list in a global-scale blockchain, such as Bitcoin or Ethereum, which would require an adversary to be exceptionally powerful to mount a successful attack. When revoking a set of leaves, a principal uses those keys to sign a predetermined object (as in Section 4.2 of [49]), proving it owns an ancestor of that key in the hierarchy. To keep the revocation list private, one can use JEDI's encryption to ensure that only principals with permission to publish to a particular resource can see which keys are revoked for that resource (since publishers too have signing keys, as described in Section 4.1.3).

4.2.1 Security Guarantee

In this section, we formalize the security definition and proof for the revocation protocol in Section 4.2. This is also shown in Appendix E.3 of [49].

Theorem 2. *Suppose revocation in JEDI is instantiated with a Selective-ID CPA-secure [26, 17, 1], history-independent WKD-IBE scheme. Then, there exists no probabilisitc polynomial-time adversary $\mathcal{A}$ who can win the following security game against a challenger $\mathcal{C}$ with more than negligible advantage:*
Initialization. *$\mathcal{A}$ selects a (URI, time) pair and a list $\mathcal{L}$ of revoked leaves to attack.*
Setup. *$\mathcal{C}$ gives $\mathcal{A}$ the public parameters of the JEDI instance.*
Phase 1. *$\mathcal{A}$ can make three types of queries to $\mathcal{C}$.*
1. $\mathcal{A}$ asks $\mathcal{C}$ to create a principal; $\mathcal{C}$ returns a name in $\{0,1\}^*$, which $\mathcal{A}$ can use to refer to that principal in future queries. A special name exists for the authority.
2. $\mathcal{A}$ can ask $\mathcal{C}$ for the key set of any principal; $\mathcal{C}$ gives $\mathcal{A}$ the keys that the principal has. The only restriction is that, at the time this query is made, every key in the requested key set whose URI and time are prefixes of the challenge (URI, time) must have leaves that are entirely in the challenge list $\mathcal{L}$.
3. $\mathcal{A}$ can ask $\mathcal{C}$ to make any principal to make a delegation of $\mathcal{A}$'s choice to another principal. The new principal may have restricted pattern or fewer leaves.
Challenge. *When $\mathcal{A}$ chooses to end Phase 1, it sends $\mathcal{C}$ two messages, m_0 and m_1, of the same length. Then $\mathcal{C}$ chooses a random bit $b \in \{0,1\}$, encrypts m_b under the challenge (URI, time) pair and list $\mathcal{L}$ of revoked*

leaves, and gives $\mathcal{A}$ all of the ciphertexts.
Phase 2. *$\mathcal{A}$ can make additional queries as in Phase 1.*
Guess. *$\mathcal{A}$ outputs $b' \in \{0,1\}$, and wins the game if $b = b'$. The advantage of an adversary $\mathcal{A}$ is $\left|\Pr[\mathcal{A}\ \textit{wins}] - \frac{1}{2}\right|$.*

Proof. We use a hybrid argument. The hybrid game $\mathcal{H}^{(0)}$ is one that the adversary $\mathcal{A}$ has no chance of winning. Hybrid $\mathcal{H}^{(n)}$ is a game that perfectly simulates WKD-IBE's revocation protocol. We prove using the security of the underlying WKD-IBE scheme that for all $i \in \{1, ..., n\}$, the difference between $\mathcal{A}$'s advantage in hybrid game $\mathcal{H}^{(i-1)}$ and hybrid game $\mathcal{H}^{(i)}$ is negligible.

The number n, which controls the number of hybrids, is defined to be the size of the subset cover, which depends on the list $\mathcal{L}$ that $\mathcal{A}$ declares in the Initialization phase. Note that this is the same as the number of WKD-IBE ciphertexts in an encrypted message with revocation list $\mathcal{L}$.

We define the hybrid game $\mathcal{H}^{(i)}$, for $i \in \{0, ..., n\}$, as identical to the game given in Theorem 2, with the following difference. The JEDI ciphertext returned to $\mathcal{A}$ in the Challenge phase consists of n WKD-IBE ciphertexts; $\mathcal{H}^{(i)}$ generates the first i ciphertexts correctly, and then replaces each of the remaining $n - i$ ciphertexts with an encryption of 0 under the same pattern. Observe that $\mathcal{A}$ has no chance of winning $\mathcal{H}^{(0)}$, because the challenge ciphertext is chosen independently of the bit b chosen by the challenger. Also, observe that $\mathcal{H}^{(n)}$ is identical to the game described in Theorem 2.

All that remains to prove is that the difference between $\mathcal{A}$'s advantage in game $\mathcal{H}^{(i-1)}$ and game $\mathcal{H}^{(i)}$ is negligible for all $i \in \{1, \ldots, n\}$. We do so via a reduction: given a probabilistic polynomial-time adversary $\mathcal{A}$ whose difference in advantage is non-negligible, we construct a probabilistic polynomial-time adversary $\mathcal{B}$ that wins the IND-sWKID-CPA game with non-negligible probability. In our reduction, $\mathcal{B}$ acts as the challenger in the hybrid game; we denote the challenger in the IND-sWKID-CPA game as $\mathcal{C}$.

In the Initialization phase, $\mathcal{A}$ specifies the pair (URI, time) and revocation list $\mathcal{L}$ that it will attack. Then, $\mathcal{B}$ computes the subset cover over all leaves not in $\mathcal{L}$, and selects ID_i, the ID of ith subset in the subset cover. $\mathcal{B}$ parses (URI, time) and ID_i into the pattern S^* and gives it to $\mathcal{C}$. $\mathcal{C}$ generates the master key pair $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}$ and gives B the master public key mpk. $\mathcal{B}$ forwards mpk to $\mathcal{A}$.

For any of three queries from $\mathcal{A}$ in Phase 1, $\mathcal{B}$ processes it as following:

- $\mathcal{A}$ asks $\mathcal{B}$ to create a principal: $\mathcal{B}$ returns a fresh name in $\{0,1\}^*$ corresponding to the new principal. $\mathcal{B}$ creates mappings from this name to an empty set, for both the key set and pattern set, indicating that this new principal has not been delegated any keys.
- $\mathcal{A}$ asks $\mathcal{B}$ for the key set of a principal p: $\mathcal{B}$ finds in its local state the key set and pattern set for p. For each pattern in p's pattern set, it queries $\mathcal{A}$ for the corresponding WKD-IBE secret key. It adds each WKD-IBE secret key to p's key set, and then replaces p's pattern set in its local state with an empty set. Then it returns the keys in p's key set to $\mathcal{A}$. Note that $\mathcal{B}$ will not query $\mathcal{C}$ the secret key for a pattern that matches S^*, because $\mathcal{A}$ is not allowed to request a key set containing a key whose URI and time match the challenge pair (URI, time) and the leaves must be in the challenge list $\mathcal{L}$. Also note that the keys given to $\mathcal{A}$ are distributed exactly as they would be in the JEDI protocol, because the underlying WKD-IBE scheme is assumed to be history-independent (Definition 3).
- $\mathcal{A}$ asks an principal p to make a delegation of $\mathcal{A}$'s choice of another principal q and specifies which leaves are included in the delegation: $\mathcal{B}$ finds in its local state the key set and pattern set for p. $\mathcal{B}$ obtains the pattern corresponding to each key in p's key set. Let M be the set containing those patterns. $\mathcal{B}$ computes the set N, which is the union of M and p's pattern set. Based on the patterns in N, $\mathcal{B}$ computes the patterns corresponding to the keys that p would generate and delegate to q. For each such key, $\mathcal{B}$ adds the corresponding pattern to q's pattern set.

At the end of Phase 1, $\mathcal{A}$ outputs two equal-length challenge messages m_0 and m_1, and sends them to $\mathcal{B}$. $\mathcal{B}$ then chooses a random bit b^*. $\mathcal{B}$ computes the JEDI ciphertext, which consists of n WKD-IBE ciphertexts as follows. To compute the jth WKD-IBE ciphertext, where $1 \leq j \leq i-1$, it encrypts 0 with the pattern corresponding to the challenge (URI, time) and ID_j. To compute jth WKD-IBE ciphertext, where $i+1 \leq j \leq n$, it encrypts m_{b^*} with the pattern corresponding to the challenge (URI, time) and ID_j. To compute the ith WKD-IBE ciphertext, it forwards 0 and m_{b^*} to $\mathcal{C}$. $\mathcal{C}$ chooses a random bit b, and sends $\mathcal{B}$ either an encryption of 0 or an encryption of m_{b^*} depending on b.the ciphertext of m_b. $\mathcal{B}$ uses this as the ith ciphertext. It assembles the n WKD-IBE ciphertexts, computed as above, into a JEDI ciphertext, and forwards them to $\mathcal{A}$. Note that if $b = 0$, then $\mathcal{B}$ played game $\mathcal{H}^{(i-1)}$, and if $b = 1$, then $\mathcal{B}$ played game $\mathcal{H}^{(i)}$.

In Phase 2, $\mathcal{A}$ makes additional queries as in Phase 1, and $\mathcal{C}$ can process them as before.

Finally, $\mathcal{A}$ will return the bit b'. $\mathcal{B}$ checks if $b' = b^*$. If $b' = b^**$, then $\mathcal{B}$ guesses that $b = 1$; otherwise, it guesses that $b = 0$. It sends its guess to $\mathcal{C}$. Because $\mathcal{A}$ is assumed to have a non-negligible difference in advantage between $\mathcal{H}^{(i-1)}$ and $\mathcal{H}^{(i)}$, $\mathcal{B}$'s advantage in the IND-sWKID-CPA game is non-negligible. □

4.2.2 Optimizing JEDI's Immediate Revocation

A single JEDI ciphertext, with revocation enabled, consists of $O(r \log \frac{n}{r})$ WKD-IBE ciphertexts if we are using the CS method. To compute them efficiently, we observe that there is a large overlap in the patterns used in individual WKD-IBE encryptions, allowing us to use the "precomputation with adjustment" strategy from Section 3.6.2 of [49].

Even with the above optimization, immediate revocation substantially increases the cost of JEDI's cryptography. To reduce this cost, we make three observations. First, to extend JEDI's hybrid encryption to work with revocation, it is sufficient to additionally rotate keys whenever the revocation list changes, in addition to the end of each hour (as in Section 3.6.1 of [49]). This means that, in the common case where the revocation list does not change in between two messages, efficient symmetric-key encryption can be used. Second, the revocation list used to encrypt a message need only contain revoked leaves for the *particular URI* to which the message is sent. This not only makes the broadcast encryption more efficient (smaller r), but also causes the effective revocation list for a stream of data to change even more rarely, allowing JEDI to benefit more from hybrid encryption. Third, we can do the same thing as above using the expiry time rather than the URI, allowing us to *cull* the revocation list by removing keys from it once they expire.

The efficiency of hybrid encryption depends on the revocation list changing *rarely*. We believe this is a reasonable assumption; most revocation will be handled by expiry, so immediate revocation is only needed if a principal must lose access *unexpectedly*. In the smart buildings use case, for example, a key would need to be revoked if a principal unexpectedly transfers to another job.

5 Evaluation

We evaluate revocation protocol in JEDI via microbenchmarks, determine its power consumption on a low-power sensor. We measure the overhead of applying JEDI to bw2 [3], and compare it to other systems in the full paper [49].

5.1 Implementation

We implemented JEDI as a library in the Go programming language. Our implementation makes anonymous signatures in Section 4.1.3 optional and implements revocation separately. We expect JEDI's key delegation to be computed on relatively powerful devices, like laptops, smartphones, or Raspberry Pis; less powerful devices (e.g., right half of Fig. 4) will primarily send and receive messages, rather than generate keys for delegation. Therefore, our focus for low-power platforms was on the "sense-and-send" use case [25, 33, 35] typical of indoor environmental sensing, where a device periodically publishes sensor readings to a URI. Whereas our Go library provides higher-level abstractions, we expect low-power devices to use JEDI's crypto library directly.

Evaluation Setup. Benchmarks labeled "Laptop" were produced on a Lenovo T470p laptop with an Intel Core i7-7820HQ CPU @ 2.90 GHz. Benchmarks labeled "Raspberry Pi" were produced on a Raspberry Pi 3 Model B+ with an ARM Cortex-A53 @ 1.4 GHz.

5.2 Performance of BLS12-381 in JEDI

Table 1 compares the performance of JEDI's BLS12-381 implementation on the three platforms, with our assembly optimizations. As expected from Fig. 4, the Raspberry Pi performance is an order of magnitude slower than Laptop performance, and performance on the Hamilton sensor is additional two-to-three orders of magnitude slower.

Operation	Laptop	Rasp. Pi	Sensor
$\mathbb{G}_1$ Mul. (Chosen Scalar)	109 µs	1.33 ms	509 ms
$\mathbb{G}_2$ Mul. (Chosen Scalar)	343 µs	3.86 ms	1.44 s
$\mathbb{G}_T$ Mul. (Rand. Scalar)	504 µs	5.47 ms	1.90 s
$\mathbb{G}_T$ Mul. (Chosen Scalar)	507 µs	5.48 ms	2.81 s
Pairing	1.29 ms	14.0 ms	3.83 s

Table 1: Latency of JEDI's implementation of BLS12-381

5.3 Performance of WKD-IBE in JEDI

In Table 2, we used a pattern of length 20 for all operations, which would correspond to, e.g., a URI of length 14 and an Expiry hierarchy of depth 6. To measure decryption and signing time, we measure the time to decrypt the ciphertext or sign the message, plus the time to generate a decryption key for that pattern or ID. For example, if one receives a message on `a/b/c/d/e/f`, but has the key for `a/*`, he must generate the key for `a/b/c/d/e/f` to decrypt it.

Table 2 demonstrates that the JEDI encrypts and signs messages and generates qualified keys for the delegation at practical speeds. On a laptop, all WKD-IBE operations take less than 10 ms with up to 20 attributes. On a Raspberry Pi, they are 10x slower (as expected), but still run at interactive speeds.

5.4 Performance of Immediate Revocation in JEDI

Fig. 5 shows the cost of JEDI's immediate revocation protocol (Section 4). A private key containing k leaves consists of $O(\log k + \log n)$ WKD-IBE secret keys where n is the total number of leaves. Therefore, the performance of immediate revocation depends primarily on the number of leaves.

	Laptop	Rasp. Pi
Enc.	3.08 ms	37.3 ms
Dec.	3.61 ms	43.9 ms
KeyD.	4.77 ms	58.5 ms
Sign	4.80 ms	61.2 ms
Verify	4.78 ms	56.3 ms

Table 2: Latency of **Encrypt**, **Decrypt**, **KeyDer**, **Sign**, and **Verify** with 20 attributes

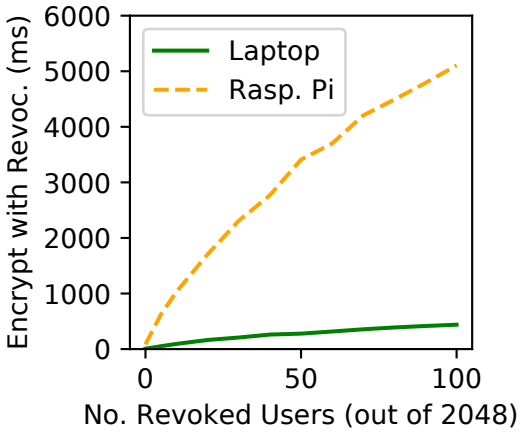

Figure 5: Encryption with Revocation

To encrypt a message, one WKD-IBE encryption is performed for each subtree needed to cover all unrevoked leaves. In general, encryption is $O(r \log \frac{n}{r})$, where r is the number of revoked leaves. Each key contains a set of *consecutive* leaves, so encryption is also $O(R \log \frac{n}{R})$, where R is the number of revoked JEDI keys. Decryption time remains almost the same, since only one WKD-IBE decryption is needed.

To benchmark revocation, we use a complete binary tree of depth 16 ($n = 65536$). The time to generate a new key for delegation is essentially independent of the number of leaves conveyed in that key, because $\log k \ll \log n$. We empirically confirmed this; the time to generate a key for the delegation was constant at 2.4 ms on a laptop and 31 ms on a Raspberry Pi as the number of leaves in the key was varied from 5 to 1,000.

To benchmark encryption with revocation, we assume that there exist 2,048 users in the system, each with 32 leaves. We measure encryption time with a pattern with 20 fixed slots (for URI and time) as we vary the number of revoked users. Fig. 5 shows that encryption becomes expensive when the revocation list is large (500 milliseconds on laptop and ≈ 5 seconds on Raspberry Pi). However, such encryption only needs to be performed by a publisher when the URI, time, or revocation list changes; subsequent messages can reuse the underlying symmetric key (Section 4.2.2). Furthermore, the revocation list includes only revoked keys that match the (URI, time) pair being used, so it is not expected to grow very large.

6 Related Work

We organize related work into the following categories.

Traditional Public-Key Encryption. SiRiUS [40] and Plutus [46] are encrypted filesystems based on traditional public-key cryptography, but they do not support delegable and qualifiable keys like JEDI. Akl et al. [2] and further work [29, 30] propose using key assignment schemes for access control in a hierarchy. A line of work [70, 43, 10, 9] builds on this idea to support both hierarchical structure and temporal access. Key assignment approaches, however, require the full hierarchy to be known at setup time, which is not flexible in the IoT setting. JEDI does not require this, allowing different subtrees of the hierarchy to be managed separately.

Identity-Based Encryption. Tariq et al. [69] use Identity-Based Encryption (IBE) [19] to achieve end-to-end encryption in publish-subscribe systems, without the router's participation in the protocol. However, their approach does not support hierarchical resources. Further, encryption and private keys are on a credential-basis, so each message is encrypted multiple times according to the credentials of the recipients.

Wu et al. [77] use a prefix encryption scheme based on IBE for mutual authentication in IoT. Their prefix encryption scheme is different from JEDI, in that users with keys for identity `a/b/c` can decrypt messages encrypted with prefix identity `a`, `a/b` and `a/b/c`, but not identities like `a/b/c/d`.

Hierarchical Identity-Based Encryption. Since the original proposal of Hierarchical Identity-Based Encryption (HIBE) [39], there have been multiple HIBE constructions [39, 17, 18, 38] and variants of HIBE [78, 1]. Although seemingly a good match for resource hierarchies, HIBE cannot be used as a black box to efficiently instantiate JEDI. We considered alternative designs of JEDI based on existing variants of HIBE, but as we elaborate in the appendix of our extended paper [49], each resulting design is either less expressive or significantly more expensive than JEDI.

Attribute-Based Encryption. A line of work [80, 72] uses Attribute-Based Encryption (ABE) [41, 13] to delegate permission For example, Yu et al. [80] and Sieve [72] use Key-Policy ABE (KP-ABE) [41] to control which principals have access to encrypted data in the cloud. Some of these approaches also provide a means to revoke users, leveraging proxy re-encryption to safely perform re-encryption in the cloud. Our work additionally supports hierarchically-organized resources and decentralized delegation of keys, which [80] and [72] do not address. WKD-IBE is substantially more efficient than KP-ABE and provides enough functionality for JEDI.

Other approaches prefer Ciphertext-Policy ABE (CP-ABE) [13]. Existing work [73, 74] combines HIBE with CP-ABE to produce Hierarchical ABE (HABE), a solution for sharing data on untrusted cloud servers. The "hierarchical" nature of HABE, however, corresponds to the hierarchical organization of domain managers in an enterprise, not a hierarchical organization of *resources* as in our work.

Proxy Re-Encryption. NuCypher KMS [34] allows a user to store data in the cloud encrypted under her public key, and share it with another user using Proxy Re-Encryption (PRE) [15]. While NuCypher assumes limited collusion among cloud servers and recipients (e.g., m of n secret sharing) to achieve properties such as expiry, JEDI enforces expiry via cryptography, and therefore remains secure against *any* amount of collusion. Furthermore, NuCypher's solution for resource hierarchies requires a keypair for each node in the hierarchy, meaning that the creation of resources is centralized. Finally, keys in NuCypher are not qualifiable.

PICADOR [24], a publish-subscribe system with end-to-end encryption, uses a lattice-based PRE scheme. However, PICADOR requires a central Policy Authority to specify access control, by creating a re-encryption key for every permitted pair of publisher and subscriber. In contrast, JEDI's access control is decentralized.

Revocation Schemes. Broadcast encryption (BE) [59, 32, 20, 21, 50, 22, 23] is a mechanism to achieve revocation, by encrypting messages such that they are only decryptable by a specific set of users. However,

these existing schemes do not support key qualification and delegation, and therefore, cannot be used in JEDI directly. Another line of work builds revocation directly into the underlying cryptography primitive, achieving Revocable IBE [16, 54, 64, 75], Revocable HIBE [63, 65, 55] and Revocable KP-ABE [11]. These papers use a notion of revocation in which URIs are revoked. In contrast, JEDI supports revocation at the level of keys. If multiple principals have access to a URI, and one of their keys is revoked, then the other principal can still use its key to access the resource. Some systems [34, 12] rely on the participation of servers or routers to achieve revocation.

Secure Reliable Multicast Protocol. Secure Reliable Multicast [57, 58] also uses a many-to-many communication model, and ensures correct data transfer in the presence of malicious routers. JEDI, as a protocol to *encrypt* messages, is complementary to those systems.

Authorization Services. JEDI is complementary to authorization services for IoT, such as bw2 [6], Vanadium [68], WAVE [7], and AoT [60], which focus on expressing authorization policies and enabling principals to prove they are authorized, rather than on encrypting data. Droplet [66] provides encryption for IoT, but does not support delegation beyond one hop and does not provide hierarchical resources.

An authorization service that provides secure in-band permission exchange, like WAVE [7], can be used for key distribution in JEDI. JEDI can craft keys with various permissions, while WAVE can distribute them without a centralized party by including them in its attestations.

7 Conclusion

In this report, we presented delegable broadcast encryption schemes and showed how to design the immediate revocation protocol in JEDI, a protocol for end-to-end encryption for IoT, based on delegable broadcast encryption schemes. Our revocation scheme supports decentralized key delegation on complex resource hierarchies. More details can be found in our full paper [49].

Availability

The JEDI cryptography library is available at `https://github.com/ucbrise/jedi-pairing` and our implementation of the JEDI protocol for bw2 is available at `https://github.com/ucbrise/jedi-protocol`.

Acknowledgments

I thank the coauthors of JEDI, Sam Kumar, Michael P Andersen, Raluca Ada Popa, and David E. Culler for their efforts in JEDI and this report. I thank JEDI's anonymous reviewers and shepherd William Enck for their invaluable feedback. I would also like to thank students from the RISE Security Group and BETS Research Group for giving feedback on early drafts of JEDI. This research was supported by Intel/NSF CPS-Security #1505773 and #20153754, DoE #DE-EE000768, California Energy Commission #EPC-15-057, NSF CISE Expeditions #CCF-1730628, NSF GRFP #DGE-1752814, and gifts from the Sloan Foundation, Hellman Fellows Fund, Alibaba, Amazon, Ant Financial, Arm, Capital One, Ericsson, Facebook, Google, Intel, Microsoft, Scotiabank, Splunk and VMware.

References

[1] M. Abdalla, E. Kiltz, and G. Neven. Generalized key delegation for hierarchical identity-based encryption. Cryptology ePrint Archive, Report 2007/221.

[2] S. G. Akl and P. D. Taylor. Cryptographic solution to a problem of access control in a hierarchy. *TOCS*, 1983.

[3] M. P. Andersen. BOSSWAVE 2 development. `https://github.com/immesys/bw2`, 2017.

[4] M. P Andersen, G. Fierro, and D. E. Culler. System design for a synergistic, low power mote/BLE embedded platform. In *IPSN*, 2016.

[5] M. P. Andersen, H.-S. Kim, and D. E. Culler. Hamilton - a cost-effective, low power networked sensor for indoor environment monitoring. In *BuildSys*, 2017.

[6] M. P. Andersen, J. Kolb, K. Chen, D. E. Culler, and R. Katz. Democratizing authority in the built environment. In *BuildSys*, 2017.

[7] M. P Andersen, S. Kumar, M. AbdelBaky, G. Fierro, J. Kolb, H.-S. Kim, D. E. Culler, and R. A. Popa. WAVE: A decentralized authorization framework with transitive delegation. In *USENIX Security*, 2019.

[8] P. Arjunan, N. Batra, H. Choi, A. Singh, P. Singh, and M. B. Srivastava. SensorAct: A privacy and security aware federated middleware for building management. In *BuildSys*, 2012.

[9] M. J. Atallah, M. Blanton, N. Fazio, and K. B. Frikken. Dynamic and efficient key management for access hierarchies. In *TISSEC*, 2009.

[10] M. J. Atallah, M. Blanton, and K. B. Frikken. Incorporating temporal capabilities in existing key management schemes. In *ESORICS*, 2007.

[11] N. Attrapadung and H. Imai. Conjunctive broadcast and attribute-based encryption. In *ICPBC*, 2009.

[12] S. Belguith, S. Cui, M. R. Asghar, and G. Russello. Secure publish and subscribe systems with efficient revocation. In *SAC*, 2018.

[13] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *S&P*, 2007.

[14] A. Birgisson, J. G. Politz, Ú. Erlingsson, A. Taly, M. Vrable, and M. Lentczner. Macaroons: Cookies with contextual caveats for decentralized authorization in the cloud. In *NDSS*, 2014.

[15] M. Blaze, G. Bleumer, and M. Strauss. Divertible protocols and atomic proxy cryptography. *EUROCRYPT*, 1998.

[16] A. Boldyreva, V. Goyal, and V. Kumar. Identity-based encryption with efficient revocation. In *CCS*, 2008.

[17] D. Boneh and X. Boyen. Efficient selective-ID secure identity-based encryption without random oracles. In *EUROCRYPT*, 2004.

[18] D. Boneh, X. Boyen, and E.-J. Goh. Hierarchical identity based encryption with constant size ciphertext. In *EUROCRYPT and Cryptology ePrint Archive*, 2005.

[19] D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. In *CRYPTO*, 2001.

[20] D. Boneh, C. Gentry, and B. Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In *CRYPTO*, 2005.

[21] D. Boneh and B. Waters. A fully collusion resistant broadcast, trace, and revoke system. In *CCS*, 2006.

[22] D. Boneh, B. Waters, and M. Zhandry. Low overhead broadcast encryption from multilinear maps. In *CRYPTO*, 2014.

[23] D. Boneh and M. Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. *Algorithmica*, 2017.

[24] C. Borcea, A. B. D. Gupta, Y. Polyakov, K. Rohloff, and G. Ryan. PICADOR: End-to-end encrypted publish-subscribe information distribution with proxy re-encryption. *FGCS*, 2017.

[25] D. Brunelli, I. Minakov, R. Passerone, and M. Rossi. POVOMON: An ad-hoc wireless sensor network for indoor environmental monitoring. In *EESMS*, 2014.

[26] R. Canetti, S. Halevi, and J. Katz. A forward-secure public-key encryption scheme. In *EUROCRYPT*, 2003.

[27] Cisco. The Internet of things reference model. Technical report, Cisco, 2014.

[28] D. Clarke, J.-E. Elien, C. Ellison, M. Fredette, A. Morcos, and R. L. Rivest. Certificate chain discovery in SPKI/SDSI. *Journal of Computer Security*, 2001.

[29] J. Crampton, N. Farley, G. Gutin, M. Jones, and B. Poettering. Cryptographic enforcement of information flow policies without public information. In *ACNS*, 2015.

[30] J. Crampton, K. Martin, and P. Wild. On key assignment for hierarchical access control. In *CSFW*, 2006.

[31] S. Dawson-Haggerty, X. Jiang, G. Tolle, J. Ortiz, and D. E. Culler. sMAP: A simple measurement and actuation profile for physical information. In *SenSys*, 2010.

[32] Y. Dodis and N. Fazio. Public key broadcast encryption for stateless receivers. In *DRM*, 2002.

[33] P. Dutta, D. E. Culler, and S. Shenker. Procrastination might lead to a longer and more useful life. In *HotNets*, 2007.

[34] M. Egorov and M. Wilkison. NuCypher KMS: decentralized key management system. *CoRR*, 2017.

[35] M. C. Feldmeier. *Personalized Building Comfort Control*. PhD thesis, MIT, 2009.

[36] A. Fiat and M. Naor. Broadcast encryption. In *Annual International Cryptology Conference*, 1993.

[37] G. Fierro and D. E. Culler. XBOS: An extensible building operating system. Technical report, EECS Department, University of California, Berkeley, 2015.

[38] C. Gentry and S. Halevi. Hierarchical identity based encryption with polynomially many levels. In *TCC*, 2009.

[39] C. Gentry and A. Silverberg. Hierarchical ID-based cryptography. In *ASIACRYPT*, 2002.

[40] E.-J. Goh, H. Shacham, N. Modadugu, and D. Boneh. SiRiUS: Securing remote untrusted storage. In *NDSS*, 2003.

[41] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *CCS*, 2006.

[42] Hamilton IoT. `https://hamiltoniot.com/`.

[43] H.-F. Huang and C.-C. Chang. A new cryptographic key assignment scheme with time-constraint access control in a hierarchy. *Computer Standards & Interfaces*, 2004.

[44] J. Hviid and M. B. Kjaergaard. Activity-tracking service for building operating systems. In *PerCom*, 2018.

[45] W. Jonker and J-P Linnartz. Digital rights management in consumer electronics products. *IEEE Signal Processing Magazine*, 2004.

[46] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu. Plutus: Scalable secure file sharing on untrusted storage. In *FAST*, 2003.

[47] H.-S. Kim, M. P. Andersen, K. Chen, S. Kumar, W. J. Zhao, K. Ma, and D. E. Culler. System architecture directions for post-SoC/32-bit networked sensors. In *SenSys*, 2018.

[48] A. Krioukov, G. Fierro, N. Kitaev, and D. E. Culler. Building application stack (BAS). In *BuildSys*, 2012.

[49] S. Kumar, Y. Hu, M. P Andersen, R. A. Popa, and D. E. Culler. JEDI: Many-to-many end-to-end encryption and key delegation for iot. In *USENIX Security*, 2019.

[50] A. Lewko, A. Sahai, and B. Waters. Revocation systems with very small private keys. In *S&P*, 2010.

[51] C. Li, Z. Li, M. Li, F. Meggers, A. Schlueter, and H. B. Lim. Energy efficient HVAC system with distributed sensing and control. In *ICDCS*, 2014.

[52] B. Libert, T. Peters, and M. Yung. Group signatures with almost-for-free revocation. In *CRYPTO*, 2012.

[53] B. Libert, T. Peters, and M. Yung. Scalable group signatures with revocation. In *EUROCRYPT*, 2012.

[54] B. Libert and D. Vergnaud. Adaptive-ID secure revocable identity-based encryption. In *CT-RSA*, 2009.

[55] W. Liu, J. Liu, Q. Wu, B. Qin, D. Naccache, and H. Ferradi. Compact CCA2-secure hierarchical identity-based broadcast encryption for fuzzy-entity data sharing. Cryptology ePrint Archive, Report 2016/634.

[56] J. Lotspiech, S. Nusser, and F. Pestoni. Anonymous trust: Digital rights management using broadcast encryption. *Proceedings of the IEEE*, 2004.

[57] D. Malkhi, M. Merritt, and O. Rodeh. Secure reliable multicast protocols in a WAN. *Dist. Computing*, 2000.

[58] D. Malkhi and M. Reiter. A high-throughput secure reliable multicast protocol. *Computer Security*, 1997.

[59] D. Naor, M. Naor, and J. Lotspiech. Revocation and tracing schemes for stateless receivers. In *CRYPTO*, 2001.

[60] A. L. M. Neto, A. L. F. Souza, I. Cunha, M. Nogueira, I. O. Nunes, L. Cotta, N. Gentille, A. A. F. Loureiro, D. F. Aranha, H. K. Patil, and L. B. Oliveira. AoT: Authentication and access control for the entire IoT device life-cycle. In *SenSys*, 2016.

[61] Particle Mesh. `https://www.particle.io/mesh`. Feb. 2, 2019.

[62] A. Perrig, R. Szewczyk, V. Wen, D. E. Culler, and J. D. Tygar. SPINS: Security protocols for sensor networks. In *MobiCom*, 2001.

[63] J. H. Seo and K. Emura. Efficient delegation of key generation and revocation functionalities in identity-based encryption. In *CT-RSA*, 2013.

[64] J. H Seo and K. Emura. Revocable identity-based encryption revisited: Security model and construction. In *PKC*, 2013.

[65] J. H. Seo and K. Emura. Revocable hierarchical identity-based encryption: History-free update, security against insiders, and short ciphertexts. In *CT-RSA*, 2015.

[66] H. Shafagh, L. Burkhalter, S. Duquennoy, A. Hithnawi, and S. Ratnasamy. Droplet: Decentralized authorization for IoT data streams. *CoRR*, 2018.

[67] Solace cloud. `https://solace.com`. Jan. 17, 2018.

[68] A. Taly and A. Shankar. Distributed authorization in Vanadium. In *FOSAD VIII*, 2016.

[69] M. A. Tariq, B. Koldehofe, and K. Rothermel. Securing broker-less publish/subscribe systems using identity-based encryption. *TPDS*, 2014.

[70] W.-G. Tzeng. A time-bound cryptographic key assignment scheme for access control in a hierarchy. *TKDE*, 2002.

[71] VOLTTRON. `https://volttron.org/`. Jan. 23, 2019.

[72] F. Wang, J. Mickens, N. Zeldovich, and V. Vaikuntanathan. Sieve: Cryptographically enforced access control for user data in untrusted clouds. *NSDI*, 2016.

[73] G. Wang, Q. Liu, and J. Wu. Hierarchical attribute-based encryption for fine-grained access control in cloud storage services. In *CCS*, 2010.

[74] G. Wang, Q. Liu, J. Wu, and M. Guo. Hierarchical attribute-based encryption and scalable user revocation for sharing data in cloud servers. *Computers & Security*, 2011.

[75] Y. Watanabe, K. Emura, and J. H. Seo. New revocable IBE in prime-order groups: Adaptively secure, decryption key exposure resistant, and with short public parameters. In *CT-RSA*, 2017.

[76] C. A. Wood and E. Uzun. Flexible end-to-end content security in ccn. In *CCNC*, 2014.

[77] D. J. Wu, A. Taly, A. Shankar, and D. Boneh. Privacy, discovery, and authentication for the Internet of things. In *ESORICS*, 2016.

[78] D. Yao, N. Fazio, Y. Dodis, and A. Lysyanskaya. ID-based encryption for complex hierarchies with applications to forward security and broadcast encryption. In *CCS*, 2004.

[79] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient MAC protocol for wireless sensor networks. In *INFOCOM*, 2002.

[80] S. Yu, C. Wang, K. Ren, and W. Lou. Achieving secure, scalable, and fine-grained data access control in cloud computing. In *INFOCOM*, 2010.