Cryptographically Enforced Orthogonal Access Control at Scale

Bob Wall
IronCore Labs, Inc.
Bozeman, Montana
bob.wall@ironcorelabs.com

Patrick Walsh
IronCore Labs, Inc.
Boulder, Colorado
patrick.walsh@ironcorelabs.com

ABSTRACT

We propose a new approach to cryptographically enforced data access controls that uses public key cryptography to secure large numbers of documents with arbitrarily large numbers of authorized users. Our approach uses a *proxy re-encryption (PRE)* scheme to handle the problems typical of public key cryptography including key management, rotation, and revocation, in a highly scalable way, while providing end-to-end encryption and provable access.

In this paper we describe a system based on this approach. We call it an *orthogonal access control* system, because it allows the decision about the groups to which to encrypt a piece of data to be made independently and asynchronously from the decision about who belongs to a group and can therefore decrypt the data. We define specific requirements for a PRE scheme needed to support the system, and we provide a specific instance that meets these requirements. We detail the algorithms that make up the scheme, and we present an enhancement that provides better revocability of keys.

CCS CONCEPTS

• **Security and privacy → Access control**; **Key management**; **Public key encryption**;

KEYWORDS

cryptographic access control, proxy re-encryption, pairing-based cryptography, key management, orthogonal access control

ACM Reference Format:
Bob Wall and Patrick Walsh. 2018. Cryptographically Enforced Orthogonal Access Control at Scale. In *SCC'18: 6th International Workshop on Security in Cloud Computing, June 4, 2018, Incheon, Republic of Korea.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3201595.3201602

1 INTRODUCTION

Securing data in transit, at rest, and within applications is difficult, particularly at cloud scale. As data becomes increasingly distributed between cloud services, mobile devices, the Internet of Things, and portable media, managing access to that data wherever it is stored or used is progressively more difficult. Cryptographic techniques are commonly used to increase trust in cloud services and to provide more reliable security and access control in a world of distributed data. In particular, *end-to-end encryption* is seeing more wide-spread adoption. However, current end-to-end encryption tools are mostly restricted to person-to-person communication applications like messaging and email.

Practical systems that manage access to data by groups of users typically resort to a model using an access control server that manages symmetric keys and hands them to requesting users based on policy. While public key-based systems have a better security model, they suffer from scalability and complexity problems. Consider, for example, using PGP to secure data. A user can securely encrypt a file to a set of other users, given their public keys. However, there is a linear increase in the time and space required to encrypt the file to each user. When access to an encrypted file must be granted to a new user, someone with access to the file must decrypt it, then re-encrypt it to the entire list, plus the new recipient. If files are encrypted to teams of people and a person leaves the team, every file shared with the team must be found, decrypted, and re-encrypted to the list minus the departing user. The same process must be followed if a user's keys are compromised. This solution does not scale to large groups of users.

We propose a system that is built to embed in applications to provide cryptographically backed access controls for data. The system uses public key cryptography, and it overcomes the scalability difficulties using abstract entities that represent groups of users. These groups facilitate *orthogonal access control*, because users can choose the groups to which to encrypt data, while group administrators can independently add and remove users at any time. These changes are done in constant time and are independent of the number of groups, group members, or affected files in the system. The system provides end-to-end encryption, with the private keys required to decrypt data retained on users' devices. This allows servers to be semi-trusted, with no access to unencrypted data or ability to decrypt data.

Proxy re-encryption (PRE) schemes provide properties that are well suited to building such a system. In the remainder of the paper, we will first present background on proxy re-encryption, then describe our orthogonal access control system. We then present a previously developed PRE scheme that meets our requirements. We describe modifications we have made to the algorithm that improve performance by eliminating features we do not use. We then present a multi-party computation technique for augmenting the asymmetric key pairs to enhance the system's ability to revoke access to encrypted data.

2 BACKGROUND

The idea of proxy re-encryption was first introduced in 1998 by Blaze, Bleumer, and Strauss, who provided a concrete scheme based

ACM ISBN 978-1-4503-5759-3/18/06…$15.00
https://doi.org/10.1145/3201595.3201602

on El Gamal public key cryptography [2]. This PRE scheme had the following properties:

(1) Each participant has a public-private key pair.
(2) A participant, the *delegator*, can delegate decryption of messages to a *delegatee*. The delegator generates a *re-encryption key* that can be used to re-encrypt any messages encrypted to her public key so that the delegatee can decrypt them using her private key. This avoids the need for the delegator to share her private key with the delegatee.
(3) The re-encryption keys are held by one or more semi-trusted *proxies* that perform the re-encryption of messages. A re-encryption key does not allow decryption of messages or provide access to either party's private key.
(4) Delegation is revoked when a proxy deletes the relevant re-encryption key.

For example, if Alice plans to be unavailable for a period of time and wants to delegate her messages to Bob, she uses her private key and Bob's public key to generate a re-encryption key, which she stores with her proxy. Each message that is encrypted to Alice and delivered to the proxy is re-encrypted to Bob and is delivered to him. When Alice is again available and wants to revoke delegation, she just removes the re-encryption key from the proxy.

In 2006, Ateniese et al. introduced the first *unidirectional* PRE scheme [1]. The authors also enumerated a list of useful properties of PRE protocols, including the following:

(1) *Directionality*, which describes whether delegation from A to B also allows re-encryption from B to A. Unidirectional schemes do not allow this.
(2) *Interactivity*, which describes whether both parties must be actively involved in order to generate the re-encryption key. A non-interactive scheme only requires the public key of the delegatee.
(3) *Transitivity*, which describes whether a proxy can re-delegate decryption. That is, if the proxy holds a re-encryption key from A to B and a re-encryption key from B to C, can it generate a re-encryption key from A to C? A non-transitive scheme does not allow this.
(4) *Collusion safety*, which describes whether it is possible for a delegatee to collaborate with a proxy that holds a re-encryption key to that delegatee to recover the secret key of the delegator. A collusion-safe scheme does not allow this.

In 2007, Canetti and Hohenberger proposed a definition of security against chosen-ciphertext attacks (CCA-security) on PRE schemes and introduced an algorithm that satisfied the definition [4]. They also outlined several open problems in PRE construction, including the construction of a unidirectional PRE scheme that was also *multi-hop* (also called *multi-use*); that is, a scheme that allows an encrypted message that has been re-encrypted from Alice to Bob to subsequently be re-encrypted from Bob to Carol, if Bob has delegated access to Carol. (The scheme they proposed was multi-hop, but it was bidirectional.)

In 2009, Wang and Cao proposed a scheme that addressed this problem – a CCA-secure unidirectional, multi-hop, collusion-safe, non-interactive PRE algorithm [9]. The CCA-security of their scheme was subsequently challenged by Zhang and Wang[13]. In 2014, Cai and Liu expanded on the issue and introduced a second security issue with the algorithm, then provided a modification that resolved both problems [3]. They also included a proof of CCA-security for the modified scheme.

This PRE scheme, like most others that have been proposed, provides five distinct cryptographic primitives:

- KeyGen – client-side generation of public/private key pair. This is the standard elliptic curve key generation algorithm.
- ReKeyGen – client-side generation of a re-encryption key between a pair of entities.
- Encrypt – client-side encryption of a message to a recipient.
- ReEncrypt – proxy-side re-encryption of an encrypted message.
- Decrypt – client-side decryption of an encrypted or re-encrypted message.

2.1 Prior Research

Proxy re-encryption has been a field of active research since it was first introduced by Blaze et al.

There has been a large volume of research on access control in cloud computing environments, including the publication on outsourcing computation without relinquishing control by Chow et al. [5], the work on cloud-scale fine grain access control by Yu et al. [12], and the work on end-to-end secure content sharing by Xiong et al. [10]. Several different cryptographic approaches have been proposed, including identity-based and attribute-based approaches such as the hierarchical ABE scheme proposed by Wang et al. [8].

Proxy re-encryption has featured prominently in a significant amount of research. Among the many works, Xu et al. describe a certificateless PRE scheme [11], and Liu et al. propose a time-limited delegation scheme based on PRE [6]. Qin et al. present a comprehensive survey of works proposing PRE for use in data sharing in the cloud [7].

2.2 Terminology

The term *re-encryption* is often used to describe the process of decrypting a message then encrypting it with a new key. In a PRE scheme, however, re-encryption refers to the *transformation* of an encrypted message without decryption and without the ability to learn about the message or the keys that could be used to decrypt it. We find the word transformation to be more descriptive of the process, so we will hereafter use that term instead of re-encryption. For instance, we reference TransformKeyGen and Transform in the following, rather than ReKeyGen and ReEncrypt.

3 ORTHOGONAL ACCESS CONTROL SYSTEM

We have built a system that can be used to add strong security and cryptographically backed access control to an application. Specifically, the system provides orthogonal access control capabilities. The system manages users and allows individual users to encrypt data and grant access to that data to other individuals, but it also introduces *groups* that are collections of users. This allows a user to choose one or more groups to whom to grant access to encrypted data. Independently and asynchronously, group administrators choose which users belong to the groups. Adding a member to the group is simply accomplished in constant time, irrespective of the size of the group or the amount of data the group can access. Likewise, removing a member from a group is a constant time operation that does not require modification of any encrypted data.

In addition to providing groups as collections of users, the user represents a collection of devices. A user must utilize some computing device to access and decrypt data, so the system allows each user to control which devices are authorized to access their encrypted data. A user can subsequently decide to revoke access from a particular device (say a smart phone that was lost or stolen).

The individual user delegation capability that PRE schemes provide can be readily adapted to provide orthogonal access control. A PRE scheme that is unidirectional, non-interactive, non-transitive, and collusion safe is a solid foundation for an end-to-end encrypted system where users do not need to trust the server to keep data secure. A multi-hop scheme allows the system to provide delegation from groups to users and from users to devices.

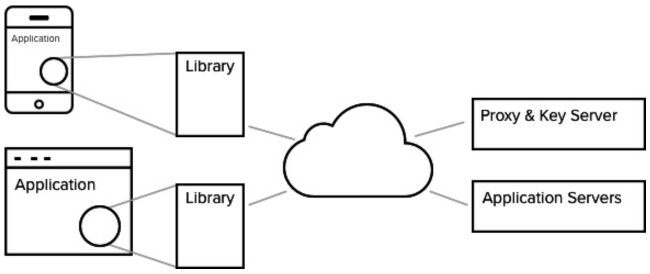

Figure 1: System architecture

Our system consists of a library that is embedded in an application, along with a service that acts as the proxy and also as the key server, storing public keys for groups and users and allowing clients to retrieve them. To facilitate compatibility of this system with a variety of applications and systems, the PRE library does not handle user authentication. The application is responsible for providing a signed assertion of the user's identity; the system is configured with the public key to validate the signature for the app. Likewise, the system does not handle the storage of encrypted application data; the application is free to store this data in a way that is compatible with the rest of the application.

3.1 Cryptographic Protocols

Our cryptosystem provides the following cryptographic protocols and operations, built on the primitives provided by PRE. The membership relationship (a user belonging to a group or a device belonging to a user) is represented by a PRE transform key.

Initialize User - adds a new user to the system. The client invokes the underlying KeyGen to create an encryption key pair for the user and the signing function $\mathcal{G}$ to generate a corresponding signing key pair. The client provides a method to encrypt the private keys, so they can be escrowed securely. The client registers the user's ID, public key, and encrypted private keys on the proxy, then invokes KeyGen and $\mathcal{G}$ again to create separate key pairs for the current device. The client stores the device's private key in secure local storage on the device, then invokes TransformKeyGen to create the transform key from the user to the device. The client registers the device public keys and the transform key on the proxy.

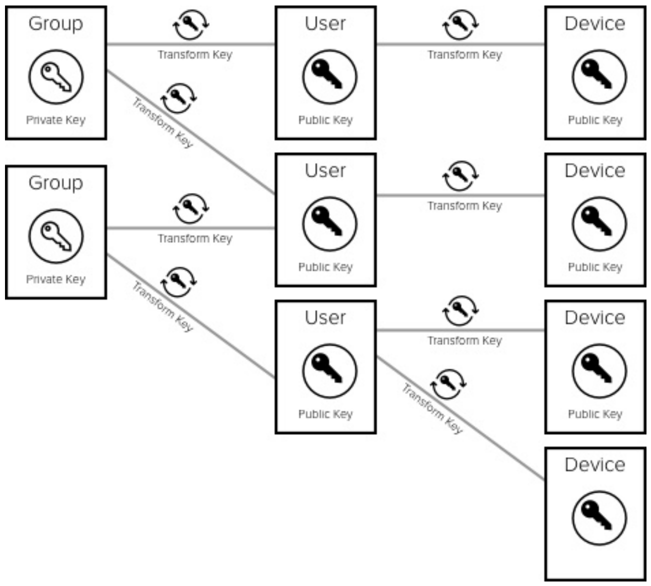

Figure 2: Relationship of Groups and Users

Add Device - authorizes a new device to access the user's data. When the client accesses the system from a new device that does not have device private encryption and signing keys, the client retrieves the user's escrowed private encryption key and decrypts it. The client invokes KeyGen and $\mathcal{G}$ to create encryption and signing key pairs for the current device. The client stores the private keys in secure local storage on the device, then invokes TransformKeyGen to create the transform key from the user to the device. The client registers the user's identity, device public keys, and the transform key on the proxy, then discards the user's private encryption key as it is no longer needed. The device private keys are not escrowed on the proxy; if they are lost, the user re-registers the device using this process.

Remove Device - revokes authorization for a device to access the user's data. If a user decides to revoke access, she uses another authorized device to perform the operation. This request is sent to the proxy, causing it to remove the transform key from the user to the specified device.

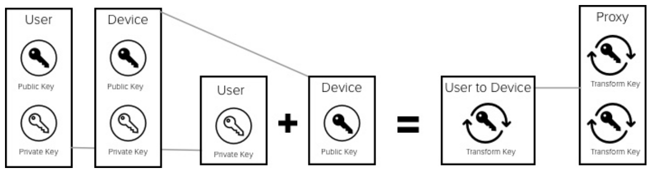

Figure 3: Transform Key from User to Device

Create Group - adds a new group to the system. The user that creates the group is the initial administrator for the group. The client invokes KeyGen to create a key pair for the group, then invokes Encrypt, treating the group's private key as the message and encrypting it to the public key of the current user; this makes the creating user an administrator. The client also invokes

TransformKeyGen to compute a transform key from the group to the creating user; this makes the creating user the first member of the group. The client registers the group ID, public key, user ID and encrypted private key for the administrator, and transform key for the group member with the proxy.

Add Member to Group. This must be invoked by a user that is an administrator for the group, and it requires that the user to be added as a group member must have already been added to the system (implying that the user has a public key). The client first retrieves the public key of the user to be added from the proxy. If this is successful, the client requests the group private key from the proxy. The proxy looks up the group, determines whether the requesting user is an administrator, and if so, locates the transform key from the requesting user to the requesting device. If this key is found, the proxy invokes Transform to transform the encrypted private key from the user to the device and returns the transformed encrypted key. The client uses the device's private key to invoke Decrypt to recover the group private key. Once the client has the group's private key and the user's private key, it invokes TransformKeyGen to generate a transform key, and it registers that with the proxy.

Remove Member from Group. This must be invoked by a user that is an administrator for the group. The administrator requests that the proxy remove a specific user from the group. The proxy confirms that the requesting user is an administrator of the group and if so, looks up the transform key from the group to the user to be removed. If this is found, the proxy deletes the transform key.

The system also allows a group member to remove herself from the group. The proxy confirms that the requesting user is a group member, and if so, locates the transform key from the group to that user and deletes it.

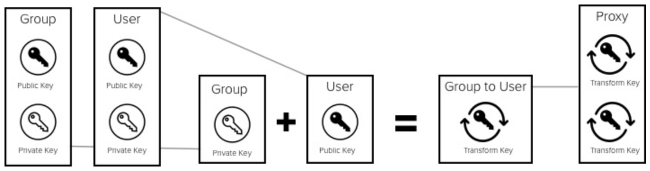

Figure 4: Transform Key from Group to User

Add Admin to Group. This must be invoked by a group administrator. It is similar to the protocol to add a member to the group; the new admin must be a user in the system. The client retrieves that user's public key and the group's private key. Once the client has decrypted the group's private key, it invokes Encrypt using the new admin's public key, and it associates the new admin's user ID and the new encrypted public key with the group and registers this information with the proxy.

Remove Admin from Group. This must be invoked by a group administrator. The operation is similar to removing a member from the group; the client sends the group ID and the user ID of the administrator to be removed to the proxy. The proxy confirms that the requesting user is an admin for the specified group. If so, it locates the encrypted private key for the group that is associated with the admin to be deleted. If that entry is found, the proxy deletes it.

Encrypt Document is invoked with a document and an associated unique identifier. The client first chooses a random *document encryption key (DEK)* and invokes a symmetric key encryption primitive to encrypt the document data. It then retrieves the current user's public key from the proxy and invokes Encrypt using that key, with the DEK as the data to be encrypted. This produces an *encrypted DEK (EDEK)* that is associated with the user's ID. The client sends the document ID, the current user's ID, and the EDEK to the proxy for storage. The library returns the encrypted document to the embedding application for storage.

Grant Access is invoked from a specific device on behalf of the current user. It requires a document ID and a list of recipients that are allowed to access the document, each of which can be a group or an individual. The user that invokes this method must have initially encrypted the document or been granted access previously. The client first requests the EDEK for the document from the proxy. The proxy searches for the shortest access path that is available between the document and the user. This involves searching the list of EDEKs for the document to find one that grants access to the user. If there isn't one, it looks for one that grants access to a group that includes the user. If it finds an EDEK, it invokes Transform on the entry to transform it to the user's device key and returns it. The client uses the device's private key to decrypt the response and retrieve the DEK. The client then retrieves the public key of each recipient from the proxy and invokes Encrypt with the DEK and each public key to generate a new list of EDEKs. The client sends this list along with the document ID to the proxy for storage.

Decrypt Document is invoked from a specific device on behalf of the current user. Similar to the sharing case, the client sends a request for the EDEK to the proxy. The proxy searches for the EDEK and if found, transforms it to the user's device and returns it. The client decrypts the EDEK using the device private key to retrieve the DEK. If this is successful, the client invokes the symmetric key decryption primitive, using the DEK to decrypt the encrypted document that was provided by the embedding application.

Revoke Access is used to remove access to a specific document from a user or group. The client passes the document ID, current user ID, and the ID of the user or group whose access is being revoked to the proxy. The proxy confirms that the requesting user has access to the document, and if so, deletes the EDEK entry for the specified user or group.

Update Document is invoked to update the contents of a previously encrypted document. The client first verifies that the current user has access to the document by searching for a path from the document to the user. If this exists, the client randomly chooses a new DEK and uses it to symmetrically encrypt the new version of the document. It then requests the list of EDEKs from the proxy. For each EDEK, it retrieves the public key, and it invokes Encrypt with the new DEK and the public key. It sends the document and the replacement list of EDEKs to the proxy for storage, and it returns the encrypted document to the embedding application for storage.

This method is provided to allow for revocation of access. If a user was granted access to a document at some point in time, used the system to decrypt the document, and captured the DEK from the client, she could retain this DEK. At a future time, after her access was revoked, if she could retrieve the encrypted data for an updated version of the document, she could use the DEK directly to decrypt it. By rotating the DEK each time the document is updated, she is limited to only decrypting versions of the document to which she was granted access.

4 PRE ALGORITHMS

Our system uses the PRE algorithms described by Cai and Liu, with some simplifications. Our system has a single proxy which performs all transformations for multiple hops at the same time. Rather than computing an authentication code for each hop using the relatively expensive pairing, as specified in the Wang algorithm, our proxy signs the entire ciphertext or transformed ciphertext with a simple message hash and a much faster Ed25519 signature, using a separate signing key.

The following algorithms describe the primitives of the Cai and Liu PRE scheme, with our signature changes. As a reminder, we renamed ReKeyGen to TransformKeyGen and ReEncrypt to Transform.

The five primitives are KeyGen, Encrypt, TransformKeyGen, Transform, and Decrypt.

Let $params = (k, p, \mathbb{G}_1, \mathbb{G}_T, e, \mathbf{g}, \mathbf{g_1}, \mathsf{SHA256}, \mathbf{H_2}, Sig)$ be public parameters, where:

- k is the number of bits required to store keys;
- p is a prime;
- $\mathbb{G}_1$ and $\mathbb{G}_T$ are abelian groups with $\mathbb{G}_1$ written additively and $\mathbb{G}_T$ written multiplicatively;[1]
- $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T$ is a bilinear pairing;
- $\mathbf{g}$ is an arbitrary fixed nonzero element of $\mathbb{G}_1$.
- $\mathbf{g_1}$ is a random element of $\mathbb{G}_1$ which does not lie in the cyclic subgroup generated by $\mathbf{g}$.
- $\mathsf{SHA256} : \{0,1\}^* \to$ 256-bit hash and $\mathbf{H_2} : \mathbb{G}_T \to \mathbb{G}_1$ are two one-way collision-resistant hash functions.
- $Sig = (\mathcal{G}, \mathcal{S}, \mathcal{V})$ is the Ed25519 strongly unforgeable signature scheme, with a key generation algorithm, a signing algorithm, and a verification algorithm.

$\mathsf{KeyGen}(params) \to (\mathbf{pk}, sk)$:

Generate a public/private key pair.

(1) *secret key* $sk \leftarrow_R \mathbb{Z}_p^*$
(2) *public key* $\mathbf{pk} \leftarrow sk \cdot \mathbf{g}$

$\mathsf{TransformKeyGen}(params, sk_i, \mathbf{pk}_j, (spk_i, ssk_i)) \to tk_{i\to j}$:

Generate a transform key from user i (the delegator) to user j (the delegatee). Requires the delegator's private key (sk_i), the delegatee's public key ($\mathbf{pk}_j$), and the delegator's signing key pair (spk_i, ssk_i). Produces a transform key that is a tuple of five values.

(1) *transform key pair* $(\mathbf{tpk}, tsk) \leftarrow \mathsf{KeyGen}$
(2) *transform value* $K \leftarrow_R \mathbb{G}_T$ Âă
(3) *encrypted transform value* $eK \leftarrow K \cdot e(\mathbf{pk}_j, \mathbf{g_1})^{tsk}$
(4) *signature* $sig \leftarrow \mathcal{S}(\mathbf{tpk}||eK||spk_i, ssk_i)$
(5) *transform point* $\mathbf{tep} \leftarrow \mathbf{H_2}(K) + (-sk_i) \cdot \mathbf{g_1}$
(6) *transform key* $tk_{i\to j} \leftarrow (\mathbf{tpk}, eK, spk_i, sig, \mathbf{tep})$

Note that the signature does not include **tep**. In the transformation process, this value does not get copied into the transform block, so omitting it from the signature allows the signature to be copied into the transform block and verified if desired.

The transform key $tk_{i\to j}$ is sent to the proxy via a secure channel.

$\mathsf{Encrypt}(params, m, \mathbf{pk}_j, (spk_i, ssk_i)) \to C_j$:

Encrypt a message $m \in \mathbb{G}_T$ to delegatee j, given j's public key ($\mathbf{pk}_j$) and the sender i's signing key pair (spk_i, ssk_i). Produces a ciphertext C_j that is a tuple of five values.

(1) *ephemeral key pair* $(\mathbf{epk}, esk) \leftarrow \mathsf{KeyGen}$
(2) *encrypted message* $em \leftarrow m \cdot e(\mathbf{pk}_j, \mathbf{g_1})^{esk}$
(3) *authentication hash* $ah \leftarrow \mathsf{SHA256}(epk||m)$
(4) *signature* $sig \leftarrow \mathcal{S}(\mathbf{epk}||em||ah||spk_i, ssk_i)$
(5) *ciphertext* $C_j \leftarrow (\mathbf{epk}, em, ah, spk_i, sig)$

$\mathsf{Transform}(params, C_i, tk_{i\to z}, (spk, ssk)) \to C_j$ – or –
$\mathsf{Transform}(params, C_i, [tk_{i\to a}, ..., tk_{y\to z}], (spk, ssk)) \to C_j$

Transform a ciphertext encrypted to i (C_i) into a ciphertext encrypted to j (C_j), given a list of one or more transform keys (tk), provided in the order in which they must be applied, and the proxy's signing key pair (spk, ssk). (This operation is performed by the proxy rather than the client.)

First, validate the signature on the encrypted message and on each of the transform keys:

(1) If any parse or verify step fails, return $\perp$
(2) Parse C_i into $(\mathbf{epk}, em, ah, spk_m, sig_m)$
(3) Verify $\mathcal{V}(\mathbf{epk}||em||ah||psk_m, sig_m, spk_m)$
(4) Parse each tk into $(\mathbf{tpk}, eK, spk_{tk}, sig_{tk}, \mathbf{tep})$
(5) Verify $\mathcal{V}(\mathbf{tpk}||eK||spk_{tk}, sig_{tk}, spk_{tk})$

Apply the first transform from the list:

(1) *random key pair* $(\mathbf{rpk}, rsk) \leftarrow \mathsf{KeyGen}$
(2) *random transform value* $rK \leftarrow_R \mathbb{G}_T$
(3) *random encrypted transform value*
$reK \leftarrow rK \cdot e(\mathbf{pk}_j, \mathbf{g_1})^{rsk}$
(4) *transformed encrypted message*
$em' \leftarrow em \cdot e(\mathbf{epk}, \mathbf{tep} + \mathbf{H_2}(rK))$
(5) *modified ciphertext* $C'_i \leftarrow (\mathbf{epk}, em', ah)$
(6) *transform block* $TB \leftarrow (\mathbf{tpk}, eK, \mathbf{rpk}, reK)$
(7) *transformed ciphertext* $C_j \leftarrow (C'_i, TB)$

If there are additional transform keys in tk, process each of them in turn.

(1) Parse the last transform block of C_j into
$(\mathbf{tpk}_{prev}, eK_{prev}, \mathbf{rpk}_{prev}, reK_{prev})$
(2) Parse the next transform key from the list, tk, into
$(\mathbf{tpk}, eK, spk_{tk}, sig_{tk}, \mathbf{tep})$
(3) *random key pair,* $(\mathbf{rpk}, rsk) \leftarrow_R \mathbb{Z}_p^*$
(4) *random transform value,* $rK \leftarrow_R \mathbb{G}_T$
(5) *random encrypted transform value*
$reK \leftarrow rK \cdot e(\mathbf{pk}_j, \mathbf{g_1})^{rsk}$
(6) *transformed encrypted transform value*
$eK' \leftarrow eK_{prev} \cdot e(\mathbf{tpk}_{prev}, \mathbf{tep} + \mathbf{H_2}(rK))$
(7) *transformed random encrypted transform value*
$reK' \leftarrow reK_{prev} \cdot e(\mathbf{rpk}, \mathbf{tep} + \mathbf{H_2}(rK))$

[1]Throughout, we will use **bold** to denote elements of $\mathbb{G}_1$.

(8) Replace eK_{prev} and reK_{prev} in the last transform block with eK' and reK'
(9) *transform block* $TB \leftarrow (\mathbf{tpk}, eK, \mathbf{rpk}, reK)$
(10) Append TB to C_j.

In summary, on each transformation (after the first), the last eK and reK values from the previous transformation are modified, then the first two elements of the new transform key and the new **rpk** and reK are appended. Note that the encrypted message em from the original ciphertext C_i is only modified once, in the first transformation. After that, it is not changed again.

After all transforms have been applied, the entire transformed ciphertext is signed.

(1) *signature* $sig \leftarrow \mathcal{S}(C_j||spk, ssk)$
(2) $C_j \leftarrow (C_j, spk, sig)$

Decrypt$(params, C_i, sk_i) \rightarrow m$:

Decrypt a signed ciphertext (C_i) given the private key of the recipient i (sk_i). Returns the original message that was encrypted, m. As above, we return $\perp$ if any parse or verify step fails.

First, validate the signature on the ciphertext:

(1) Extract spk and sig, the last two elements of SC
(2) Extract C, all of C_i preceding spk
(3) Verify $\mathcal{V}(C||spk, sig, spk)$

To decrypt a first-level ciphertext, where C includes no transform blocks:

(1) Parse C into $(\mathbf{epk}, em, ah)$
(2) $m \leftarrow em \cdot e(\mathbf{epk}, (-sk_i) \cdot \mathbf{g_1})$

To decrypt a transformed ciphertext, where C includes l transform blocks:

(1) Parse C into $(C', TB'^{(1)}, \ldots, TB'^{(l-1)}, TB^{(l)})$
(2) Parse C' into $(\mathbf{epk}, em', ah)$
(3) Parse $TB^{(l)}$ into $(\mathbf{tpk}^{(l)}, eK^{(l)}, \mathbf{rpk}^{(l)}, reK^{(l)})$
(4) For each integer k in $[1, l-1]$, parse $TB'^{(k)}$ into $(\mathbf{tpk}^{(k)}, eK'^{(k)}, \mathbf{rpk}^{(l)}, reK^{(l)})$
(5) $K^{(l-1)} \leftarrow eK^{(l)} \cdot e(\mathbf{tpk}^{(l)}, (-sk_i) \cdot \mathbf{g_1})$
(6) $rK^{(l-1)} \leftarrow reK^{(l)} \cdot e(\mathbf{rpk}^{(l)}, (-sk_i) \cdot \mathbf{g_1})$
(7) For each integer k from $l-2$ down to 0
$K^k \leftarrow eK^{(k+1)} \cdot e(\mathbf{tpk}^{(k+1)}, -\mathbf{H_2}(K^{(k+1)}))$
$rK^k \leftarrow reK^{(k+1)} \cdot e(\mathbf{rpk}^{(k+1)}, -\mathbf{H_2}(rK^{(k+1)}) - \mathbf{H_2}(K^{(k+1)}))$
(8) $m \leftarrow em' \cdot e(\mathbf{epk}, -\mathbf{H_2}(K^0) - \mathbf{H_2}(rK^0))$

Finally, verify SHA256$(epk||m) = ah$

The *random* elements in each transform were introduced to the original algorithms by Cai and Liu to resolve a problem they called *proxy bypass*. Without those values, when a user decrypts a transformed message, she recovers the transformation value K from each of the transform keys that were applied to the message. This means, for example, that if Alice transformed a message to Bob, Bob transformed the message to Carol, and Carol transformed the message to Eve, when Eve decrypted the message, she would have the K values from all of the transform keys, and if she subsequently intercepted a message that was transformed from Alice to Bob, she could use the K from that transform key to decrypt the message. The introduction of the random elements in the transformation process does not prevent this, but the K values by themselves are not sufficient to decrypt the transformed message.

5 ENHANCED REVOCATION

A significant benefit of the orthogonal access control system as described is the ability to revoke access from group members with minimal overhead and impact. It is not necessary to decrypt any documents that were shared with the group, generate new keys, and encrypt the documents again; removal of the transform key between the group and the member effectively revokes access.

One potential weakness in this scheme is that a user who has been made an administrator of a group has access to the private key for the group. Even if that user's group access is subsequently revoked and she is removed as an administrator of the group, she could retain a copy of the group private key to directly decrypt any data encrypted to the group, without requiring a transform or assistance from the proxy. Consequently, the group key pair would need to rotate and all files encrypted to that group would need to be decrypted and re-encrypted to the replacement key pair whenever an administrator leaves. This has a negative impact on scalability.

We propose a solution that addresses this weakness and achieves the following:

(1) A group private key should only be able to generate transform keys; it should not be able to decrypt data.
(2) The transform keys generated with a group private key should not be able to transform data outside of the proxy.

To achieve this, we use a multi-party computation to augment the group's public key and to perform a corresponding augmentation of transform keys. The client creates a partial key, and the semi-trusted proxy computes another part of the key and combines them. This operation does not compromise the non-transitivity or collusion-safety of the PRE scheme.

When a group is created, a key pair is generated for that group, and the public key is sent to the proxy. On receipt of the public key, the proxy executes the following algorithm:

AugmentPublicKey$(params, \mathbf{pk}_c) \rightarrow (sk_p, \mathbf{pk}_{aug})$:

(1) *proxy key pair* $(\mathbf{pk}_p, sk_p) \leftarrow$ KeyGen
(2) *augmented public key* $\mathbf{pk}_{aug} \leftarrow \mathbf{pk}_c + \mathbf{pk}_p$

where the addition of public keys is simply addition of two points on the elliptic curve. The resulting augmented public key is stored and distributed as the group's public key, and all data encrypted to the group is encrypted to this augmented key. The proxy retains the group private key that it generated, sk_p, and keeps it secure.

Neither party in the computation (client or proxy) is able to determine the other party's secret key, even though each knows its own secret key, the component public keys, and the resulting augmented public key. Since $\mathbf{pk}_{aug} = sk_c \cdot \mathbf{g} + sk_p \cdot \mathbf{g}$, either party can recover the other party's public key by subtracting its own public key from the augmented public key. But recovering the other party's secret key still requires solving the elliptic curve discrete logarithm problem (ECDLP).

After augmentation, any data that is encrypted to the group's public key can no longer be decrypted by the group's private key. Decryption requires possession of both the client and the proxy secret keys.

In order to allow decryption of a message on the user's device without sharing the proxy's secret key, we also augment each transform key that is created from the group to a user. The transform key that is generated on the client transforms from an unaugmented group public key to a user's key, so we augment the transform key using the same secret key the proxy generated for the group.

On adding a user to a group, when the proxy receives the transform key from the client, it executes the following algorithm:

```
AugmentTransformKey($params, tk_c, sk_p$) $\longrightarrow tk_{aug}$:
  Augment a transform key from a group to a user ($tk_c$), given the
  proxy secret key that was used to augment the group's public
  key ($sk_p$).
    (1) Parse $tk_c$ into $(\mathbf{tpk}, eK, spk_{tk}, sig_{tk}, \mathbf{tep})$
    (2) Verify $\mathcal{V}(\mathbf{tpk}||eK||spk_{tk}, sig_{tk}, spk_{tk})$
    (3) augmented transform point $\mathbf{tep'} \leftarrow \mathbf{tep} + -sk_p \cdot \mathbf{g_1}$
    (4) $tk_{aug} \leftarrow (\mathbf{tpk}, ek, spk_{tk}, sig_{tk}, \mathbf{tep'})$
```

This augmentation of the transform key is performed before the transform key is stored on the proxy. The augmentation process does not give the proxy access to any additional information about the group private key, so it does not allow the proxy to generate new transform keys or decrypt files. But group administrators can only generate partial transform keys, which cannot be used to transform data to any user without cooperation by the proxy.

5.1 Users and Devices

We perform the same augmentation of user public keys and the transform keys between user and device. In this way, someone who captures the user's private key on a device cannot use that key to decrypt any data that was encrypted to the user. Device public keys are not augmented; this allows the device private key to decrypt data that is encrypted to the device public key without additional information. However, the system never directly encrypts data to a device; it only transforms data encrypted to a group or a user to the device. Thus, revocation of device access, which deletes the transform key from the device to the user, renders the device private key useless to decrypt data.

When a device requests a document that was encrypted to the user's augmented public key, the factor of $-sk_p$ in the augmented transform encryption point cancels out the factor of sk_p in the augmented public key, and the device is able to decrypt the message. The appendix shows the cancellation in detail.

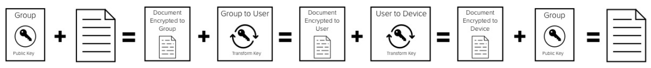

Figure 5: Message Transfer from Group to Device

If a device requests a document that was encrypted to a group, the first transform converts the encrypted data from the group's augmented key to the user's augmented key, cancelling out the proxy's key for the group. The second transform converts from the user's augmented key to the device's public key, cancelling out the proxy's key for the user and producing a ciphertext that the device can decrypt with its secret key.

6 CONCLUSIONS

We have presented a secure data sharing system that implements cryptographically backed access control built on multi-hop, unidirectional, non-interative, and collusion safe proxy re-encryption. The system has the advantageous property of orthogonal access control whereby data can be encrypted to a group and, asynchronously and without accessing the encrypted data, the administrator for that group can determine which users are able to decrypt the data. Similarly, a user can determine what devices can decrypt data that is intended for the user. The encrypted data can be stored anywhere.

Revocation: Access to a group's data by a user or to a user's data by a device is revoked by removing the corresponding transform key from the proxy. It is not necessary to decrypt previously encrypted data, choose a new key, and encrypt with the new key when access is revoked, since the data was never encrypted directly to the device that will retrieve the data.

Additionally, we have introduced a multi-party computation to augment the group and user public keys in the system. This augmentation enhances the revocation of access to data. Even if a group administrator retains a group's secret key after her access is revoked, or if a device retains the user's secret key after its access is revoked, that secret key cannot be used to decrypt any of the group's or user's data without access to a partner secret key that is held by the proxy.

Semi-Trusted Proxy: Because we use end-to-end encryption to enforce access control, the proxy that stores and distributes the encrypted information never has access to the keys to decrypt the data. Users do not need to trust the proxy to keep data secure. Users do need to trust that when they revoke access, the proxy will delete transform keys when instructed to do so. But the proxy can never, by itself, decrypt data or grant access to data to a user.

PRE Choices: We show a specific PRE scheme that meets our criteria and has been previously proven chosen ciphertext attack (CCA) secure. We also propose changes to the signing portion of the scheme for practical purposes that strengthen verifiability and improve performance.

Implications: We believe that cryptographically backed security, and specifically end-to-end encryption, are crucial components of future cloud computing environments. Data is everywhere and perimeters around cloud servers are imperfect. Further, it is impossible to build software systems of reasonable complexity without bugs, some of which can likely be exploited.

By keeping data encrypted through its lifecycle, and by pushing keys off of the storage server and to the points of use, the impact of a server compromise are drastically reduced. Even if a web application is compromised, it is handling encrypted data that can only be unlocked by an authorized user with the cooperation of the proxy. End-to-end encryption that can handle large groups of users at cloud scale is the future of cloud security.

The core library that implements the proxy re-encryption algorithms we use for the system is available as free and open source software - look on GitHub for the IronCoreLabs/recrypt repository.

REFERENCES

[1] G. Ateniese, K. Fu, M. Green, and S. Hohenberger. 2006. Improved Proxy Re-encryption Schemes with Applications to Secure Distributed Storage. *ACM Transactions on Information and System Security (TISSEC)* 9, 1 (2006), 1–30.
[2] M. Blaze, G. Bleumer, and M. Strauss. 1998. Divertible protocols and atomic proxy cryptography. In *EUROCRYPT*. Springer-Verlag, 127–144.
[3] Y. Cai and X. Liu. 2014. A Multi-use CCA-secure Proxy Re-encryption Scheme. *IEEE 12th International Conference on Dependable, Autonomic, and Secure Computing* 7 (2014).
[4] R. Canetti and S. Hohenberger. 2007. Chosen-ciphertext Secure Proxy Re-encryption. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS)*. ACM, New York, NY, USA, 185–194. https://doi.org/10.1145/1315245.1315269
[5] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, and J. Molina. 2009. Controlling data in the cloud: outsourcing computation without outsourcing control. In *Proceedings of the ACM Workshop on Cloud Computing Security*. ACM, 85–90.
[6] Q. Liu, G. Wang, and J. Wu. 2014. Time-based proxy re- encryption scheme for secure data sharing in a cloud environment. In *Information Sciences*, Vol. 258. Elsevier, 355–370.
[7] Z. Qin, H. Xiong, S. Wu, and J. Batamuliza. [n. d.]. A Survey of Proxy Re-Encryption for Secure Data Sharing in Cloud Computing. In *IEEE Transactions on Services Computing*, Vol. PP.
[8] G. Wang, Q. Liu, and J. Wu. 2010. Hierarchical attribute-based encryption for fine-grained access control in cloud storage services. In *Proceedings of the ACM Conference on Computer and Communications Security*. IEEE, 735–737.
[9] H. Wang and Z. Cao. 2009. A Fully Secure Unidirectional and Multi-use Proxy Re-encryption Scheme. *ACM CCS Poster Session* (2009).
[10] H. Xiong, X. Zhang, D. Yao, and X. Wu. 2012. Towards End-to-End Secure Content Storage and Delivery with Public Cloud. In *Proceedings of the second ACM conference on Data and Application Security and Privacy (CODASPY'12)*. 257–266.
[11] L. Xu, X. Wu, and X. Zhang. 2012. CL-PRE: A certificateless proxy re-encryption scheme for secure data sharing with public cloud. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*. ACM, New York, NY, USA, 87–88.
[12] S. Yu, C. Wang, K. Ren, and W. Lou. [n. d.]. Achieving secure, scalable, and fine-grained data access control in cloud computing. In *Proceedings of the IEEE International Conference on Computer Communications*.
[13] J. Zhang and X. A. Wang. 2013. On the Security of Two Multi-use CCA-secure Proxy Re-encryption Schemes. *Int. J. Intelligent Information and Database Systems* 7, 5 (2013), 422–440.

7 APPENDIX

7.1 Correctness of PRE algorithms

These algorithms rely on the bilinear properties of the pairing function. As a reminder, bilinearity requires that

$$\forall a, b \in \mathbb{F}_p{}^*, \forall P \in \mathbb{G}_1, \forall Q \in \mathbb{G}_2 : e(aP, bQ) = e(P, Q)^{ab}$$

$$\forall R, S \in \mathbb{G}_1, T \in \mathbb{G}_2, e(R + S, T) = e(R, T) \cdot e(S, T)$$

For the simple encryption case without transformation, we know that in Encrypt,

$$\begin{aligned}\mathbf{pk}_j &= sk_j \cdot \mathbf{g}\\ \mathbf{epk} &= esk \cdot \mathbf{g}\\ em &= m \cdot e(\mathbf{pk}_j, \mathbf{g_1})^{esk}\end{aligned}$$

and that in Decrypt,

$$m = em \cdot e(\mathbf{epk}, (-sk_j) \cdot \mathbf{g_1})$$

Rewriting the decrypt expression, we get

$$\begin{aligned}m = m \cdot e(\mathbf{pk}_j, \mathbf{g_1})^{esk} \cdot e(\mathbf{epk}, (-sk_i) \cdot \mathbf{g_1}) &=\\ m \cdot e(sk_j \cdot \mathbf{g}, \mathbf{g_1})^{esk} \cdot e(esk \cdot \mathbf{g}, (-sk_j) \cdot \mathbf{g_1}) &=\\ m \cdot (esk \cdot \mathbf{g}, \mathbf{g_1})^{sk_j} \cdot e(esk \cdot \mathbf{g}, \mathbf{g_1})^{-sk_j} &=\\ m \cdot 1 &=\\ m&\end{aligned}$$

For a simple single-hop transformation from Alice to Bob, we know that

$$\begin{aligned}eK &= K \cdot e(\mathbf{pk}_B, \mathbf{g_1})^{tsk}\\ \mathbf{tep} &= \mathbf{H_2}(K) + (-sk_A) \cdot \mathbf{g_1}\\ reK &= rK \cdot e(\mathbf{pk}_B, \mathbf{g_1})^{rsk}\\ em' &= em \cdot e(\mathbf{epk}, \mathbf{H_2}(rK) + \mathbf{tep}) =\\ m \cdot e(\mathbf{pk}_A, \mathbf{g_1})^{esk} &\cdot e(\mathbf{epk}, \mathbf{H_2}(rK) + \mathbf{tep}) =\\ m \cdot e(\mathbf{pk}_A, \mathbf{g_1})^{esk} &\cdot e(\mathbf{epk}, \mathbf{H_2}(rK) + \mathbf{H_2}(K) + (-sk_A) \cdot \mathbf{g_1})\end{aligned}$$

and in Decrypt,

$$\begin{aligned}K = eK \cdot e(\mathbf{tpk}, (-sk_B) \cdot \mathbf{g_1}) &=\\ K \cdot e(\mathbf{pk}_B, \mathbf{g_1})^{tsk} \cdot e(\mathbf{tpk}, (-sk_B) \cdot \mathbf{g_1}) &=\\ K \cdot e(sk_B \cdot \mathbf{g} \cdot tsk, \mathbf{g_1}) \cdot e(\mathbf{tpk}, (-sk_B) \cdot \mathbf{g_1}) &=\\ K \cdot e(\mathbf{tpk}, \mathbf{g_1})^{sk_B} \cdot e(\mathbf{tpk}, \mathbf{g_1})^{-sk_B} &=\\ K&\\ rK = reK \cdot e(\mathbf{rpk}, (-sk_B) \cdot \mathbf{g_1}) &=\\ rK \cdot e(\mathbf{pk}_B, \mathbf{g_1})^{rsk} \cdot e(\mathbf{rpk}, (-sk_B) \cdot \mathbf{g_1}) &=\\ rK \cdot e(\mathbf{rpk}, \mathbf{g_1})^{sk_B} \cdot e(\mathbf{rpk}, \mathbf{g_1})^{-sk_B} &=\\ rK&\\ m = em' \cdot e(\mathbf{epk}, -\mathbf{H_2}(K) - \mathbf{H_2}(rK)) &=\\ m \cdot e(\mathbf{pk}_A, \mathbf{g_1})^{esk} \cdot e(\mathbf{epk}, \mathbf{H_2}(rK) + \mathbf{H_2}(K) &+\\ (-sk_A) \cdot \mathbf{g_1}) \cdot e(\mathbf{epk}, -\mathbf{H_2}(K) - \mathbf{H_2}(rK)) &=\\ m \cdot e(\mathbf{epk}, sk_A \cdot \mathbf{g_1}) \cdot e(\mathbf{epk}, \mathbf{H_2}(rK) + \mathbf{H_2}(K) &+\\ (-sk_A) \cdot \mathbf{g_1}) \cdot e(\mathbf{epk}, -\mathbf{H_2}(K) - \mathbf{H_2}(rK)) &=\\ m \cdot e(\mathbf{epk}, sk_A \cdot \mathbf{g_1} + \mathbf{H_2}(rK) + \mathbf{H_2}(K) &+\\ (-sk_A) \cdot \mathbf{g_1}) \cdot e(\mathbf{epk}, \mathbf{H_2}(K) + \mathbf{H_2}(rK))^{-1} &=\\ m \cdot e(\mathbf{epk}, \mathbf{H_2}(K) + \mathbf{H_2}(rK)) &\cdot\\ e(\mathbf{epk}, \mathbf{H_2}(K) + \mathbf{H_2}(rK))^{-1} &=\\ m&\end{aligned}$$

For a two-hop transformation from Alice to Bob, then from Bob to Carol, the process is the same, with the modification of *eK* and *reK* in the first transform block and the addition of the second transform block. In Transform,

$$eK_{A\to B} = K_{A\to B} \cdot e(\mathbf{pk}_B, \mathbf{g_1})^{tsk_{A\to B}}$$
$$reK^{(1)} = rK^{(1)} \cdot e(\mathbf{pk}_B, \mathbf{g_1})^{rsk^{(1)}}$$
$$eK'^{(1)} = eK^{(1)} \cdot e(\mathbf{tpk}_{A\to B}, \mathbf{H_2}(rK^{(2)}) + \mathbf{tep}_{B\to C})$$
$$reK'^{(1)} = reK^{(1)} \cdot e(\mathbf{rpk}^{(1)}, \mathbf{H_2}(rK^{(2)}) + \mathbf{tep}_{B\to C})$$
$$eK_{B\to C} = K_{B\to C} \cdot e(\mathbf{pk}_C, \mathbf{g_1})^{tsk_{B\to C}}$$
$$reK^{(2)} = rK^{(2)} \cdot e(\mathbf{pk}_c, \mathbf{g_1})^{rsk^{(2)}}$$
$$em' = em \cdot e(\mathbf{epk}, \mathbf{H_2}(rK) + \mathbf{tep}) =$$
$$m \cdot e(\mathbf{pk}_A, \mathbf{g_1})^{esk} \cdot e(\mathbf{epk}, \mathbf{H_2}(rK) + \mathbf{tep}) =$$
$$m \cdot e(\mathbf{pk}_A, \mathbf{g_1})^{esk} \cdot e(\mathbf{epk}, \mathbf{H_2}(rK) + \mathbf{H_2}(K) + (-sk_A) \cdot \mathbf{g_1})$$

In Decrypt,

$$dK^{(1)} = eK_{B\to C} \cdot e(\mathbf{tpk}^{(2)}, (-sk_C) \cdot \mathbf{g_1}) =$$
$$K_{B\to C} \cdot e(\mathbf{pk}_C, \mathbf{g_1})^{tsk_{B\to C}} \cdot e(\mathbf{tpk}^{(2)}, (-sk_C) \cdot \mathbf{g_1}) =$$
$$K_{B\to C} \cdot e(\mathbf{tpk}_{B\to C}, \mathbf{g_1})^{sk_C} \cdot e(\mathbf{tpk}_{B\to C}, \mathbf{g_1})^{-sk_C} = K_{B\to C}$$
$$drK^{(1)} = reK^{(2)} \cdot e(\mathbf{rpk}^{(2)}, (-sk_C) \cdot \mathbf{g_1}) =$$
$$rK^{(2)} \cdot e(\mathbf{pk}_C, \mathbf{g_1})^{rsk^{(2)}} \cdot e(\mathbf{rpk}^{(2)}, (-sk_C) \cdot \mathbf{g_1}) =$$
$$rK^{(2)} \cdot e(\mathbf{rpk}^{(2)}, \mathbf{g_1})^{sk_C} \cdot e(\mathbf{rpk}^{(2)}, \mathbf{g_1})^{-sk_C} = rK^{(2)}$$
$$dK^{(0)} = eK^{(1)} \cdot e(\mathbf{tpk}^{(1)}, (-sk_B) \cdot \mathbf{g_1}) =$$
$$K_{A\to B} \cdot e(\mathbf{pk}_B, \mathbf{g_1})^{tsk_{A\to B}} \cdot e(\mathbf{tpk}^{(1)}, (-sk_B) \cdot \mathbf{g_1}) =$$
$$K_{A\to B} \cdot e(\mathbf{tpk}_{A\to B}, \mathbf{g_1})^{sk_B} \cdot e(\mathbf{tpk}_{A\to B}, \mathbf{g_1})^{-sk_B} = K_{A\to B}$$
$$drK^{(0)} = reK^{(1)} \cdot e(\mathbf{rpk}^{(1)}, (-sk_B) \cdot \mathbf{g_1}) =$$
$$rK^{(1)} \cdot e(\mathbf{pk}_B, \mathbf{g_1})^{rsk^{(1)}} \cdot e(\mathbf{rpk}^{(1)}, (-sk_B) \cdot \mathbf{g_1}) =$$
$$rK^{(1)} \cdot e(\mathbf{rpk}^{(1)}, \mathbf{g_1})^{sk_B} \cdot e(\mathbf{rpk}^{(1)}, \mathbf{g_1})^{-sk_B} = rK^{(1)}$$
$$m = em' \cdot e(\mathbf{epk}, -\mathbf{H_2}(dK^{(0)}) - \mathbf{H_2}(drK^{(0)})) =$$
$$m \cdot e(\mathbf{pk}_A, \mathbf{g_1})^{esk} \cdot e(\mathbf{epk}, \mathbf{H_2}(rK^{(1)}) + \mathbf{H_2}(K_{A\to B}) + (-sk_A) \cdot \mathbf{g_1}) \cdot$$
$$e(\mathbf{epk}, -\mathbf{H_2}(dK^{(0)}) - \mathbf{H_2}(drK^{(0)})) =$$
$$m \cdot e(\mathbf{epk}, \mathbf{g_1})^{sk_A} \cdot e(\mathbf{epk}, \mathbf{H_2}(rK^{(1)}) + \mathbf{H_2}(K) + (-sk_A) \cdot \mathbf{g_1}) \cdot e(\mathbf{epk}, -\mathbf{H_2}(K_{A\to B}) - \mathbf{H_2}(rK^{(1)})) = m$$

7.2 Correctness with key augmentation

Suppose Alice generates a user key pair $(\mathbf{pk}_A, sk_A)$, a device key pair $(\mathbf{pk}_1, sk_1)$, and a transform key $tk_{A\to 1)}$, and that the proxy generates the augmentation key pair $(\mathbf{ppk}_A, psk_A)$. Alice's augmented public key is

$$\mathbf{pk}_{Aaug} \leftarrow \mathbf{pk}_A + \mathbf{ppk}_A$$

The augmented transform key's transform encryption point is

$$\mathbf{tep}_{aug} \leftarrow \mathbf{tep} - psk_A \cdot \mathbf{g_1}$$

When a document m is encrypted to Alice's augmented public key,

$$em = m \cdot e(\mathbf{pk}_{Aaug}, \mathbf{g_1})^{esk} =$$
$$m \cdot e(\mathbf{pk}_A + \mathbf{ppk}_A, \mathbf{g_1})^{esk}$$

If this is transformed to device 1,

$$em' = em \cdot e(\mathbf{epk}, \mathbf{tep}_{aug}) =$$
$$em \cdot e(\mathbf{epk}, \mathbf{tep} - psk_A \cdot \mathbf{g_1}) =$$
$$m \cdot e(\mathbf{pk}_A + \mathbf{ppk}_A, \mathbf{g_1})^{esk} \cdot e(\mathbf{epk}, \mathbf{tep} - psk_A \cdot \mathbf{g_1}) =$$
$$m \cdot e(\mathbf{pk}_A, \mathbf{g_1})^{esk} \cdot e(\mathbf{ppk}_A, \mathbf{g_1})^{esk} \cdot$$
$$e(\mathbf{epk}, \mathbf{tep}) \cdot e(\mathbf{epk}, \mathbf{g_1})^{-psk_A} =$$
$$m \cdot e(\mathbf{pk}_A, \mathbf{g_1})^{esk} \cdot e(\mathbf{epk}, \mathbf{tep}) \cdot$$
$$e(\mathbf{pk}, \mathbf{g_1})^{psk_A} \cdot e(\mathbf{epk}, \mathbf{g_1})^{-psk_A} =$$
$$m \cdot e(\mathbf{epk}, \mathbf{g_1})^{sk_A} \cdot e(\mathbf{epk}, \mathbf{tep})$$

In Decrypt,

$$m = em' \cdot e(\mathbf{epk}, -\mathbf{H_2}(K)) =$$
$$m \cdot e(\mathbf{epk}, \mathbf{g_1})^{sk_A} \cdot e(\mathbf{epk}, \mathbf{tep}) \cdot e(\mathbf{epk}, -\mathbf{H_2}(K)) =$$
$$m \cdot e(\mathbf{epk}, \mathbf{g_1})^{sk_A} \cdot e(\mathbf{epk}, \mathbf{H_2}(K) + (-sk_A) \cdot \mathbf{g_1}) \cdot$$
$$e(\mathbf{epk}, -\mathbf{H_2}(K)) =$$
$$m \cdot e(\mathbf{epk}, \mathbf{g_1})^{sk_A} \cdot e(\mathbf{epk}, \mathbf{g_1})^{-sk_A} \cdot$$
$$e(\mathbf{epk}, \mathbf{H_2}(K)) \cdot e(\mathbf{epk}, \mathbf{H_2}(K))^{-1} = m$$

The cancellations for transforms from a group's augmented public key to a user's augmented public key to a device's public key are similar.