Decentralized Delegation of Encryption Keys on Resource Hierarchies

Sam Kumar, Yuncong Hu, Michael P Andersen, Raluca Ada Popa, David E. Culler
University of California, Berkeley

Abstract

Resource hierarchies are relevant to a diverse set of distributed systems, such as filesystems, the web, and the Internet of Things. Achieving secure syndication on resource hierarchies is challenging because entities interact with resources, not with each other. In this paper, we present a set of cryptographic protocols for end-to-end encryption on resource hierarchies and decentralized delegation of the required encryption keys. At the heart of these protocols is OAQUE, a new encryption scheme. OAQUE extends Hierarchical Identity-Based Encryption to support our protocols, while providing substantially better performance than Attribute-Based Encryption schemes that would otherwise be necessary. Our work is especially relevant to large-scale distributed systems with decentralized trust and hierarchically organized resources.

1 Introduction

People increasingly rely on cloud computing and large-scale distributed systems to share information and communicate with each other. End-to-end encryption would clearly benefit such systems, guaranteeing that users' data remain private even if computer systems are attacked. In this paper, we provide a set of cryptographic protocols for end-to-end encryption that are useful to a broad class of distributed systems.

We motivate our design with the following example. Suppose that a company with 1,000,000 employees stores highly valuable confidential data in a large filesystem. The CEO owns all of the data, and wants to give each employee access to only the files he needs to do his work. The file server is not trusted; the system administrators, who have `root` access, should not be able to see the company's confidential data.

We present five challenges in providing end-to-end encryption in information sharing systems like this one, and our insights in overcoming them. Although we use this specific example to justify meeting these challenges simultaneously, we emphasize that these challenges, and our contributions, are more widely applicable.

Challenge #1: The sender may not know which entities will receive its messages. In our example, if an employee writes to a file, he cannot simply encrypt it under the key of each employee who reads it—there may be multiple readers, and they may not all be known when the file is written. More generally, requiring writers to keep track of each reader limits the system's scalability. For example, in the Internet-of-Things vision, the sheer number of entities, coupled with memory and power constraints, make it infeasible for a producer of data to be aware of all of the entities interested in that data.

Insight #1: We think of entities as interacting with resources, rather than other entities. When an entity writes to a resource, it encrypts the message using the resource's key, rather than the readers' keys. Entities authorized to read a file are given the key to decrypt it.

Challenge #2: Delegation of access to hierarchically organized resources must be scalable. Resource hierarchies are ubiquitous in modern systems: filesystems and Web URLs are examples that people use every day. Even in IoT, hierarchies describe the organization of physical resources: smart cities contain buildings, which contain floors, which contain rooms, which contain sensors [1]. Access to resource subtrees—directories in a filesystem, or rooms in a building—should be delegable as units. In our example, the CEO should be able to delegate access to a directory without providing a separate key for each file in the directory tree.

Insight #2: Resources are represented hierarchically and permissions for subtrees can be delegated succinctly. Like [1], we represent resources as Uniform Resource Indicators (URIs), to capture the hierarchical organization of resources. A URI is like a Web URL or file path—for example, `a/b/c` is a URI. Subtrees in the hierarchy are represented by URI prefixes, such as `a/b/*`. We design our cryptography such that the owner of a hierarchy can grant permission on a URI prefix (an entire subtree) by providing a *single* decryption key. In particular, the granter need not traverse the entire subtree.

Challenge #3: Key delegation must be decentralized. This property is crucial to designing a scalable authorization system. In our example, it would be impractical for the CEO to individually grant each employee access to the files he needs.

Insight #3: Decryption keys are qualifiable and delegable. The owner grants access to a set of resources to some entities, which may delegate access to a subset of those resources to other entities. In our example, the CEO could grant keys to each of his subordinates to access parts of the filesystem. His subordinates can in turn grant keys on smaller subsets to *their* subordinates, and so on. Succinctness is maintained at each step of the way—given the *single* key for `a/b/*`, an entity can generate a *single* key for `a/b/c/*` and delegate it.

Challenge #4: Resource creation must be decentralized. In our example, the CEO should not be the only entity who can create new files; any employee with sufficient permissions should be able to do so.

Insight #4: We leverage Identity-Based Cryptography to handle resources' first use. Identity-Based Encryption allows the public key for a resource to be derived from its URI. All possible URIs in a hierarchy implicitly exist, and can be used for encryption at any time, without a special process to generate a keypair. Each writer can include a signature chain with each write, to prove that it is authorized to write to that resource.

Challenge #5: Delegated permissions should be temporary. In our example, an employee who is given access to a directory to complete a task, should not have access to new files written in that directory after he completes the task. Similarly, a visitor to a smart building should be able to access resources in that building only while she is visiting. Therefore, permissions should *expire* after a certain amount of time.

Insight #5: We provide the construction of OAQUE, a new cryptographic primitive. OAQUE can efficiently and simultaneously support hierarchical resources and expiry.

1.1 Threat Model

Participants in our system, called *entities*, write messages to resources. Some entities are *adversaries* who can observe messages written to all resources. Our goal is *confidentiality*; entities, including adversaries, should only be able read messages written to resources they are authorized to access.

Our threat model is quite standard and is solved with end-to-end encryption. The novelty of our work is that our decryption keys support decentralized delegation (Challenges #3 and #4) and rich semantics (Challenges #2 and #5).

1.2 Our Contributions

We start with a baseline design that addresses only Challenges #1 and #3. The owner (e.g., the CEO) can "create" a resource by generating a keypair. Anyone can write to a resource by encrypting data with its public key. To delegate permission to read a set of resources, the owner provides the private key for each resource. These keys may in turn be delegated to other entities (Challenge #3).

However, addressing Challenges #2, #4, and #5, while continuing to address Challenges #1 and #3, is nontrivial. In the rest of this paper, we build on the baseline design:

1. **Prefix-Based Permissions.** We demonstrate in Section 2 that Hierarchical Identity-Based Encryption captures the hierarchical organization of resources (Challenge #2) while avoiding bottlenecks in resource creation (Challenge #4).

2. **Expiry.** In Section 3, we explain that Expiry (Challenge #5) can also be expressed as a hierarchy. Then we introduce OAQUE, a new encryption scheme, to simultaneously achieve Prefix-Based Permissions and Expiry.

Prefix-Based Permissions and Expiry form the *core* of our protocols; together with the baseline design, they address all five challenges. We also develop two *extensions*:

3. **Revocation with Stateless Receivers.** In Section 4, we use tree-based Broadcast Encryption to make delegations revocable. Using OAQUE, we generalize Broadcast Encryption to support *delegation* in addition to revocation. We use OAQUE again to combine Delegable Broadcast Encryption with our solution for Prefixes and Expiry.

4. **Key Inheritance.** In Section 5 we explain how to use OAQUE to achieve a property called Key Inheritance. Key Inheritance makes an authorization system more flexible and useful, by allowing a grantee to "inherit" keys later delegated by other entities to the granter.

We describe our implementation of these cryptographic protocols in Section 6 and perform microbenchmarks in Section 7. In Section 8 we demonstrate their applicability to existing systems, using three examples: WAVE, a decentralized, blockchain-based syndication and authorization system for IoT; IPFS, a content-addressed, peer-to-peer, distributed filesystem; and Dropbox, a cloud-based file sharing system. We summarize related work in Section 9 and future work in Section 10. Finally, Section 11 concludes the paper.

2 Prefix-Based Permissions

In this section, we briefly explain Identity-Based Encryption (IBE) and Hierarchical Identity-Based Encryption (HIBE). Then, we explain how HIBE can be used to achieve Prefix-Based Permissions. Although our final solution uses OAQUE instead of HIBE, the insights in this section motivate for our final design.

2.1 Identity-Based Encryption

Identity-Based Encryption (IBE) is a type of asymmetric encryption in which public keys have no structure. Any string of bytes may be used as a public key to encrypt a message. The string of bytes used to encrypt messages is called an *identity*, or ID for short. Once a message is encrypted with an ID, the private key for that ID is needed to decrypt the ciphertext. To this end, every IBE cryptosystem has a *master key* which can be used to generate the private key for any ID. The PKG uses the master key to generate private keys for authorized users.

An IBE cryptosystem is a tuple of four algorithms:
$\textbf{Setup}() \rightarrow \mathsf{Params}, \mathsf{MasterKey}$
$\textbf{KeyGen}(\mathsf{Params}, \mathsf{MasterKey}, \mathsf{ID}) \rightarrow \mathsf{Key}_{\mathsf{ID}}$
$\textbf{Encrypt}(\mathsf{Params}, \mathsf{ID}, m) \rightarrow \mathsf{Ciphertext}_{\mathsf{ID},m}$
$\textbf{Decrypt}(\mathsf{Key}_{\mathsf{ID}}, \mathsf{Ciphertext}_{\mathsf{ID},m}) \rightarrow m$

2.2 Hierarchical Identity-Based Encryption

Hierarchical Identity-Based Encryption (HIBE) is an extension of IBE, in which IDs are organized hierarchically. Each ID in HIBE is a vector of byte strings. Furthermore, private keys in HIBE have the following property: given a private key for ID_A, one can generate a private key for ID_B, as long as ID_A is a prefix of ID_B.

For example, `("a", "b", "c")` is an ID of length 3. Given the private key for this ID, one can generate the private key for `("a", "b", "c", "d")` but not for `("a", "b", "cd")` or `("a", "b", "d", "c")`.

A HIBE cryptosystem is a tuple of five algorithms:
$\textbf{Setup}() \rightarrow \mathsf{Params}, \mathsf{MasterKey}$
$\textbf{KeyGen}(\mathsf{Params}, \mathsf{MasterKey}, \mathsf{ID}) \rightarrow \mathsf{Key}_{\mathsf{ID}}$
$\textbf{QualifyKey}(\mathsf{Params}, \mathsf{Key}_{\mathsf{ID}_\mathsf{A}}, \mathsf{ID}_\mathsf{B}) \rightarrow \mathsf{Key}_{\mathsf{ID}_\mathsf{B}}$
$\textbf{Encrypt}(\mathsf{Params}, \mathsf{ID}, m) \rightarrow \mathsf{Ciphertext}_{\mathsf{ID},m}$
$\textbf{Decrypt}(\mathsf{Key}_{\mathsf{ID}}, \mathsf{Ciphertext}_{\mathsf{ID},m}) \rightarrow m$

Note that ID_A must be a prefix of ID_B for **QualifyKey**.

2.3 Prefix-Based Permissions with HIBE

We associate each URI or URI prefix to an ID in HIBE. The ID corresponding to a URI prefix simply contains each component of the URI. For example, the ID corresponding to the URI `a/b/*` is `("a", "b")`. For fully qualified URIs, we add a special termination symbol, denoted as `$`. For example, the ID corresponding to the URI `a/b/d` is `("a", "b", "d", $)`.

Notice that access to an entire subtree of resources is granted by a *single* HIBE private key, and is therefore succinct (Challenge #2). For example, the private key for `a/b/*` is sufficient to generate the private key for any resource (e.g., `a/b/d`) in that subtree. Furthermore, the private key representing a subtree (e.g., `a/b/*`) can be used to generate the private key for any subtree within it (e.g., `a/b/c/*`). This means that decryption keys can be qualified and delegated *while maintaining succinctness* (Challenge #3).Finally, because HIBE is Identity-Based, there is no need for the owner to "create" a resource by generating its keypair. Even if a resource has never been used before, anyone can write to it by encrypting data under the ID corresponding to its URI (Challenge #5).

The authors are not aware of any prior work that uses HIBE for access control in resource hierarchies. The original applications of HIBE were forward-secure encryption, broadcast encryption, and generation of short-lived keys [20, 6, 16]. HIBE was also envisioned to reflect the hierarchical structure of an organization, associating each member with an ID. In contrast, our work deals with *resource* hierarchies, not organizational ones.

3 Expiry

We first explain how to use HIBE to achieve expiry (Section 3.1). However, combining this solution with our solution for Prefix-Based Permissions in Section 2 is nontrivial (Section 3.2). We briefly explore solutions based on Attribute-Based Encryption (ABE), a heavyweight but powerful cryptographic primitive (Sections 3.3 and 3.4). Finally, we introduce OAQUE (Sections 3.5 and 3.6), a novel encryption scheme that allows us to simultaneously achieve Prefix-Based Permissions and Expiry with substantially lower computational overhead than ABE (Section 3.7).

3.1 Expiry with HIBE

To implement expiry with HIBE, we use a technique inspired by the forward-secure encryption scheme presented in [9] and the short-lived key application in [20]: we represent time as a hierarchy. Any timestamp, with the granularity of an hour, can be represented as an ID as `(year, month, day, hour)`. An entity who writes to a resource encrypts the data using the ID corresponding to the current time. To read a resource, an entity must have the key for the time at which it was written.

The hierarchical representation allows permission for an entire year, month, or day, to be granted succinctly. Therefore, to grant permission over a time range, *the* *number of keys granted is logarithmic in the length of the time range*. For example, to delegate permission to read data that was written between November 28, 2014 at 10 PM and March 3, 2017 at 3 AM, the following keys need to be generated: 2014/Nov/28/22, 2014/Nov/28/23, 2014/Nov/28/24, 2014/Nov/29/*, 2014/Nov/30/*, 2014/Dec/*, 2015/*, 2016/*, 2017/Jan/*, 2017/Feb/*, 2017/Mar/01/*, 2017/Mar/02/*, 2017/Mar/03/01, 2017/Mar/03/02, and 2017/Mar/03/03. To support long time ranges, additional levels may be added to the hierarchy, representing decades or centuries.

We can implement expiry by choosing a single start time, and delegating the full time range starting at the start time and ending at the expiry time. In our protocol, however, we support the more powerful semantics described above: the user can specify both the start time and end time for which the delegation is valid.

3.2 Prefixes and Expiry with HIBE

We have shown how to achieve prefix-based permission using HIBE, and how to achieve expiry using HIBE. However, combining these approaches is problematic.

A simple but flawed approach is to encrypt each resource twice, once with the resource ID and again with the time ID. Each delegation contains two decryption keys, one for the resource, and another for the time range. The flaw is that an attacker can "mix-and-match" HIBE keys in different delegations. For example, suppose that an entity has permission to read a/b for 2018/Feb/*, and has permission to read a/c for 2018/Jan/*. A malicious reader could decrypt a message written to a/c in February 2018 by using the expiry key from the first delegation with URI key from the second delegation.

To obtain collusion-resistance and solve this problem, one possible approach is to interleave the resource hierarchy and time hierarchy. In this setup, each (resource, time) pair corresponds to multiple IDs in the HIBE system—all possible interleavings of the URI the Expiry IDs. However, for a URI hierarchy of depth m and a time hierarchy of depth n, there are $\frac{(m+n)!}{m! \cdot n!}$ different interleavings of any particular URI and Time. Therefore, this approach infeasible for deep hierarchies.

3.3 Attribute-Based Encryption

Attribute-Based Encryption (ABE) refers to a class of flexible encryption schemes. There are two types of ABE: Key-Policy ABE (KP-ABE) and Ciphertext-Policy ABE (CP-ABE). In KP-ABE, a message is encrypted with a set of attributes. An attribute set is like a string of bits; for each attribute in the KP-ABE system, it is either present in the set (1) or not present (0). Private keys are generated with an *access tree*, which can be thought of as a circuit containing only AND and OR gates. A private key can be used to decrypt a message if its access tree, when, evaluated on the bits representing the attribute set of the message, evaluates to 1. CP-ABE is the exact opposite: messages are encrypted with an access tree, and keys contain sets of attributes.

We are interested in KP-ABE with two additional properties:

1. **Delegable.** Given the private key for an access tree, one can generate a private key for a more restrictive access key, and delegate it to another entity.

2. **Large Universe.** The space of attributes $\mathscr{A}$ is exponentially large in the security parameter k. This is similar to IBE, as any string of bytes can be hashed to an attribute.

Note that the Large Universe property requires attribute sets to be represented sparsely; we can still logically think of an attribute set as a bitmap describing which attributes are set, but it is only be feasible to set a polynomial number of bits to 1. Similarly, the access tree, which is an AND-OR circuit, can only be polynomially large, and therefore can only depend on a polynomial number of attributes.

Formally, a Delegable Large-Universe KP-ABE cryptosystem is a tuple of five algorithms, listed below. Note that $\mathscr{T}$ denotes an access tree (AND-OR circuit) and γ represents a set of attributes.

$\textbf{Setup}() \rightarrow \mathsf{Params}, \mathsf{MasterKey}$

$\textbf{KeyGen}(\mathsf{Params}, \mathsf{MasterKey}, \mathscr{T}) \rightarrow \mathsf{Key}_{\mathscr{T}}$

$\textbf{Encrypt}(\mathsf{Params}, \gamma, m) \rightarrow \mathsf{Ciphertext}_{\gamma,m}$

$\textbf{QualifyKey}(\mathsf{Params}, \mathsf{Key}_{\mathscr{T}_\mathsf{A}}, \mathscr{T}_\mathsf{B}) \rightarrow \mathsf{Key}_{\mathscr{T}_\mathsf{B}}$

$$\textbf{Decrypt}(\mathsf{Key}_{\mathscr{T}}, \mathsf{Ciphertext}_{\gamma,m}) \rightarrow \begin{cases} m & \mathscr{T}(\gamma) = 1 \\ \perp & \text{otherwise} \end{cases}$$

For **QualifyKey** to return correctly, $\mathscr{T}_\mathsf{B}$ must be more restrictive than $\mathscr{T}_\mathsf{A}$: for every attribute set for which $\mathscr{T}_\mathsf{B}$ evalutes to 1, $\mathscr{T}_\mathsf{A}$ must also evaluate to 1.

Such Delegable Large-Universe KP-ABE cryptosystems do exist. For example, the GPSW construction [18, 19], based on Bilinear Maps, satisfies these properties.

3.4 Prefixes and Expiry with ABE

The authors of [19] note that Delegable Large-Universe KP-ABE, as they constructed, subsumes HIBE. Each component of an ID in HIBE can be concatenated with its index, and then hashed into the space of attributes. In this way, an ID in HIBE can be converted into an attribute set for encryption in KP-ABE. For private keys, access trees consist of a single AND gate checking if the attribute corresponding to each component of the ID is contained in the attribute set of the ciphertext.

We can use a similar approach to combine multiple hierarchies with collusion-resistance. We have two logical hierarchies: one for URI, and another for Time. To encrypt a message, a writer first obtains the ID for the URI and the ID for Time. For each component of the URI ID, the component is concatenated with its index in that ID, and then with the string "URI", and the result is hashed to obtain an attribute. A similar process happens for the Time ID. For example, to write to the resource `a/b` at the time `2017/Jan/07/08`, the message would be encrypted with the following attributes: `a-1-URI`, `b-2-URI`, `$-3-URI`, `2017-1-Time`, `Jan-2-Time`, `07-3-Time`, and `08-4-Time`. The private key for a URI prefix and Time prefix contains an access tree consisting of a single AND gate, which checks for the attributes for the correct components of the URI and Time IDs. In summary, Delegable Large-Universe Key-Policy Attribute-Based Encryption provides a way to combine multiple hierarchies in a collusion-resistant manner, allowing us to achieve Prefixes and Expiry simultaneously.

3.5 Ordered Attribute-Qualified Encryption

Although we were able to simultaneously achieve both Prefixes and Expiry using KP-ABE, that solution is not ideal because KP-ABE is computationally heavy. Real implementations of KP-ABE take several hundred milliseconds to encrypt a message with just tens of attributes—an order of magnitude longer than encrypting with HIBE.

In this section, we provide a way to simultaneously achieve both Prefixes and Expiry, with computational overhead comparable to HIBE. To do this, we develop Ordered Attribute-Qualified Encryption (OAQUE), which extends HIBE to support multiple collusion-resistant hierarchies without incurring the computational overhead of ABE.

Delegable Large-Universe KP-ABE is strictly more powerful than OAQUE. The novelty of OAQUE lies in the fact that it meaningfully extends the functionality of HIBE with no significant performance penalty.

Ordered Attribute-Qualified Encryption (OAQUE) supports encryption in a space of attributes. Unlike Attribute-Based Encryption schemes, attributes do not grant permissions. Instead, attributes *qualify* permissions. A key without any attributes can decrypt all messages; a key with some attributes can decrypt messages with those attributes (at the same positions) and possibly other attributes as well. Furthermore, the attributes are ordered; a key with an attribute at position 2 can decrypt all messages encrypted with that particular attribute at position 2. In summary, permissions are qualified by ordered attributes—hence the name, OAQUE.

3.5.1 Overview of OAQUE

An OAQUE cryptosystem is initialized with a fixed number ℓ of attribute slots. An attribute set is an assignment of attribute values to some of those slots.

Definition 1. *An attribute set S, in an OAQUE cryptosystem, is a mapping from slots to attribute values, where each slot is mapped to at most one attribute value. Formally,*

$$S = \{(i, a_i) \mid i \in \{1, 2, \ldots, \ell\}, a_i \in \mathbb{Z}_p^*\}$$

where no two elements of S have the same first component.

A slot is represented by an index in $\{1, 2, \ldots, \ell\}$, and an attribute value is represented by an element of $\mathbb{Z}_p^*$. In an attribute set S, a slot containing a value is referred to as *fixed*, and a slot without an attribute value is referred to as *free*. We emphasize that, even though we represent S as a set of (slot, attribute value) pairs, each slot may contain at most one value. For example, $S = \{(1,3), (1,5)\}$ is not a valid attribute set.

Definition 2. *Let S and T be attribute sets. We say "T is a subset of S" and write $T \subseteq S$ if the following conditions hold:*

1. *Every slot that is fixed in T is also fixed in S.*
2. *Every slot that is fixed in both T and S contains the same attribute value in T and S.*

We may also say "S is a superset of T" and write $S \supseteq T$, to mean the same thing.

Anyone with the public parameters of an OAQUE cryptosystem can encrypt a message m with an attribute set S. The resulting ciphertext can be decrypted using a private key for S. Furthermore, given a private key for an attribute set S_1, it is possible to generate a private key for S_2, where S_2 is a superset of S_1. Therefore, a ciphertext encrypted under S can be decrypted by any user with a private key for a subset of S.

The private key for the empty set can decrypt all messages, and is called the *master key*. Private keys for large attribute sets are less powerful than private keys for small attribute sets. Thus, attributes qualify permissions.

To summarize, OAQUE is a tuple of five algorithms:
Setup$(\ell) \rightarrow \mathsf{Params}, \mathsf{MasterKey}$
KeyGen$(\mathsf{Params}, \mathsf{MasterKey}, \mathsf{AS}) \rightarrow \mathsf{Key}_{\mathsf{AS}}$
QualifyKey$(\mathsf{Params}, \mathsf{Key}_{\mathsf{AS}_\mathsf{A}}, \mathsf{AS}_\mathsf{B}) \rightarrow \mathsf{Key}_{\mathsf{AS}_\mathsf{B}}$
Encrypt$(\mathsf{Params}, \mathsf{AS}, m) \rightarrow \mathsf{Ciphertext}_{\mathsf{AS},m}$
Decrypt$(\mathsf{Key}_{\mathsf{AS}}, \mathsf{Ciphertext}_{\mathsf{AS},m}) \rightarrow m$

ℓ is total number of slots supported in the cryptosystem. AS_A must be a subset of AS_B in **QualifyKey**.

3.6 Construction of OAQUE

Like the BBG construction of HIBE [6], our construction of OAQUE is based on bilinear groups, defined below:

Definition 3. *Let $\mathbb{G}$ and $\mathbb{G}_T$ be two (multiplicative) cyclic groups of prime order. $\mathbb{G}$ is* bilinear *if $\mathbb{G}$ and $\mathbb{G}_T$ are related by a map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ with the following properties:*

1. *Bilinearity. For all $u, v \in \mathbb{G}$ and $a, b \in \mathbb{Z}$, the following equality holds: $e(u^a, v^b) = e(u,v)^{ab}$.*
2. *Non-degeneracy. For all $g \in \mathbb{G}$ it is true that $e(g,g) \neq 1$.*

The map e with these properties is called a bilinear map.

We now present the construction of OAQUE. It is based on the BBG construction of HIBE [6]; the reader is encouraged to look at the two constructions side-by-side.

Setup(ℓ): Select $g, g_2, g_3, h_1, \ldots, h_\ell \xleftarrow{\$} \mathbb{G}$. Then select $\alpha \xleftarrow{\$} \mathbb{Z}_p$ and let $g_1 = g^\alpha$. $(g, g_1, g_2, g_3, h_1, \ldots, h_\ell)$ are the public parameters of the system. The master key is g_2^α.

KeyGen(g_2^α, S): Select $r \xleftarrow{\$} \mathbb{Z}_p$. The private key for the attribute set S is the following triple.

$$\left(g_2^\alpha \left(g_3 \cdot \prod_{(i,a) \in S} h_i^a \right)^r, g^r, \left\{ (j, h_j^r) \mid (j, \cdot) \notin S, 1 \leq j \leq \ell \right\} \right)$$

Regarding notation: $(j, \cdot) \notin S$ is true if slot j is free (i.e., there exists no element in S with j as its first component).

QualifyKey$((a_0, a_1, b), S)$: Select $t \xleftarrow{\$} \mathbb{Z}_p$. The private key for the attribute set S can be computed as follows:

$$\left(a_0 \cdot \left(g_3 \cdot \prod_{(i,a) \in S} h_i^a \right)^t \cdot \prod_{\substack{(i,a) \in S \\ (i,b_i) \in b}} b_i^a, \quad g^t \cdot a_1, \right.$$
$$\left. \left\{ (j, h_j^t \cdot b_j) \mid (j, \cdot) \notin S, 1 \leq j \leq \ell \right\} \right)$$

Encrypt(S, m): Here, $m \in \mathbb{G}_T$. Select $s \xleftarrow{\$} \mathbb{Z}_p$ and output

$$\left(e(g_1, g_2)^s \cdot m, \quad g^s, \quad \left(g_3 \cdot \prod_{(i,a) \in S} h_i^a \right)^s \right)$$

Decrypt$((a_0, a_1, b), (A, B, C))$: The first argument is the private key, and the second is the ciphertext. Output

$$A \cdot e(a_1, C) \cdot e(B, a_0)^{-1}$$

Our construction has a property called *partial delegation*, an analogue of the "limited delegation" property of BBG HIBE [6]. When delegating a private key (a_0, a_1, b), some elements of b may be omitted, preventing the recipient from filling the corresponding slots. This property will be useful in Section 4 when we construct Delegable Broadcast Encryption.

The proof of completeness and security of the above OAQUE construction is provided in Appendix A.

3.7 Prefixes and Expiry with OAQUE

If we require that the slots in OAQUE are filled in from left to right, then OAQUE degenerates to HIBE. This provides a method to combine multiple hierarchies. To combine a hierarchy of maximum depth ℓ_1 and a hierarchy of maximum depth ℓ_2, one can instantiate an OAQUE cryptosystem with the number of slots equal to $\ell_1 + \ell_2$. The first ℓ_1 slots are filled in left-to-right for the first hierarchy and the remaining ℓ_2 slots are filled in left-to-right for the second hierarchy.

We can use this technique to simultaneously achieve Prefixes and Expiry. For example, consider a maximum URI length of 16 and a maximum Time length of 4. Then the combination of `a/b/c/*` and `2017/Jan/*` can be expressed as the following OAQUE attribute set: $\{(1, \texttt{a}), (2, \texttt{b}), (3, \texttt{c}), (17, \texttt{2017}), (18, \texttt{Jan})\}$. OAQUE guarantees *collusion-resistance*: given a key for `a/*` and `2017/Jan/*` and another key for `a/b/*` and `2017/*`, one cannot generate a key for `a/*` and `2017/*`.

When used in this way, OAQUE actually allows for richer delegation on hierarchies than Prefix-Based Permissions. Prefix-Based Permissions allow us to use the `*` wildcard at the end of an ID; with OAQUE, we can also place a `+` wildcard in the middle of an ID, allowing a single component of the ID to remain unspecified. For example, the URI `a/+/b` matches all URIs of length 3 where the first component is `a` and the third component is `b`; the second component could be anything. To implement the `+` wildcard, we simply leave the slots corresponding to the `+` components free.

The `+` wildcard is extremely useful. In the "smart buildings" setting, one could imagine a resource hierarchy of the form `building/floor/room/sensor_id/reading_type`, where `reading_type` could be either "temperature" or "humidity". The `+` wildcard allows one to delegate permission to see only the temperature readings, by granting permission on the URI `+/+/+/+/temperature`. It is also useful for the Time hierarchy. An organization may want to give an employee access to a resource from 8 AM to 5 PM every day, which can be accomplished by using the `+` wildcard for the slots corresponding to the year, month, and day.

In short, OAQUE not only allows us to simultaneously achieve Prefix-Based Permissions and Expiry with low computational overhead, but also allows more powerful semantics beyond simple hierarchies.

3.8 Comparison to Other Approaches

One approach for expiry, used by some existing systems, is to periodically update keys. At each unit of time, the central authority generates a new set of keys, and new keys are distributed to authorized entities. An alternative approach to expiry is to rely on a trusted server to "forget" keys when they expiry. NuCypher [15] uses this approach; untrusted recipients rely on a set of trusted servers to hold key shares, and at least one of them is trusted to delete its share when the key expires.

Our approach is unique because (1) it enforces expiry in the cryptography itself, and (2) it is fully stateless. This allows our protocols to simultaneously provide stronger security guarantees and better scalability than existing approaches.

4 Revocation with Stateless Receivers

To implement revocation, we leverage existing work on Broadcast Encryption [13]. We explain the Public-Key Broadcast Encryption schemes in [13], and then extend them to support *delegation*. Finally, we explain how to combine Delegable Broadcast Encryption with the protocols we developed in Sections 2 and 3 to achieve Prefixes, Expiry, and Revocation simultaneously.

4.1 Public-Key Broadcast Encryption

Broadcast Encryption [25, 13] provide a way to encrypt a message such that it is visible to many users. However, a small subset of "revoked" users, known when the message is encrypted, cannot decrypt the message.

The techniques in [13] leverage the idea of *Subset Cover*. Let N be the (finite) set of all users. A family of subsets $\mathscr{S}$ is defined over N. Each each element S_i of $\mathscr{S}$ is a subset of N, and corresponds to a keypair (PK_{S_i}, SK_{S_i}). Each user is given the secret key SK_{S_i} for every set S_i containing that user.

Now, suppose that someone wants to encrypt a message such that it is visible only to a subset of users, denoted B. To achieve this, she must find a *subset cover* for B: some sets $B_1, \ldots, B_p \in \mathscr{S}$ such that $B = \bigcup_{j=1}^{p} B_i$. Then, she encrypts her message under the public key PK_{B_j} for each B_j in the subset cover. Only users in B have the secret key for one of the B_j to decrypt the message.

We will focus on two techniques discussed in [13], called Complete Subtree (CS) and Subset Difference (SD). CS and SD are two different constructions of the subset family $\mathscr{S}$, with different tradeoffs between (1) the number of ciphertexts required to send a message, and (2) the number of secret keys each user is given.

Note that each entity in our protocols may possess the keys for multiple users in Broadcast Encryption. To avoid confusion, we will henceforth use the term "decryption permission" instead of "user."

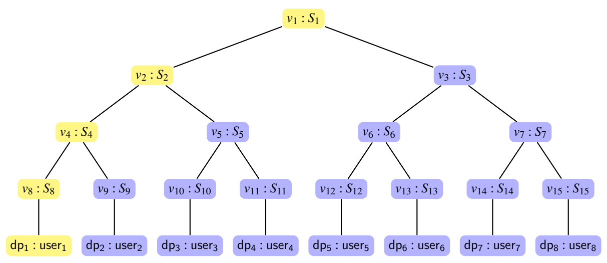
Both CS and SD organize decryption permissions into a binary tree, denoted as $\mathscr{T}$, where each decryption permission corresponds to a leaf (Figure 1). Each edge in $\mathscr{T}$ is labelled with 0 or 1 according to whether the edge connects to a left child or right child. This allows us to assign an identifier to each node v in the tree: $\mathsf{ID}(v)$ is the bitstring obtained reading all the labels in the path from root down to v.

Let N denote the set of all decryption permissions, and let $n = |N|$. Let R denote the set of revoked users, and let $r = |R|$. To achieve revocation using Broadcast Encryption, we let $B = N \setminus R$ be the set of users who can decrypt a message. Both CS and SD perform well when r is small.

4.2 Overview of Complete Subtree Method

The CS method defines the subset family $\mathscr{S}$ as follows: each subset corresponds to a *subtree* of the binary tree $\mathscr{T}$, and contains all of the decryption permissions corresponding to leaves in the subtree. We denote a node in $\mathscr{T}$ as v_i (Figure 1), and let S_i denote the subset corresponding to the subtree rooted at node v_i. Each node v_i in the tree is assigned a keypair $(\mathsf{PK}_i, \mathsf{SK}_i)$. The keypair of a subset $S_i \in \mathscr{S}$ is the keypair of the corresponding node v_i.

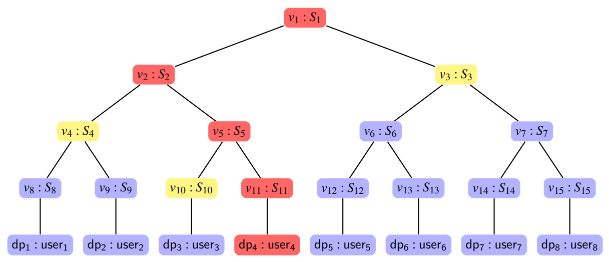
Key Assignment. Recall that each decryption permission has the secret key for each subset $S_i \in \mathscr{S}$ to which it belongs. In the CS method, these are simply the secret keys corresponding to each of its ancestors in the tree. For example, in Figure 1, the decryption permission dp_1 only needs to store the keys corresponding to yellow nodes. It is easy to see that each decryption permission has $O(\log n)$ secret keys.



$v_1 : S_1$
$v_2 : S_2$
$v_3 : S_3$
$v_4 : S_4$
$v_5 : S_5$
$v_6 : S_6$
$v_7 : S_7$
$v_8 : S_8$
$v_9 : S_9$
$v_{10} : S_{10}$
$v_{11} : S_{11}$
$v_{12} : S_{12}$
$v_{13} : S_{13}$
$v_{14} : S_{14}$
$v_{15} : S_{15}$
$\mathsf{dp}_1 : \mathsf{user}_1$
$\mathsf{dp}_2 : \mathsf{user}_2$
$\mathsf{dp}_3 : \mathsf{user}_3$
$\mathsf{dp}_4 : \mathsf{user}_4$
$\mathsf{dp}_5 : \mathsf{user}_5$
$\mathsf{dp}_6 : \mathsf{user}_6$
$\mathsf{dp}_7 : \mathsf{user}_7$
$\mathsf{dp}_8 : \mathsf{user}_8$

Figure 1

Encryption. Given the set R of revoked users, one must find a subset cover for $N \setminus R$. Figure 2 shows an example where $R = \{\mathsf{dp}_4\}$; the subset cover consists of S_3, S_4, and S_{10}, so the sender must encrypt his message under the public keys corresponding to v_3, v_4, and v_{10}. In general, the sender must encrypt his message $O\left(r \log\left(\frac{n}{r}\right)\right)$ times [25].



$v_1 : S_1$
$v_2 : S_2$
$v_3 : S_3$
$v_4 : S_4$
$v_5 : S_5$
$v_6 : S_6$
$v_7 : S_7$
$v_8 : S_8$
$v_9 : S_9$
$v_{10} : S_{10}$
$v_{11} : S_{11}$
$v_{12} : S_{12}$
$v_{13} : S_{13}$
$v_{14} : S_{14}$
$v_{15} : S_{15}$
dp_1 : user_1
dp_2 : user_2
dp_3 : user_3
dp_4 : user_4
dp_5 : user_5
dp_6 : user_6
dp_7 : user_7
dp_8 : user_8

Figure 2

Note that one public key must be made available for each node in $\mathscr{T}$. To lessen the overhead, [13] suggests using Identity-Based Encryption (IBE) [7]. The sender of a message can use the ID mapping described earlier to derive an IBE encryption key for each node whose subset is part of the subset cover.

4.3 Overview of Subset Difference Method

The SD method uses a different subset family $\mathscr{S}$, in which each decryption permission belongs to more subsets. This increases the number of secret keys that each decryption permission has, but allows the sender of a message to find a more efficient subset cover for $N \setminus R$, decreasing the number of times a message must be encrypted.

Key Assignment. In the SD method, each subset $S_{ij} \in \mathscr{S}$ is defined in terms of two nodes, v_i and v_j, where v_j is a descendant of v_i. S_{ij} contains all decryption permissions in the subtree rooted at v_i but not in the subtree rooted at v_j. Each decryption permission now belongs to $\sum_{k=1}^{\log(n)}(2^k - k)$ different subsets, and therefore has $O(n)$ secret keys.

Optimization. To reduce the number of secret keys each decryption permission has, [13] uses Hierarchical IBE. Observe that if a decryption permission is in S_{ij}, then it is also in S_{ik} where v_k is a descendant of v_j. Using HIBE, it possible to obtain the private key for S_{ik} from the private key for S_{ij}.

We associate each subset S_{ij} to an ID in HIBE as follows. The first component of the ID is $\mathsf{ID}(v_i)$. The remaining components of the ID are $\mathsf{ID}(v_j)$, where each bit occupies one component. This makes it possible to generate the private key for S_{ik} from the private key for S_{ij}, as long as v_k is a descendant of v_j.

If a decryption permission belongs to both S_{ij} and S_{ik}, it only needs to be given S_{ij}. With this optimization, each decryption permission only has $O(\log^2 n)$ HIBE secret keys [25].

Encryption. As in the CS method, the sender must find a subset cover for $N \setminus R$. The algorithm finding the optimal subset cover is introduced in [25]. It is also proved that only $O(r)$ subsets (and therefore $O(r)$ encryptions) are needed.

4.4 Delegable Broadcast Encryption

We extend the CS method and SD method to support both revocation and delegation for one resource. We do not consider Prefix-Based Permissions or Expiry, for now.

Suppose that an entity “Alice” owns a resource. She may delegate access to the resource to “Bob”, who may in turn delegate it to another entity “Charlie”, and so on. However, each delegation must be individually revocable. If Bob revokes his delegation to Charlie, both Alice and Bob should retain their ability to decrypt messages. If Alice revokes her delegation to Bob, then only Alice should retain her ability to decrypt messages.

Our approach is as follows. Alice owns the resource, and therefore has the master key. When she delegates access to Bob, she gives Bob the secret keys corresponding to *multiple* decryption permissions in $\mathscr{T}$. When Bob delegates access to Charlie he gives Charlie the secret keys corresponding to a *subset* of the decryption permissions given to him by Alice. To revoke her delegation to Bob, Alice would add the decryption permissions she gave Bob to R, a global set of revoked users. To revoke his delegation to Charlie, Bob would add the decryption permissions he gave Charlie to R. When writing to a resource, a writer would read the global set R, and use the Broadcast Encryption protocols to make sure those decryption permissions cannot decrypt his message. We call this construction *Delegable Broadcast Encryption*.

This approach works, but as written, it is not very efficient. When Alice delegates access to Bob, how many decryption permissions does she give him? She does not know in advance how many delegations Bob will make, so ideally she would provide many decryption permissions, say 10,000. However, this would greatly increase the number of secret keys that Alice must give Bob. Is there a way to *succinctly* delegate the secret keys for multiple decryption permissions?

Our insight is that Alice should give Bob secret keys for *consecutive* decryption permissions in the binary tree $\mathscr{T}$. Because k consecutive leaves in a tree can be coalesced into $O(\log k)$ subtrees, we can represent the secret keys for consecutive decryption permissions more succinctly. In Sections 4.5 and 4.6, we explore how to do this. As can be seen in Table 1, the savings due to our optimization are substantial when the number of delegated decryption permissions is large.

Without Our Optimization

	Private Key Storage	Number of Ciphertexts
CS method	$O(\log(n))$	$O(r\log(\frac{n}{r}))$
SD method	$O(\log^2(n))$	$O(r)$
Delegable CS method	$O(k+\log(n))$	$O(r\log(\frac{n}{r}))$
Delegable SD method	$O(k+\log^2(n))$	$O(r)$

With Our Optimization

	Private Key Storage	Number of Ciphertexts
Delegable CS method	$O(\log(k)+\log(n))$	$O(r\log(\frac{n}{r}))$
Delegable SD method	$O(\log(k)+\log^2(n))$	$O(r)$

Table 1: n is the total number of decryption permissions, k is the number of decryption permissions the private key contains, and r the number of revoked decryption permissions.

4.5 Delegable CS Method

In this section, we investigate how to succinctly represent secret keys for consecutive decryption permissions, in the CS method. First, we analyze the private key storage in the naïve construction of Delegable Broadcast Encryption. Then, we explain how to optimize the private key storage for consecutive decryption permissions.

Analysis of Private Key Storage. An entity with a set of k consecutive decryption permissions denoted DP must store the private keys for all subsets $S_j \in \mathscr{S}$ where $S_j \cap \mathsf{DP} \neq \varnothing$. There are $O(k)$ subsets where $S_j \subseteq \mathsf{DP}$, and about $O(\log n)$ subsets such that $S_j \nsubseteq \mathsf{DP}$ and $S_j \cap \mathsf{DP_i} \neq \varnothing$. Thus, the entity must store $O(k+\log n)$ private keys in all.

Optimization. We use HIBE for the key for each subset S_i. The ID corresponding to a node v_i is $\mathsf{ID}(v_i)$ (where each bit is a separate component of the ID) followed by \$, a termination symbol. The termination symbol makes it impossible to abuse the **QualifyKey** function in HIBE to derive the ID for v_j from the ID for v_i. However, if we omit the termination symbol \$, we can succinctly represent the private keys for all nodes in an entire subtree of $\mathscr{T}$, as a single HIBE private key.

Using this insight, we can represent the secret keys for k consecutive decryption permissions more efficiently. Recall that there are $O(k)$ subsets S_j where $S_j \subseteq \mathsf{DP}$. These subsets can be partitioned into $O(\log k)$ different complete subtrees in $\mathscr{T}$. Let $\mathsf{TR}(\mathsf{DP})$ be the set of root nodes of these subtrees. Figure 3 shows the case for $\mathsf{DP} = \{\mathsf{dp}_3, \mathsf{dp}_4, \mathsf{dp}_5, \mathsf{dp}_6\}$. The red nodes represents the subsets S_j satisfying $S_j \subseteq \mathsf{DP}$, and are partitioned into two complete subtrees rooted at v_5 and v_6.

The secret keys for the $O(k)$ subsets can be represented with just $O(\log k)$ HIBE secret keys: one HIBE secret key, without the terminator symbol \$, per element of $\mathsf{TR}(\mathsf{DP})$. This reduces the number of secret keys for k consecutive leaves from $O(k+\log n)$ to $O(\log k+\log n)$.

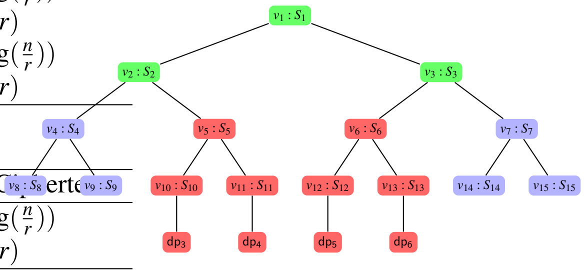


Figure 3

4.6 Delegable SD Method

We now investigate how to apply the optimizations to Delegable Broadcast Encryption that we made to the CS method, to the SD method. In particular, we optimize the private key storage for consecutive decryption permissions.

Analysis of Private Key Storage. An entity with a set of k consecutive decryption permissions denoted DP must store private keys for all subsets $S_{jk} \in \mathscr{S}$ where $S_{jk} \cap \mathsf{DP} \neq \varnothing$. But we can leverage HIBE as in the original SD Method (Section 4.3) to reduce the private key storage.

We define S_j as in the CS method: the set of decryption permissions in the subtree rooted at v_j. Define $\mathsf{copath}(v)$ as in a Merkle Tree [23]. Then define $\mathsf{CoPath}(\mathsf{DP})$ as follows:

$$\mathsf{CoPath}(\mathsf{DP}) = \mathsf{copath}(\mathsf{leaf}(\mathsf{dp_{min}})) \cup \mathsf{copath}(\mathsf{leaf}(\mathsf{dp_{max}}))$$

where $\mathsf{dp_{min}}$ stands for the first decryption permission in DP and $\mathsf{dp_{max}}$ stand for the last decryption permission in DP. $\mathsf{LeftChild}(v_i)$ and $\mathsf{RightChild}(v_i)$ stands for the left child node and the right child node of v_i respectively.

Then an entity with DP only needs to store the private keys for subsets S_{jk} satisfying the following properties:

1. $S_{jk} \cap \mathsf{DP} \neq \varnothing$
2. If $S_{\mathsf{LeftChild}(v_j)} \cap \mathsf{DP} \neq \varnothing$ and $S_{\mathsf{RightChild}(v_j)} \cap \mathsf{DP} \neq \varnothing$, then v_k must satisfy $v_k = v_j$.
3. If $S_{\mathsf{LeftChild}(v_j)} \cap \mathsf{DP} = \varnothing$ or $S_{\mathsf{RightChild}(v_j)} \cap \mathsf{DP_i} = \varnothing$, then v_k must satisfy $v_k \in \mathsf{CoPath}(\mathsf{DP})$, or the parent of v_k is in $\mathsf{TR}(\mathsf{DP})$.

There will be $O(k+\log(n))$ subsets S_{jk} satisfying the second conditions, and $O(\log^2 n)$ subsets S_{jk} satisfying the third condition, thus the total key storage will be $O(k+\log^2 n)$.

Optimization. We attempt to use a similar idea used in Delegable CS method for optimization. In particular, we will represent the $O(k)$ subsets satisfying the second condition using only $O(\log k)$ keys (Note that there are $O(\log(n))$ subsets satisfying the second condition which cannot be optimized). We will do this by introducing a hierarchy for the first index (the i in S_{ij}). However, we are already using a hierarchy for the second index (the j in S_{ij}), and HIBE cannot support two hierarchies simultaneously. Therefore, we will use OAQUE to combine the two hierarchies, as we did in Section 3.7.

We use an OAQUE system with $2\log(n)+1$ slots. We use the first $\log(n)+1$ slots to support a hierarchy for the first index, and the remaining $\log n$ slots to support a hierarchy for the second index. Each subset S_{ij} is associated to an Attribute Set in OAQUE as follows. The first hierarchy encodes $\mathsf{ID}(v_i)$ (each bit in a separate slot), followed by \$, a termination symbol. The second hierarchy (with $\log n$ slots) encodes $\mathsf{ID}(v_j)$, where each bit is stored in a separate slot.

With this construction, we preserve all of the functionality from before. The second hierarchy representing the second index can be extended, just as in Section 4.3. As before, the first index cannot be modified, because the terminator symbol \$ prevents the first hierarchy from being extended.

Now, we can represent the secret keys for the $O(k)$ subsets satisfying the second condition above, more efficiently. Observe that, as in the Delegable CS Method, the k consecutive leaves can be grouped into $O(\log k)$ consecutive subtrees. Let $\mathsf{TR}(\mathsf{DP})$ be the set containing the root nodes of these subtrees. For each node $v_i \in \mathsf{TR}(\mathsf{DP})$, we must provide the private key corresponding to S_{ii}, without including the terminator symbol \$ in the attribute set. From these $\log k$ OAQUE keys, an entity can obtain all $O(k)$ keys corresponding to subsets S_{ij} satisfying the second condition above.

This reduces the total number of secret keys for k consecutive leaves from $O(k+\log^2 n)$ to $O(\log k+\log^2 n)$.

4.7 Construction of ROAQUE

Integrating Delegable Broadcast Encryption with our earlier protocol for Prefixes and Expiry is straightforward: we use the same technique as we used in Section 3.7. We can divide up the ℓ OAQUE slots into three parts: $\ell = \ell_1 + \ell_2 + \ell_3$. We use the first ℓ_1 slots for Delegable Broadcast Encryption, the second ℓ_2 slots for the resource hierarchy, and the third ℓ_3 slots for the time hierarchy for Expiry.

Delegation and encryption are done exactly as in Delegable Broadcast Encryption, except that the URI and Time are also encoded into the last $\ell_2+\ell_3$ slots. In general, the remaining slots not used for Delegable Broadcast Encryption are like a new OAQUE scheme, but inherit the revocation properties. Therefore, we call this construction "Revocable OAQUE."

Although we used \$ as a terminator symbol in Sections 4.5 and 4.6 to prevent users from extending certain hierarchies, we can also use the *partial delegation* property of OAQUE (Section 3.6) for this purpose. This would also reduce the size of private keys. We use it in our implementation.

4.8 Comparison to Expiry

Simultaneously supporting Expiry and Revocation is useful for two reasons. First, revoking a delegation requires the granter of that delegation to come online to publish a signed message, informing writers of the revocation. Second, revoking a delegation has a cost: the amount of work done by publishers is $O(r)$, where r is the number of revoked delegations. In contrast, expiration is fully stateless: delegations expire "silently" without requiring additional work by any party. In cases where the $O(r)$ cost of publishing is unacceptable, Expiry can be used as a primitive means of revocation.

5 Key Inheritance

Section 5.1 describes Key Inheritance and explains why it is useful. Section 5.2 explains how to achieve Key Inheritance with OAQUE. Section 5.3 explains how to use Key Inheritance with the techniques for Prefixes, Expiry and Revocation that we developed.

5.1 Key Inheritance as a Primitive

Suppose Alice works for a company with a distributed filesystem accessible to its employees. The filesystem has two top-level directories, `work` and `secrets`. Alice has permission to access some files in both directories (i.e., she has permission on `work/file1`, `work/file2`, `secrets/file1`, etc.). Alice has a secretary who should be able to see all files accessible to Alice in the `work` directory, but none of the files in the `secrets` directory.

In the current setup, Alice must give each key delegated to her in the `work` directory, to her secretary. This is not succinct, as there may be many such keys. If Alice later receives access to another file in the `work` directory, she must come online to forward the key to her secretary.

Key Inheritance provides a way to alleviate this problem. Alice can delegate the ability to inherit keys on `work/*` to her secretary. This allows her secretary to gain access to keys delegated to Alice for files in the `work` directory, but not in any other directories. We call this type of delegation an *inheritance delegation*. Second, when Bob delegates permission to Alice on `work/somefile`, he can delegate the key such that it is visible to other entities to whom Alice has given inheritance delegations.

We call this an *inheritable delegation*. Bob may choose not to make his delegation inheritable, in which case his key will not be accessible to those other entities.

Note that there are no temporal restrictions. Bob's inheritable delegation to Alice can be made either before or after Alice's inheritance delegation to her secretary.

5.2 Key Inheritance with OAQUE

We use OAQUE to implement Key Inheritance. Each entity has its own OAQUE system. We assume that the public parameters of each entity's OAQUE system are publicly known, but the master key is a secret known only to that entity. We refer to secret keys in an entity's cryptosystem as *inheritance keys*, and we refer to secret keys used to decrypt resources as *decryption keys*.

- To make an inheritance delegation for an attribute set, Alice generates the secret key for that attribute set in her own OAQUE system, and gives it to the recipient.

- To give Alice a decryption key for an attribute set via an inheritable delegation, Bob takes the decryption key, encrypts it under the same attribute set in Alice's OAQUE system, and publishes the ciphertext globally.

Let us return to the original example. Via an inheritance delegation, Alice's secretary has the secret key for `work/*` in Alice's OAQUE system. Bob makes an inheritable delegation to Alice, taking the decryption key for `work/somefile` encrypting it under `work/somefile` in Alice's system, and publishing the ciphertext. So Alice's secretary can decrypt the decryption key for `work/somefile`, thereby inheriting it.

Our Key Inheritance scheme is *chainable*: inheritance delegations can also be made inheritable. Bob can make an inheritable inheritance delegation to Alice by generating the private key for the attribute set in his own OAQUE system, encrypting that private key under the same attribute set in Alice's OAQUE system, and publishing the resulting ciphertext globally.

Our method is reminiscent of Protected Declarations of Trust (PDOTs) in WAVE [2]. Unlike PDOTs, which hides the delegations themselves, Key Inheritance provides a flexible means to delegate encryption keys.

5.3 Key Inheritance and Revocation

Using our core protocols (Sections 2 and 3) with Key Inheritance is straightforward—simply use the Attribute Set that expresses Prefix-Based Permissions and Expiry for Key Inheritance. This gives inheritance delegations expiry times as well—only keys that expire before the the time of the inheritance delegation can be inherited.

To use Key Inheritance with Revocation (Section 4), note that the slots used for Delegable Broadcast Encryption simply tag each key with a token used for revocation, and do not contribute to the actual permissions being conveyed. A simple approach is to not include the slots used for Delegable Broadcast Encryption when granting inheritance keys, or when encrypting keys in an entity's OAQUE system. In this approach, decryption keys are still individually revocable, but inheritance keys are not.

To make inheritance keys revocable, we can apply Broadcast Encryption to inheritance keys. Each entity includes extra slots in its OAQUE system for Broadcast Encryption, and associates each inheritance delegation with a decryption permission. When Bob makes an inheritable delegation to Alice, Bob finds which decryption permissions Alice has revoked, and encrypts his key in Alice's system using Broadcast Encryption such that it is not decryptable by those keys.

Even this method is not perfect when Key Inheritance is used in a chaining configuration. If Bob grants Alice an inheritable inheritance delegation on `work/directory1/*`, and Alice grants her secretary an inheritance delegation on `work/*`, Alice's secretary has inherited Bob's inheritance delegation on `work/directory1/*`. Therefore, when another entity grants Bob an inheritable delegation on `work/directory1/somepath`, Alice's secretary will inherit that key, even if Alice had revoked the inheritance delegation given to her secretary. Nevertheless, we believe that this method of revocation is suitable for most uses of Key Inheritance.

6 Implementation

We implemented our protocols in the Go programming language. The BBG HIBE construction [6], GPSW KP-ABE construction [18], and our OAQUE construction, can be instantiated with *asymmetric* bilinear groups.

Definition 4. *Let $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ be cyclic groups of prime order. $\mathbb{G}_1$ and $\mathbb{G}_2$ are* asymmetric bilinear groups *if $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ are related by a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$.*

Specifically, we use Barreto-Naehrig elliptic curves with 128 bits of security, which provide a pair of asymmetric bilinear groups. They are believed to be suitable for cryptographic schemes that rely on the Bilinear Diffie-Hellman Assumption [11, 22]. We use the assembly-optimized implementation described in [24], with a lightweight shim provided by the authors of Vuvuzela [28] that allows the code to be called from Go programs. Some of our benchmarks use a Raspberry Pi; for these benchmarks, we use a portable, C-based implementation of the elliptic curves. Although slower than the assembly-optimized implementation, the C-based implementation is faster than the stock Go-based version.

We also provide libraries for Prefixes, Expiry, Revocation, and Key Inheritance, implemented on top of OAQUE. We use these libraries in Section 8 to augment existing systems with end-to-end encryption. Our code is open-source and available on GitHub.

7 Microbenchmarks

In this section, we evaluate our protocols in the context of their performance.

7.1 OAQUE, HIBE, and KP-ABE

OAQUE's novelty lies in the fact that it is substantially more efficient than Delegable Large-Universe KP-ABE. In this section, we use performance benchmarks to verify that OAQUE is indeed more performant than KP-ABE.

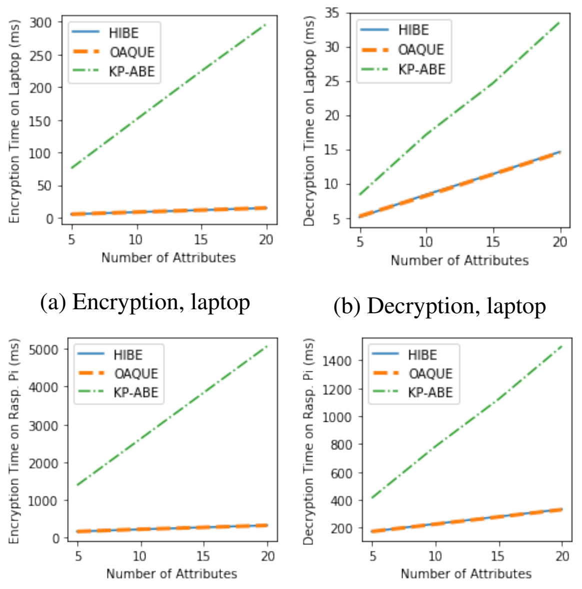
We implemented the GPSW Delegable Large-Universe KP-ABE construction [18, 19] and the BBG HIBE construction [6]. We use the same implementation of Barreto Naehrig curves as we used for OAQUE, so that the comparison is fair.

Benchmarks labeled "Laptop" were produced on a Lenovo T470p laptop with an Intel Core i7-7820HQ CPU @ 2.90 GHz, using the amd64-optimized bn256 library. Performance benchmarks labeled "Raspberry Pi" were produced on a Raspberry Pi with an ARM v7 processor, using the portable bn256 library implemented in C. We focus on measuring the performance of **Encrypt** and **Decrypt** because they are used on the data plane. In contrast, the performance of **Setup**, **KeyGen**, and **QualifyKey** is less critical.

The results are shown in Figure 4. To measure encryption time with x attributes, we measure the time to encrypt a message with an ID of length x in HIBE, x slots filled in OAQUE, or x attributes set in KP-ABE.

To measure decryption time, we first encrypt a message with x attributes. We measure the time taken for KP-ABE to decrypt the ciphertext with a key whose access tree consists of a single AND gate with x inputs. For OAQUE and HIBE, we measure both the time taken to decrypt the ciphertext, and the time to generate a decryption key for that attibute set or ID. For example, if I have the key for `a/*`, and I receive a message on `a/b/c/d/e/f`, I have to first generate the key for `a/b/c/d/e/f` before I can decrypt. Note that this represents an upper bound; generating the key for `a/b/c/d/e/f` is faster if you are starting from `a/b/*` than it is if you are starting from `a/*`.

Figure 4, shows that OAQUE is 15-20x faster than KP-ABE for encryption, and 2-5x faster for decryption, including the time to generate subkeys. This confirms that OAQUE is substantially more efficient than KP-ABE. Furthermore, there is essentially no performance gap between OAQUE and HIBE, even though OAQUE extends HIBE's functionality in a meaningful way.



(a) Encryption, laptop (b) Decryption, laptop

(c) Encryption, Raspberry Pi (d) Decryption, Raspberry Pi

Figure 4: Performance of OAQUE, HIBE, and KP-ABE

Operation	Scheme	Pairings	Exponentiations
Encrypt	HIBE	1	$3+r$
Decrypt	HIBE	2	r (to generate key)
Encrypt	OAQUE	1	$3+r$
Decrypt	OAQUE	2	r (to generate key)
Encrypt	KP-ABE	1	$2+r+r\cdot\ell$
Decrypt	KP-ABE	$r+1$	$2r$

Table 2: Pairings and exponentiations required for HIBE, OAQUE, and KP-ABE. The single pairing required for encryption can be precomputed in all three schemes.

For the implementation of Barreto-Naehrig curves that we used, the two most expensive operations are exponentiation and pairing. We also compare the performance of KP-ABE, OAQUE, and HIBE in terms of the number of exponentiations and pairings. This is shown in Table 2. r is the number of attributes used, and ℓ is the total number of attributes that can be used for a single message (the implicit argument to **Setup**). This shows that OAQUE's performance is theoretically better than KP-ABE's performance, and explains the measurements in Figure 4.

8 Applications to Real Systems

In this section, we implement our protocols and apply them to three real systems. We demonstrate that our protocols not only are useful in system design, but also can be easily bootstrapped onto existing systems in order to provide stronger security guarantees.

8.1 WAVE: IoT Syndication/Authorization

WAVE [1, 2] is a blockchain-based system for syndication and authorization in the Internet of Things. We first describe WAVE, and then improve it using our protocols.

8.1.1 The WAVE System

A participant in WAVE is called an *entity*. Each entity owns a hierarchy of resources, and each resource in the hierarchy is represented by a URI. The hierarchy is called a *namespace* and the entity that owns the hierarchy is the *namespace authority* for those resources.

Access to resources is granted via Delegations of Trust (DOTs). A DOT is a cryptographically signed statement produced by an entity that grants another entity access to a set of resources. For example, if Alice wants to grant Bob access to `a/b/c/*` in Charlie's namespace, she produces a statement, "I, Alice, give Bob permission to access `a/b/c/*` in Charlie's namespace," signs it with her secret key, and makes the statement publicly available. Note that Alice can produce this statement even if she does not have permission to access `a/b/c/*`. Bob only has permission to access a resource if there is a *chain* of DOTs from the namespace authority (Charlie, in this case) to Bob, and if *all* DOTs in the chain grant access to the resource. To make all DOTs and Entities publicly available without relying on any central trusted party, WAVE uses blockchain smart contracts to maintain a decentralized registry of DOTs and Entities.

WAVE's syndication model is based on Publish-and-Subscribe messaging. To publish to a resource, an entity encrypts its message under the public key of the namespace authority, signs the ciphertext with its own secret key, and sends the message to the designated router (e.g., broker) for that resource hierarchy, chosen by the namespace authority. It also includes a DOT chain, to prove that it has permission to publish to that resource. The router verifies the DOT chain and the signature. Then it decrypts the message, re-encrypts it under the public key of each subscribed entity, and forwards each ciphertext to the corresponding subscriber. To subscribe to a resource, an entity connects to the designated router and provides a DOT chain to prove that it has permission to subscribe. Then the router begins forwarding messages to it.

8.1.2 From WAVE to STARWAVE

While WAVE provides a very flexible authorization system, it relies on a *trusted* designated router to enforce the permissions on each namespace. In particular, the designated router sees messages in plaintext, because it is responsible for decrypting messages and then re-encrypting them for delivery to the message's recipients. An attacker who compromises a designated router can see all of the messages published on that namespace.

Encrypting messages end-to-end would solve this problem, but doing so is nontrivial. The publisher cannot simply encrypt the message under the public key of each consumer, because the publisher cannot be expected to keep track of subscribers in the IoT setting (see Challenge #1 in Section 1).

The protocols that we have developed in this paper allow us to scalably achieve end-to-end encryption in WAVE. DOTs that delegate permission to subscribe to a URI or set of URIs contain OAQUE keys that are used to decrypt messages for those URIs. DOTs that delegate permission to publish to a URI remain unchanged—the difference is that the *subscribers* verify the DOT chains instead of the designated router.

Our protocols augment WAVE by guaranteeing confidentiality of messages, even if the designated router is compromised by an adversary. We call the resulting system STARWAVE because it provides **S**ecure **T**ransport with an **A**dversarial **R**outer, in the context of **WAVE**.

8.1.3 Features of STARWAVE

Our core protocols, developed in Sections 2 and 3 allow us to simultaneously support the URI hierarchy of WAVE and expiry times on each DOT.

However, we decided not to incorporate our work on revocation in Section 4 into STARWAVE. The reason is that, in the protocol presented in Section 4, publishers must be aware of which delegations are revoked in order to properly encrypt messages. In the context of IoT, it is crucial that publishers do not have to keep track of such state, for scalability reasons.

DOTs in WAVE may be granted out-of-order; the only requirement is that a complete chain exists at the time of publication or subscription. Achieving fully out-of-order delegation with end-to-end encryption keys is difficult, and we leave it as an open problem. However, Key Inheritance (Section 5) allows us to support out-of-order delegation in some cases. In particular, the "chaining" property of Key Inheritance allows us to support out-of-order DOT chains, as long as the permissions delegated at each DOT are a superset of the permissions delegated by the previous DOT in the chain.

8.1.4 Implementation of STARWAVE

First, we implement Prefix-Based Permissions, Expiry, and Key Inheritance on top of OAQUE in a system-level library. This code can be used by any system, and is not specific to WAVE.

Then, we implemented the WAVE-specific portion as a wrapper around the WAVE client library. It transparently includes additional public information, required for STARWAVE, in the "Contact" and "Comment" fields of DOTs and Entities. When subscribing to a URI, the wrapper transparently derives the OAQUE private key from the comments in the DOT chain and decrypts the messages, and delivers the plaintext to the application. Similarly, when the application publishes to a URI, the wrapper library transparently encrypts the message before sending it to the WAVE network.

The WAVE-specific wrapper is less than 1,000 lines of Go code. Our implementation required absolutely no changes to code in WAVE's client library, router, blockchain, or core—the wrapper is a separate module. Most importantly, the wrapper library provides exactly the same API as the standard WAVE client library. In fact, it can be used as a drop-in replacement for the standard WAVE client library, to easily port existing WAVE applications to use STARWAVE.

8.2 IPFS: Peer-to-Peer Distributed Storage

The InterPlanetary File System (IPFS) [3] is a peer-to-peer distributed filesystem. We first explain the parts of IPFS relevant to our work, and then use our protocols to implement end-to-end encryption in IPFS.

8.2.1 The InterPlanetary File System

At the core of IPFS is a content-addressed object store that allows any IPFS object to be looked up by its hash. Each object stores an arbitrary blob of bytes—in this way, it can act as a file. Each object also stores a set of *links*, which are mappings from names to hashes of other objects—in this way, it can also act as a directory. Given the hash of an object, one can retrieve a file by its filepath relative to that object by following links. For example, given a hash h_0 and a filepath `a/b/c`, one can look up the object r_0 corresponding to h_0, obtain the hash h_1 mapped to link `a` in r_0, look up the object r_1 corresponding to h_1, obtain the hash mapped to link `b` in r_1, and so on. Because all objects are referenced by their hash, the objects in IPFS collectively form a Merkle DAG.

One question remains: how is the root of the filesystem managed? A user can sign an object with her private key to make that object the root of her filesystem, accessible at `/ipns/<hash of user's public key>`. When she updates one of the files in her filesystem, the new hashes propagate to the filesystem root; the user then signs the new root to update the filesystem.

8.2.2 End-to-End Encryption in IPFS

We associate each user's filesystem in IPFS with an OAQUE cryptosystem. Each user possesses the OAQUE master key for her filesystem at `ipns/<hash of user's public key>`, and therefore, possesses the OAQUE master key for that system. URIs in our protocols correspond to filepaths in the user's filesystem. Using the protocols we developed in Sections 2, 3, 4, and 5, we can achieve decentralized distribution of keys with rich usage semantics (Prefix-Based Permissions, Expiry, Revocation) and flexible delegation (Key Inheritance).

For Key Inheritance, users must be able to see the *encrypted* keys given to other users. We need publicly accessible infrastructure to store these encrypted keys. This infrastructure must be decentralized in the same vein as IPFS, in order to preserve the advantages of using IPFS in the first place. Our solution is to reuse the mechanism for key delegation in STARWAVE (Section 8.1)—keys are delegated to users on WAVE's blockchain and stored in WAVE's decentralized registry. Each user in IPFS is tied to an entity in WAVE, via WAVE's aliasing system. URIs in an entity's namespace correspond to filepaths in corresponding user's IPFS filesystem.

When a file or directory in IPFS is changed, the hash of the root of the filesystem is changed and must be signed by the owner. Therefore, only the owner of the filesystem can mutate any files or directories. We emphasize that this is a restriction of the IPFS itself, not of our protocols.

8.2.3 Implementation

We implemented a utility in Go that allows users to interact with IPFS with end-to-end encryption. Our utility supports five commands—`mkfile`, `mkdir`, `ls`, `cat`, and `register-fs`. It transparently builds DOT chains in STARWAVE and decrypts files when read, and transparently encrypts files appropriately when written. Our implementation is less than 300 lines of code, demonstrating that the mechanisms for key delegation that we implemented in STARWAVE are general enough to be reused in other systems.

8.3 Dropbox: Cloud-Based File Sharing

Dropbox [14] is a file hosting service that offers cloud storage, file synchronization, personal cloud, and client software. We summarize file sharing in Dropbox, and then explain how to use our encryption scheme in such a cloud storage system.

8.3.1 File Sharing in Dropbox

In Dropbox, users store their files in the cloud and can share them with other users. If a user shares her files with another user, that user can in turn share this file with others. When users modify the shared file, the server stores both the original version and the modified version, so users can access the history of the shared file.

Our goal is to achieve an end-to-end encrypted cloud-based file sharing system with the following properties:

- A user can share a folder with another user, granting access to all files and subfolders inside the folder.

- A user can grant access to files for only a specified time range. The recipient cannot decrypt versions of a shared file outside of the time range.

- A user who is granted access to a file can grant other users access. The user may also restrict access when doing this; a user who is granted access to an entire directory may grant another user access to only certain files in that directory.

- A user should be able to revoke permission that she previously gave to other users. If a user's permission is revoked, all of the permissions which this user further delegated to others are also revoked.

8.3.2 End-to-End Encryption in Dropbox

We can use the "Revocable OAQUE" (ROAQUE) construction introduced in Section 4.7 to encrypt all files stored in the cloud, and achieve all of the properties listed above: Prefix-Based Permissions, Expiry, and Revocation. Each user of Dropbox sets up a ROAQUE cryptosystem, used to encrypt his files. Files are encrypted using the filepath and current timestamp. When the user shares one of his files (or directories) with Alice, he can specify the expiry time of the permission, and how many "decryption permissions" Alice can further delegate. Then the owner uses this information to generate the corresponding ROAQUE private key (which may consist of multiple OAQUE private keys). The owner can send the private key to Alice via email, and at the same time share the encrypted file with Alice through Dropbox. By the properties of ROAQUE, Alice can access the versions of all the file under the specified filepath with timestamp within the specified time range. Furthermore, she can delegate a subset of these permissions to other users.

For revocation, ROAQUE supports the property that if a private key is revoked, all private keys generated from that private key are also revoked. We need to maintain a set *R* of revoked decryption permissions that is visible to future writers. This can be in the form of another file in Dropbox that is shared with everyone. To revoke a key, a user simply adds the decryption permissions that she wishes to revoke, to *R*.

9 Related Work

We compare our protocols to related work, tracking developments in end-to-end encryption over time.

9.1 Traditional Public-Key Encryption

Among the earliest scenarios where Challenge #1 (Section 1) holds is file sharing on untrusted storage. SiRiUS [17] addresses this with a similar insight to us: keys are associated with files, rather than users. In SiRiUS, each user needs to keep track of two keys: a master signing key (MSK) and a master encryption key (MEK). The symmetric key for each file is stored on the untrusted server, encrypted under the MEK of each user authorized to access the file and signed by the file's owner. While this approach is simple, it is also limited: permissions are centralized (see Challenge #3). Only the owner of a file can grant access to the file, and that permission cannot in turn be delegated by the recipient.

In Plutus [21] users are in possession of the encryption keys for files they can access, meaning that permissions are delegable. Because managing a separate key for each file is not practical, Plutus allows files to be aggregated into *filegroups*, each managed by a single key. However, the key management overhead is proportional to the number of filegroups, making Plutus unsuitable for fine-grained access control.

9.2 Attribute-Based Encryption

One line of work uses Attribute-Based Encryption (ABE) [27] to delegate permission. For example, [31] uses Key-Policy ABE (KP-ABE) [18] to control which entities have access to encrypted data in the cloud. [31] also provides a means to revoke users by re-encrypting ciphertexts, leveraging Proxy Re-Encryption to safely perform the re-encryption in the cloud. The semantics of revocation in [31] are stronger than the semantics of revocation in our work. Whereas [31] hides existing data from revoked users, our revocation protocol (Section 4) only guarantees that revoked users cannot see new data, similar to the threat model in [21]. We believe that our approach is acceptable, because the user could have downloaded existing data before being revoked. Our work also supports hierarchically organized resources and decentralized delegation of keys, which [31] does not address.

Other approaches prefer Ciphertext-Policy ABE (CP-ABE) [4] because it is conceptually closer to Role-Based

Access Control than KP-ABE. Existing work [29, 30] combines HIBE with CP-ABE to produce Hierarchical ABE (HABE), a solution for sharing data on untrusted cloud servers. However, the "hierarchical" nature of HABE corresponds to the hierarchical organization of domain managers in an enterprise, rather than to a resource hierarchy as in our work.

9.3 Proxy Re-Encryption

Proxy Re-Encryption (PRE) [5] is a cryptographic primitive that, in addition to standard public-key encryption, supports re-encryption keys. A re-encryption key $\mathsf{RK}_{\mathsf{PK}_1 \rightarrow \mathsf{PK}_2}$ supports the following functionality, without leaking the plaintext message: a ciphertext of a message encrypted under PK_1 can be converted into a ciphertext of the same message encrypted under PK_2. Recent work leverages proxy re-encryption to support delegation of access to resources.

NuCypher KMS [15] allows a user to store data in the cloud encrypted under her public key, and share it with another user (the *recipient*) by providing a re-encryption key to a cloud server, which performs the re-encryption at the recipient's request. The cloud server is trusted to enforce conditional or temporal access policies. Therefore, NuCypher employs various techniques to discourage cloud servers from misbehaving: pseudoanonymity (identities hidden to prevent collusion), split-key secret sharing (m of n servers must cooperate to re-encrypt), and challenge protocols (to detect misbehavior).

While NuCypher assumes limited collusion among cloud servers and recipients (e.g., m of n secret sharing) to achieve properties such as Expiry, our protocols enforce Expiry and Revocation via cryptography, and therefore remain 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 (Challenge #4 in Section 1). Finally, keys in NuCypher are not qualifiable; given a key for `a/*`, one cannot generate a key for `a/b/*` to give to another entity.

Another system that uses PRE is PICADOR [8], a publish-and subscribe system which leverages the work in [26]. PICADOR uses proxy re-encryption to let the broker (the equivalent of a designated router in WAVE) to re-encrypt the ciphertexts from the writer's key to the readers' keys, while hiding the plaintext messages from the broker. A trusted server called the Policy Authority provides the broker with the re-encryption keys. One may draw similarities between the namespace authority in STARWAVE and Policy Authority in PICADOR: both are trusted and hold critical secrets. However, in STARWAVE, delegation of permissions is decentralized; the namespace authority grants some DOTs, but does not handle every single delegation. Meanwhile, the Policy Authority in PICADOR must create re-encryption keys from every authorized publisher to every authorized subscriber, requiring it to be online more often.

10 Future Work

Anonymous messaging systems, such as Riposte [12] and Talek [10], hide resource access patterns to resist traffic analysis, but assume that users have pre-shared keys. A natural extension of our work would be to devise a mechanism to delegate OAQUE keys without leaking access patterns—who is delegating a key to whom, on what resources in the hierarchy. This would allow anonymous messaging systems to use our work to distribute keys to bootstrap secure communication.

WAVE uses the powerful Delegation of Trust authorization model. To access a resource, an entity must present a chain of DOTs from the authority to that entity; the individual DOTs may have been granted in any order. An extension of our work would be to support fully out-of-order delegations like WAVE. A delegation would be a cryptographic object, such that a chain of delegations from the authority to an entity can be used by that entity to obtain an OAQUE private key.

11 Conclusion

In this paper, we presented a set of protocols for end-to-end encryption and for distributing keys for end-to-end encryption. Our protocols model entities as interacting with resources—in particular, writers need not be aware of readers—so our protocols are applicable to systems with persistent resources (e.g., IPFS, Dropbox) and to IoT-scale messaging systems (e.g., WAVE). Furthermore, our keys support prefix-based permissions on hierarchically organized resources, which are common in real systems. We incorporate expiry and revocation directly into the cryptography, providing stronger security guarantees than systems-level solutions employed by existing work (e.g., NuCypher [15]). Finally, we provide protocols for qualifying and delegating our keys, making our protocols relevant to an emerging class of decentralized systems, including blockchain-based systems like WAVE [2]. Together, these properties make our work applicable to real systems that people rely on to communicate and share information and that will be needed to support the Internet of Things.

A Appendix: Correctness of OAQUE

In this appendix, we prove the completeness and security of the construction of OAQUE in Section 3.6.

A.1 Completeness of OAQUE

There are two things to prove:

1. Generation of a more restrictive private key from a private key is equivalent to generating it from the master key.
2. Decryption of a ciphertext using the private key for its attribute set recovers the original message.

First, we prove the first statement.

Proof. Let (a_0, a_1, b) be a well-formed private key for an attribute set S^*, generated with $r \xleftarrow{\$} \mathbb{Z}_p$, and let $S \supseteq S^*$.

$$a_0 \cdot \prod_{\substack{(i,a)\in S \\ (i,b_i)\in b}} b_i^a = g_2^\alpha \cdot \left(g_3 \cdot \prod_{(i,a)\in S^*} h_i^a\right)^r \cdot \prod_{\substack{(i,a)\in S \\ (i,b_i)\in b}} b_i^a$$
$$= g_2^\alpha \cdot \left(g_3 \cdot \prod_{(i,a)\in S} h_i^a\right)^r$$

The resulting term is equivalent to the first term of the private key generated by a call to **KeyGen** with S directly. The additional terms in the **QualifyKey** function serve to re-randomize the key, changing the exponents from r to $r+t$. $\square$

Now we prove the second statement.

Proof. For a well-formed private key (a_0, a_1, b) and ciphertext (A, B, C) for an attribute set S, we have

$$\frac{A \cdot e(a_1, C)}{e(B, a_0)} = \frac{e(g, g_2)^{s\alpha} \cdot m \cdot e\left(g, g_3 \cdot \prod_{(i,a)\in S} h_i^a\right)^{rs}}{e\left(g, g_2\right)^{s\alpha} \cdot e\left(g, g_3 \cdot \prod_{(i,a)\in S} h_i^a\right)^{rs}}$$
$$= m$$

as desired. $\square$

A.2 Security of OAQUE

Our construction of OAQUE is based on the construction of HIBE in [6]. The authors of [6] explain the hardness assumptions they rely on, state the definition of IND-sID-CPA security for their scheme, and then prove that it fulfills their definition of security. We do the same below.

A.2.1 Hardness Assumption

The security of the HIBE construction in [6] is based on the intractability of the Bilinear Diffie-Hellman Exponent Problem[1], which we define below.

[1] As explained in [16], the same HIBE construction is also secure under the weaker wBDHI assumption. However, we focus on BDHE, as it is the assumption we will use to prove the security of OAQUE.

Definition 5. *Let $\mathbb{G}$ and $\mathbb{G}_T$ be (multiplicative) cyclic groups of prime order p, and let $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be an efficiently computable bilinear map. The ℓ-Bilinear Diffie Hellman Exponent (ℓ-BDHE) Problem is the following: given $g, h \xleftarrow{\$} \mathbb{G}$ and $g^{(\alpha^i)}$ for $i \in \{1, \ldots, \ell-1, \ell+1, \ldots, 2\ell\}$ and for some fixed but undisclosed $\alpha \xleftarrow{\$} \mathbb{Z}_p$, find $e(g,h)^{(\alpha^\ell)}$.*

The security of our OAQUE construction relies on the assumption that is difficult to even *recognize* a solution to the above problem—in other words, that a decisional variant of ℓ-BDHE is hard. We state the definition more formally below.

Definition 6. *Let P be an instance of the ℓ-BDHE problem:*

$$P = \left(g, h, g^\alpha, \ldots, g^{(\alpha^{\ell-1})}, g^{(\alpha^{\ell+1})}, \ldots, g^{(\alpha^{2\ell})}\right)$$

for some $g, h \xleftarrow{\$} \mathbb{G}$ and $\alpha \xleftarrow{\$} \mathbb{Z}_p$. The Decisional ℓ-BDHE Assumption is that there exists no probabilistic polynomial-time algorithm $\mathscr{B}$ such that

$$\left| \Pr_{g,h,\alpha}\left[\mathscr{B}\left(P, e(g,h)^{(\alpha^\ell)}\right) = 1\right] - \Pr_{g,h,\alpha,T}\left[\mathscr{B}(P, T) = 1\right] \right|$$

is not negligible for $T \xleftarrow{\$} \mathbb{G}_T$.

A.2.2 Definition of Security

Our construction of OAQUE supports Selective-AS Security under Chosen Plaintext Attacks, an analogue of Selective-ID Security defined in [6]. Selective-AS Security means that the attacker commits ahead of time which attribute set he wishes to attack. We define the IND-sAS-CPA game as follows:

Setup: First, $\mathscr{A}$ decides on the attribute set S^* he wishes to attack. The challenger $\mathscr{C}$ runs the `Setup` algorithm and gives $\mathscr{A}$ the system parameters.
Phase 1: $\mathscr{A}$ adaptively issues queries to the challenger $\mathscr{C}$ to generate private keys. In each query, the adversary provides an attribute set S where $S \not\subseteq S^*$, and $\mathscr{C}$ gives $\mathscr{A}$ the private key for that attribute set.
Challenge: $\mathscr{A}$ outputs two plaintexts m_0 and m_1. Then $\mathscr{C}$ picks a random bit $b \in \{0, 1\}$ and sends A the ciphertext of m_b encrypted under S^*.
Phase 2: is the same as Phase 1.
Guess: $\mathscr{A}$ outputs a guess $b' \in \{0, 1\}$ and wins if $b = b'$.

Definition 7. *The advantage of an adversary in the IND-sAS-CPA game is $\left|\Pr[b = b'] - \frac{1}{2}\right|$.*

Definition 8. *An OAQUE scheme is selective-AS secure under Chosen Plaintext Attacks (i.e., IND-sAS-CPA-secure) if no probabilistic polynomial-time adversary can win the IND-sAS-CPA game with non-negligible advantage.*

Now we will prove that the construction of OAQUE given in Section 3.6 is IND-sAS-CPA-secure.

A.2.3 Proof of Security

Our proof is very similar to the proof of security of the HIBE scheme presented in [6]. Like the scheme in [6], our OAQUE scheme is complete for all possible attribute sets, but is only secure if none of the attribute values is 0. Completeness with 0 will be useful in the proof, however.

The security of OAQUE is based on the ℓ-BDHE assumption presented in [6]. [16] proves the intractability of the ℓ-BDHE problem in the generic group model.

Theorem 1. *Let $\mathbb{G}$ be a bilinear group of prime order p, in which the Decisional $(\ell+1)$-BDHE Assumption holds. Then the OAQUE scheme presented in Section 3.6, with ℓ slots, is IND-sAS-CPA secure.*

Proof. Suppose that $\mathscr{A}$ has advantage ε in the IND-sAS-CPA game described above. We will construct an algorithm $\mathscr{B}$ that solves the $(\ell+1)$-BDHE problem in $\mathbb{G}$ with advantage ε.

The inputs to $\mathscr{B}$ are as follows. A generator $g \in \mathbb{G}$ and $\alpha \in \mathbb{Z}_p^*$ are sampled, and $\mathscr{B}$ is given $(g, h, y_1, \ldots, y_\ell, y_{\ell+2}, \ldots, y_{2\ell+2})$, where $y_i = g^{(\alpha^i)}$. $\mathscr{B}$ is also provided $T \in \mathbb{G}$, which is either $e(g,h)^{(\alpha^{\ell+1})}$, or an element chosen uniformly at random from $\mathbb{G}$.

Initialization: $\mathscr{A}$ outputs the attribute set S^* which it wishes to attack. S^* may have some free slots. However, consider the attribute set T^* which is identical to S^*, except that the slots that are free in S^* are filled with zeros in T^*. Encrypting a message under S^* is equivalent to encrypting it under T^*. Therefore, if the attacker can win the game by selecting S^*, then he can win the game by selecting T^* and doing the exact same thing, as 0 is not actually a valid attribute value. So, without loss of generality, S^* has no free slots.

Setup: $\mathscr{B}$ generates the system parameters as follows. B selects $\gamma \xleftarrow{\$} \mathbb{Z}_p$ and sets $g_1 = y_1$ and $g_2 = y_\ell \cdot g^\gamma$. Then $\mathscr{B}$ selects $\gamma_i \xleftarrow{\$} \mathbb{Z}_p$ and computes $h_i = \frac{g^{\gamma_i}}{y_{\ell-i+1}}$. $\mathscr{B}$ then samples $\delta \xleftarrow{\$} \mathbb{Z}_p$ and computes $g_3 = g^\delta \cdot \prod_{i,a \in S^*} y^a_{\ell-i+1}$.

Finally, $\mathscr{B}$ gives to $\mathscr{A}$ the public parameters $(g, g_1, g_2, g_3, h_1, \ldots, h_\ell)$. Note that these parameters are distributed exactly as if they were generated according to the standard OAQUE Setup algorithm.

Phase 1: In this phase, $\mathscr{A}$ issues multiple private key queries, each for an attribute set S that is not a subset of S^*. Note that $\mathscr{B}$ does not have the master key $g_2^\alpha = y_{\ell+1} \cdot y_1^\gamma$. However, we can still compute the private key as follows. First, we know that S is not a subset of S^*. Furthermore, S^* has no free slots; therefore, there exists some $(k, a_k) \in S$ such that $(k, b_k) \in S^*$ but $a_k \neq b_k$. Let $R = (S \cap S^*) \cup \{(k, a_k)\}$. We will first construct the private key for the attribute set R. Once we have that private key, obtaining the private key for S is simple; since $R \subseteq S$, $\mathscr{B}$ can just apply the QualifyKey algorithm to the private key for R.

To obtain the first term of the private key for R, we observe

$$g_3 \cdot \prod_{(i,a_i)\in R} h_i^{a_i} = g^\delta \cdot \left(\prod_{(i,b_i)\in S^*} y_{\ell-i+1}^{b_i}\right) \cdot \prod_{(i,a_i)\in R} \left(\frac{g^{\gamma_i}}{y_{\ell-i+1}}\right)^{a_i}$$

$$= g^{\delta + \sum_{(i,a_i)\in R} \gamma_i \cdot a_i} \cdot \left(\prod_{(i,b_i)\in S^*} y_{\ell-i+1}^{b_i}\right) \cdot \prod_{(i,a_i)\in R} y_{\ell-i+1}^{-a_i}$$

$$= g^{\delta + \sum_{(i,a_i)\in R} \gamma_i \cdot a_i} \cdot \left(\prod_{\substack{(i,b_i)\in S^* \\ (i,a_i)\in R}} y_{\ell-i+1}^{b_i - a_i}\right) \cdot \prod_{\substack{(i,b_i)\in S^* \\ (i,\cdot)\notin R}} y_{\ell-i+1}^{b_i}$$

Observe that, in the middle term, $a_i = b_i$ except when $i = k$, due to how R was constructed. Therefore, the middle term can be rewritten as $y_{\ell-k+1}^{b_k - a_k}$. So

$$g_3 \cdot \prod_{(i,a_i)\in R} h_i^{a_i} = g^{\delta + \sum_{(i,a_i)\in R} \gamma_i \cdot a_i} \cdot y_{\ell-k+1}^{b_k - a_k} \cdot \prod_{\substack{(i,b_i)\in S^* \\ (i,\cdot)\notin R}} y_{\ell-i+1}^{b_i}$$

To compute the private key, $\mathscr{B}$ samples $\tilde{r} \xleftarrow{\$} \mathbb{Z}_p$. Then it takes $r = \tilde{r} + \frac{\alpha^k}{a_k - b_k}$. Observe that $\mathscr{B}$ cannot compute r; however, it will compute the private key obtained with randomness r.

$$g_2^\alpha \cdot \left(g_3 \cdot \prod_{(i,a_i)\in R} h_i^{a_i}\right)^r = (y_{\ell+1} \cdot y_1^\gamma) \cdot \left(g^{\delta + \sum_{(i,a_i)\in R} \gamma_i \cdot a_i}\right)^r$$

$$\cdot \left(y_{\ell-k+1}^{b_k - a_k}\right)^r \cdot \prod_{\substack{(i,b_i)\in S^* \\ (i,\cdot)\notin R}} \left(y_{\ell-i+1}^{b_i}\right)^r$$

$$= (y_{\ell+1} \cdot y_1^\gamma) \cdot \left(g^{\tilde{r}} \cdot g^{\frac{\alpha^k}{a_k - b_k}}\right)^{\delta + \sum_{(i,a_i)\in R} \gamma_i \cdot a_i}$$

$$\cdot \left(y_{\ell-k+1}^{\tilde{r}\cdot(b_k - a_k)} \cdot y_{\ell-k+1}^{-\alpha^k}\right) \cdot \prod_{\substack{(i,b_i)\in S^* \\ (i,\cdot)\notin R}} \left(y_{\ell-i+1}^{\tilde{r}\cdot b_i} \cdot \left(y_{\ell-i+1}^{\alpha^k}\right)^{\frac{b_i}{a_k - b_k}}\right)$$

$$= y_1^\gamma \cdot \left(g^{\tilde{r}} \cdot y_k^{\frac{1}{a_k - b_k}}\right)^{\delta + \sum_{(i,a_i)\in R} \gamma_i \cdot a_i}$$

$$\cdot y_{\ell-k+1}^{\tilde{r}\cdot(b_k - a_k)} \cdot \prod_{\substack{(i,b_i)\in S^* \\ (i,\cdot)\notin R}} \left(y_{\ell-i+1}^{\tilde{r}\cdot b_i} \cdot (y_{2\ell-i+1})^{\frac{b_i}{a_k - b_k}}\right)$$

Now, observe that $\mathscr{B}$, using the information it has, can compute the above quantity in polynomial time. Thus, $\mathscr{B}$ can compute the first component of the private key for R.

Computing the second and third components is more straightforward. The second component is

$$g^r = g^{\tilde{r}} \cdot g^{\alpha^k \frac{1}{a_k - b_k}} = g^{\tilde{r}} \cdot y_k^{\frac{1}{a_k - b_k}}$$

The third component is a set of elements of the form

$$h_j^r = h_j^{\tilde{r}} \cdot \left(\frac{g^{\gamma_j}}{y_{\ell-j+1}} \right)^{\frac{\alpha^k}{a_k - b_k}} = h_j^{\tilde{r}} \cdot \left(\frac{y_k^{\gamma_j}}{y_{\ell-j+k+1}} \right)^{\frac{1}{a_k - b_k}}$$

Clearly, $\mathscr{B}$ can compute both of these elements in polynomial time. Therefore, $\mathscr{B}$ can compute the private key for R. Because $R \subseteq S$, it can compute the private key for S from this private key by executing the QualifyKey algorithm, using only the public OAQUE parameters and the private key for R.

Challenge: When $\mathscr{A}$ decides to move to the challenge phase, it outputs two message, $m_0, m_1 \in \mathbb{G}_T$. $\mathscr{B}$ decides which message to encrypt by selecting $b \xleftarrow{\$} \{0,1\}$. Then, it produces a ciphertext for m_b as follows:

$$\left(m_b \cdot T \cdot e(g_1, h^\gamma), \quad h, \quad h^{\delta + \sum_{(i,a_i) \in S^*} \gamma_i \cdot a_i} \right)$$

If $T = e(g,h)^{(\alpha^{\ell+1})}$, then the above is a valid ciphertext for m_b. To see this, define c such that $h = g^c$, and observe:

$$e(g,h)^{(\alpha^{\ell+1})} \cdot e(g_1, h^\gamma) = \left(e(g,g)^{(\alpha^{\ell+1})} \cdot e(g_1, g^\gamma) \right)^c$$
$$= \left(e(g_1, y_\ell) \cdot e(g_1, g^\gamma) \right)^c = e(g_1, g_2)^c$$

The second term (since $h = g^c$) and third term are also identical to the corresponding term in the ciphertext:

$$h^{\delta + \sum_{(i,a_i) \in S^*} \gamma_i \cdot a_i} = h^\delta \cdot \prod_{(i,a_i) \in S^*} h^{\gamma_i \cdot a_i}$$
$$= \left(g^\delta \cdot \prod_{(i,a_i) \in S^*} g^{\gamma_i \cdot a_i} \right)^c = \left(g_3 \cdot \prod_{(i,a_i) \in S^*} h_i^{a_i} \right)^c$$

Furthermore, because h was chosen uniformly at random from $\mathbb{G}$, c is distributed uniformly at random in $\mathbb{Z}_p$. Therefore, when $T = e(g,h)^{(\alpha^{\ell+1})}$, the result looks exactly like a valid ciphertext, and is distributed correctly.

If T is chosen at random, the first element of the ciphertext is distributed uniformly over $\mathbb{G}$. Therefore, the choice of b is not preserved in the "ciphertext" given to $\mathscr{A}$.

Phase 2: $\mathscr{B}$ responds to queries just as it does in Phase 1.

Guess: Finally, $\mathscr{A}$ outputs $b' \in \{0,1\}$. Using this result, $\mathscr{B}$ does the following. If $\mathscr{A}$ guessed correctly (i.e., $b' = b$) then $\mathscr{B}$ outputs 1. Otherwise (i.e., $b' \neq b$), $\mathscr{B}$ outputs 0.

As discussed earlier, if $T = e(g,h)^{(\alpha^{\ell+1})}$, then $\mathscr{A}$ sees information exactly consistent with the OAQUE scheme described above, and therefore guesses correctly with probability $\frac{1}{2} \pm \varepsilon$. However, if T was selected at random, then the ciphertext has the same distribution regardless of b, so $\mathscr{A}$ guesses correctly with probability $\frac{1}{2}$. Therefore,

$$\left| \Pr_{g,h,\alpha} \left[\mathscr{B}\left(P, e(g,h)^{(\alpha^\ell)}\right) = 1 \right] - \Pr_{g,h,\alpha,T} \left[\mathscr{B}(P,T) = 1 \right] \right|$$
$$= \left| \left(\frac{1}{2} \pm \varepsilon \right) - \frac{1}{2} \right| = \varepsilon$$

Therefore, if $\mathscr{A}$ can win the IND-sAS-CPA game with non-negligible probability (i.e., if ε is non-negligible) then $\mathscr{B}$ can solve the Decisional $(\ell+1)$-Bilinear Diffie-Hellman Exponent Problem with non-negligible probability. Assuming that the Decisional $(\ell+1)$-BDHE Assumption holds for the chosen $\mathbb{G}$, $\mathbb{G}_T$, and e, this means that there exists no probabilistic polynomial-time algorithm that can win the IND-sAS-CPA game with non-negligible probability. $\square$

References

[1] Andersen, M. P., Kolb, J., Chen, K., Culler, D. E., and Katz, R. Democratizing authority in the built environment. In *Proceedings of the 4th ACM International Conference on Systems for Energy-Efficient Built Environments* (2017).

[2] Andersen, M. P., Kolb, J., Chen, K., Fierro, G., Culler, D. E., and Popa, R. A. Wave: A decentralized authorization system for iot via blockchain smart contracts. Tech. Rep. UCB/EECS-2017-234, EECS Department, University of California, Berkeley, Dec 2017.

[3] Benet, J. IPFS - content addressed, versioned, P2P file system. *CoRR abs/1407.3561* (2014).

[4] Bethencourt, J., Sahai, A., and Waters, B. Ciphertext-policy attribute-based encryption. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2007), SP '07, IEEE Computer Society, pp. 321–334.

[5] Blaze, M., Bleumer, G., and Strauss, M. Divertible protocols and atomic proxy cryptography. *Proceedings of the 16th Annual International Conference on the Theory and Applications of Cryptographic Techniques* (1998), 127–144.

[6] Boneh, D., Boyen, X., and Goh, E.-J. Hierarchical identity based encryption with constant size ciphertext. In *Proceedings of the 23rd Annual International Conference on the Theory and Applications of Cryptographic Techniques* (2005), Springer, pp. 440–456.

[7] Boneh, D., and Franklin, M. Identity-based encryption from the weil pairing. In *Advances in CryptologyCRYPTO 2001* (2001), Springer, pp. 213–229.

[8] Borcea, C., Gupta, A. B. D., Polyakov, Y., Rohloff, K., and Ryan, G. Picador: End-to-end encrypted publishsubscribe information distribution with proxy re-encryption. *Future Generation Computer Systems 71*, Supplement C (2017), 177 – 191.

[9] Canetti, R., Halevi, S., and Katz, J. A forward-secure public-key encryption scheme. In *Proceedings of the 21st Annual International Conference on the Theory and Applications of Cryptographic Techniques* (2003), Springer, pp. 255–271.

[10] Cheng, R., Scott, W., Parno, B., Zhang, I., Krishnamurthy, A., and Anderson, T. Talek: a private publish-subscribe protocol. Tech. Rep. 16-11-01, University of Washington CSE, 2015.

[11] Cheon, J. H. Security analysis of the strong diffie-hellman problem. In *Proceedings of the 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques* (2006), Springer-Verlag, pp. 1–11.

[12] Corrigan-Gibbs, H., Boneh, D., and Mazires, D. Riposte: An anonymous messaging system handling millions of users. In *2015 IEEE Symposium on Security and Privacy* (May 2015), pp. 321–338.

[13] Dodis, Y., and Fazio, N. Public key broadcast encryption for stateless receivers. In *Digital Rights Management Workshop* (2002), vol. 2696, Springer, pp. 61–80.

[14] Drago, I., Mellia, M., M Munafo, M., Sperotto, A., Sadre, R., and Pras, A. Inside dropbox: understanding personal cloud storage services. In *Proceedings of the 2012 ACM conference on Internet measurement conference* (2012), ACM, pp. 481–494.

[15] Egorov, M., and Wilkison, M. Nucypher KMS: decentralized key management system. *CoRR abs/1707.06140* (2017).

[16] Goh, E.-J. *Encryption schemes from bilinear maps*. PhD thesis, Stanford University, 2007.

[17] Goh, E.-J., Shacham, H., Modadugu, N., and Boneh, D. Sirius: Securing remote untrusted storage. In *Proceedings of the Network and Distributed Systems Security (NDSS) Symposium 2003* (2003), vol. 3, pp. 131–145.

[18] Goyal, V., Pandey, O., Sahai, A., and Waters, B. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM Conference on Computer and Communications Security* (2006), ACM, pp. 89–98.

[19] Goyal, V., Pandey, O., Sahai, A., and Waters, B. Attribute-based encryption for fine-grained access control of encrypted data. Cryptology ePrint Archive, Report 2006/309, 2006. `http://eprint.iacr.org/`.

[20] Horwitz, J., and Lynn, B. Toward hierarchical identity-based encryption. In *Proceedings of the 20th Annual International Conference on the Theory and Applications of Cryptographic Techniques* (2002), Springer, pp. 466–481.

[21] Kallahalla, M., Riedel, E., Swaminathan, R., Wang, Q., and Fu, K. Plutus: Scalable secure file sharing on untrusted storage. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST)* (2003), USENIX.

[22] Kawahara, Y., Kobayashi, T., Scott, M., and Kato, A. Barreto-naehrig curves. Tech. rep., Internet-Draft draft-kasamatsu-bncurves-02. Internet Engineering Task Force. https://datatracker. ietf. org/doc/html/draft-kasamatsu-bncurves-02 Work in Progress, 2016.

[23] Merkle, R. C. A certified digital signature. In *Conference on the Theory and Application of Cryptology* (1989), Springer, pp. 218–238.

[24] Naehrig, M., Niederhagen, R., and Schwabe, P. New software speed records for cryptographic pairings. In *International Conference on Cryptology and Information Security in Latin America* (2010), Springer, pp. 109–123.

[25] Naor, D., Naor, M., and Lotspiech, J. Revocation and tracing schemes for stateless receivers. In *Advances in CryptologyCRYPTO 2001* (2001), Springer, pp. 41–62.

[26] Polyakov, Y., Rohloff, K., Sahu, G., and Vaikuntanthan, V. Fast proxy re-encryption for publish/subscribe systems. Cryptology ePrint Archive, Report 2017/410, 2017. `https://eprint.iacr.org/2017/410`.

[27] Sahai, A., and Waters, B. Fuzzy identity-based encryption. In *Proceedings of the 24th Annual International Conference on Theory and Applications of Cryptographic Techniques* (2005), Springer-Verlag, pp. 457–473.

[28] van den Hooff, J., Lazar, D., Zaharia, M., and Zeldovich, N. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles* (New York, NY, USA, 2015), SOSP '15, ACM, pp. 137–152.

[29] Wang, G., Liu, Q., and Wu, J. Hierarchical attribute-based encryption for fine-grained access control in cloud storage services. In *Proceedings of the 17th ACM conference on Computer and communications security* (2010), ACM, pp. 735–737.

[30] Wang, G., Liu, Q., Wu, J., and Guo, M. Hierarchical attribute-based encryption and scalable user revocation for sharing data in cloud servers. *Computers & Security 30*, 5 (2011), 320 – 331. Advances in network and system security.

[31] Yu, S., Wang, C., Ren, K., and Lou, W. Achieving secure, scalable, and fine-grained data access control in cloud computing. In *2010 Proceedings IEEE INFOCOM* (March 2010), pp. 1–9.