Decentralizing Public Key Infrastructures with ClaimChains

Bogdan Kulynych*, Marios Isaakidis†, Carmela Troncoso*, and George Danezis†
*IMDEA Software Institute Email: {bogdan.kulynych, carmela.troncoso}@imdea.org
†University College London Email: m.isaakidis@cs.ucl.ac.uk, g.danezis@ucl.ac.uk

***Abstract*—We present *ClaimChains*, a cryptographic construction useful for storing claims regarding users' key material and beliefs about the state of other users in a decentralized system. We use ClaimChains to build a decentralized Public Key Infrastructure (PKI). ClaimChains maintain high integrity through the use of authenticated data structures, namely hash chains and Merkle trees, and ensure authenticity and non-repudiation through the use of digital signatures. We introduce the concept of *cross-referencing* of ClaimChains to efficiently and verifiably vouch for the state of other users in a decentralized system. ClaimChains use cryptographic protections to ensure the privacy of claims, i.e., to guarantee that they can only be read by the authorized users, and that ClaimChain owners can not equivocate about the state of other users. We discuss how ClaimChains support different degrees of PKI decentralization, to trade off key availability for privacy. We show that ClaimChains provide the sought security and privacy properties, and demonstrate that they have very reasonable computation and memory requirements using a prototype implementation. We evaluate the effectiveness of key propagation using a real email dataset in a fully decentralized setting, which offers the best privacy properties. Our results suggest that a high level of privacy comes at the cost of small coverage in terms of key distribution.**

1. Introduction

End-to-end content confidentiality commonly relies on public-key cryptography, e.g., for end-to-end encryption or secure key agreements. Public Key Infrastructures (PKIs) ensure that users can learn each others' encryption keys with high assurance. PKIs provide means to record, distribute and revoke bindings between users and their public keys.

PKIs can be implemented in a number of ways. The most popular implementation is based on centralized certification authorities, e.g., SKS Keyserver[1], or Internet X.509 Public Key Infrastructure [1]. These provide good availability and ease the key management operations (like revocations or updates). On the negative side, centralized PKI providers become a single point of failure with respect to the integrity of keys and their bindings to identities. Recent proposals, however, attempt to mitigate this problems via transparency logs [2], rendering *equivocation* about users' public key material detectable and accountable. Another negative aspect is that PKI providers are placed in a privileged position to conduct surveillance of interactions of their users.

We present *ClaimChains*, a novel construction based on hash chains, that we use to build a PKI that aims to alleviate the problems mentioned above: centralization of trust, and privacy of contacts and interactions.

A ClaimChain-based PKI avoids the issue of centralization of trust inherent to traditional PKI designs by operating in a decentralized manner, i.e., with no reliance on trusted entities. In other words, the high-integrity and authenticity properties are kept *without reference* to such trusted providers, as opposed to current proposals that rely on a trusted entity (e.g., identity-key bindings signed by a CONIKS provider [2]). A ClaimChain consists of blocks, each block containing all the information necessary to represent the claims of one user about her own keys, and her beliefs about other users' ClaimChains (which we call *cross references*). Users can access each other's ClaimChains and combine these beliefs to support the provision of evidence about the binding between identities and keys. We note that the design of ClaimChains could handle any generic claims about user beliefs. However, for the sake of simplicity, in this document we restrict ourselves to the context of key management for messaging applications.

A core goal of our design is to support full decentralization. Yet, ClaimChains are flexible in terms of deployment. At one extreme, ClaimChains may reside locally on users' devices, with users being responsible for providing access to their claims to interested parties. This can be implemented, for instance, using email or chat messages as vehicle to transport ClaimChains. At the other extreme, ClaimChains can be stored in an online service that provides users with an API to interact with them. Regardless of the operation mode, the core security properties of ClaimChains are ensured by cryptographic mechanisms, thus do not require the supporting infrastructure to be trusted.

ClaimChains provide means to control who can access each claim via cryptographic capabilities: using access tokens, users can indicate which other users are

1. https://sks-keyservers.net

allowed to read which claims. ClaimChains not only ensure that non-authorized users cannot access claims, but also ensure unlinkability across blocks through the use of nonces to randomize encryption of claims. This in turn prevents the study of patterns about how users' beliefs change over time.

As an undesirable side effect of privacy of claims, users may provide conflicting versions of beliefs to different readers (equivocate), which may disrupt efficient resolution of identity-key bindings in a decentralized PKI. ClaimChains provide mechanisms to reliably and efficiently prevent equivocation,s while minimizing the leakage of user's friendship networks – a problem that plagues decentralized PKI systems such as the PGP Web of Trust [3].

The rest of this document is organized as follows. We first review related work in Section 2. Then, we present the ClaimChain design in Section 3, and describe how it can be used to implement a decentralized PKI in Section 4. We evaluate our implementation of ClaimChains, as well as formally describe the security and privacy properties they provide in Section 5; and in Section 6 we evaluate a ClaimChain-based PKI in a fully decentralized setting. Finally, we discuss integration aspects of ClaimChains in Section 7, and conclude in Section 8.

2. Related work

We review deployed and academic PKI systems focusing on guaranteeing authenticity of bindings between email addresses and keys.

Centralized PKI Infrastructure. The default PKI used by mail agents for submitting and retrieving PGP public keys is the SKS Keyserver pool. Anyone can upload bindings to an SKS keyserver, which may end up holding bindings for the same email to different public keys. Thus, there are no guarantees about which of the bindings, if any at all, are authentic.

The next generation of PGP PKI systems, e.g. PGP Global Directory[2], or Mailvelope[3], introduce an access control mechanism based on proving ownership of the email address involved in the binding, by means of a confirmation link send to that address. Bindings get published only after they have been confirmed and only one binding per email address is kept. This model does not protect users from malicious key servers who advertise fake bindings, malicious email providers who update the binding of a user without her consent, or adversaries who could intercept the verification emails via network traffic.

The Nyms Identity Directory[4] complements email ownership verification with the use of "Trusted Notaries", that also sign bindings, and a "Network Perspective auditing mechanism", which cross-checks the bindings across providers to prevent them from equivocating.

All the above systems can be complemented by CONIKS [2], which can be seen as a modification to Certificate Transparency (CT) [4] applied to PGP PKI systems. CONIKS equips providers with the means to create an auditable log of their users' public keys. Such log ensures the consistency of an email-key binding throughout time. Effectively, equivocations about a user's public key by their provider are easily detectable and accountable. KAS [5] improves performance of CONIKS, by storing key information separately for each user, yet it is still works in the setting of centralized available providers.

The building blocks of CONIKS are essentially the same as the ones we use in ClaimChains: verifiable random functions (VRF), and Merkle trees to ensure privacy, high integrity, and non-equivocation. However, the challenges caused by moving to a decentralized scenario directly affect the way in which such primitives are combined to provide the same security properties.

Kokoris-Kogias et al. [6] propose a system in which users rely on a publicly reachable set of trusted servers that form a cothority [7] (i.e., a decentralized set of authorities that collectively sign statements) that provides blockchain management as a service. Users manage their keys using this blockchain to publish their status, i.e., update, add, or remove keys and bindings. In turn, this blockchain can be accessed by others to validate bindings.

An example of a functioning decentralized PKI is Blockstack [8]. Blockstack uses a global system-wide Namecoin blockchain as a high-integrity store, with Bitcoin proof-of-work consensus mechanism [9] used by nodes to agree on the latest state of the system. Such approach is feasible as demonstrated by widespread adoption of Bitcoin and Bitcoin-like cryptocurrencies, yet comes with a number of drawbacks. Since the users need to agree on the state of the whole system at each point in time, the latency, storage and bandwidth cost increase significantly with the number of users. A ClaimChain-based PKI, on the other hand, allows not to use a shared global state that has to be agreed upon by majority of nodes, avoiding these additional computational and bandwidth costs. Instead, all users maintain hash chains containing their own claims and local views of other users' states. In such setting, local consensuses about the partial state of the system arise within cliques of communicating users, rather than a single global consensus about the state of all participants. This allows for faster propagation of updates, and ensures that only relevant portion of the state of the nodes is being considered and transmitted by users.

Social validation. To validate the authenticity of cryptographic keys, a social trust mechanism may be employed. The paradigmatic example of such mechanism is the PGP Web of Trust (WoT) [3]. Conceptually, the

2. https://keyserver.pgp.com
3. https://keys.mailvelope.com
4. https://nyms.io

WoT is similar to the endorsements by Trusted Notaries in Nyms, but in this case *any* PGP key owner can vouch for the validity of a binding. To decide whether a binding is authentic, users consider vouches of their friends, and recursively those of the friends of their friends, and so on. The current implementation of the WoT raises serious privacy concerns, since the use of vouching leaks the social relationships among users. Moreover, authenticating the WoT is difficult, and this fact facilitates equivocation attacks.

An alternative to alleviate these problems is Keybase[5]. First, to hinder equivocation attacks based on taking over the social network, Keybase users publish statements about their binding in various social media or website domains they control that others can use as trust roots when validating the authenticity of the advertised keys. Second, Keybase signature mechanisms ensure that signatures are not only public, but also ordered, i.e., cannot be rolled back. In addition, the Keybase server maintains a Merkle tree structure that, similarly to CONIKS, prevents the server from performing equivocation attacks. To further hinder the attack the root of this tree is published as a Bitcoin transaction.

Recently, a system for opportunistic in-band decentralized key distribution, called Autocrypt[6], was proposed. Autocrypt-enabled email clients embed user's key material in the messages, and in the latest version[7], also include the key material of user's contacts. This system is compatible with ClaimChains, and moreover, using ClaimChains should fix some of its existing security problems, like possible man-in-the-middle attacks. We use Autocrypt model of key distribution when designing the fully decentralized mode of operation for a ClaimChain-based PKI.

3. ClaimChain design

ClaimChains represent repositories of claims that users make about themselves or other users. A user may have one or multiple such ClaimChains, for example, associated with multiple devices or multiple pseudonyms. To account for beliefs evolving over time, ClaimChains are implemented as cryptographic chains of *blocks* of claims. Each block in the chain contains enough information to authenticate past blocks as being part of the chain, as well as validate future blocks as being valid updates. Thus, a user with access to a block of a chain that they believe is authoritative (i.e. considered to provide correct information), may both audit past states of the chain, and authenticate the validity of newer blocks.

Each block of a ClaimChain includes all claims that its owner endorses at the point in time when the block is generated, and all data needed to authenticate the chain. We deliberately choose to replicate all claims and full state to keep the design of access control and non-repudiation simple. This choice, as we show in Section 5.2, has very reasonable overhead both in terms of storage and authentication. Alternative designs, in which only updates are included in each block, proved elusive optimizations, yet might be a good avenue for future work.

A user stores three types of information in a ClaimChain:

Own metadata. This information may include the owner identity, such as her screen name, real name, email or chat identifiers; as well as cryptographic material needed for applications, like verification keys to support digital signatures, or encryption keys to support confidential messaging. Claims about a user's own metadata are initially self asserted, and gain credibility for other users by being cross-referenced, i.e., "certified", in other users' chains.

Claims about other users. The simplest claim about another user is endorsing other user's ClaimChain as being authoritative, i.e. indicate the belief that its metadata binding identifiers to keys are correct. Other kinds of claims can be, for instance, links to other ClaimChains of the same owner, or claims about compromise of other ClaimChains.

Cryptographic data. ClaimChains must contain enough information to authenticate all past states, as well as future updates of the repository. For this purpose they include digital signatures, and corresponding signing public keys. In order to enable efficient operations without the need for another party to have full visibility of all claims in the chain, we augment ClaimChains with quick cryptographic links to past states, as well as roots of high-integrity data structures such as Merkle trees. Other key material needed for ensuring privacy and non-equivocatoin is also included, as described in detail below.

We envision that users will create one ClaimChain for each of their own identities for which they want to make claims about. When a user wants to establish the validity of a claim, she uses other users' ClaimChains to gather evidence about others' beliefs about it. The evidence is processed according to some policy defined by the user performing the validation. Then she can make a decision whether the chain in question is valid and authoritative. We stress that the goal of ClaimChains is to ensure that users can gather evidence in a decentralized and secure manner to be used as input for validation. Establishing the best validation policy for a given use case is out of the scope of this paper.

Threat model & Security properties. ClaimChains must protect against three types of adversaries. First, they must prevent *owners* from *equivocating* other users about the content of claims. Second, it must ensure the *privacy of claims* and the *privacy of contacts* from

5. https://keybase.io
6. https://autocrypt.readthedocs.io/en/latest/
7. https://autocrypt.readthedocs.io/en/latest/level1.html?highlight=gossiping#key-gossip

other users, and the *storage infrastructure* on which ClaimChains are stored. We note that ClaimChains are not designed to offer protection against inferences that can be made from access patterns to the information they store, as we discuss in Section 7.

ClaimChains aim to provide the following security properties, which are defined formally in Section 5.3:

Authenticity: the information stored in a ClaimChain has been input by the owner of the chain.

Integrity: the information stored in a ClaimChain has not been modified since it was added to the chain.

Privacy: only readers authorized by the ClaimChain owner to access a claim at a particular point in time can read the content of that claim, at that point in time.

Non-equivocation: at a particular point in time, a ClaimChain owner cannot provide two authorized readers of a claim with different content for that claim.

As we describe in detail below, we rely on properties of *hash functions* to reduce integrity checks of a large, growing, set of statements about names and keys (claims), to a simple integrity check of a short fixed size sting of bits – the *head* of a *hash chain* or a *Merkle tree*. The ability to summarize all information known to a user about both her own and others' statements in such a short form, enables inexpensive sharing, and replication of identity information in a high-integrity manner. Further, cryptographic primitives, such as *digital signatures* allow for self-authenticated updates, and *encryption* and *verifiable random functions* allow us to implement privacy features at the same time ensuring non-equivocation.

3.1. ClaimChain building blocks

Preliminaries. We review two structures central to the design of ClaimChains:

Non-equivocable Merkle tree. This data structure is composed of two types of nodes:

$$\mathsf{Internal} = (\mathsf{pivot},\ \mathsf{left} : H(\mathsf{Node}),\ \mathsf{right} : H(\mathsf{Node}))$$
$$\mathsf{Leaf} = (\mathsf{key}, \mathsf{value})$$

Each Internal node contains a pivot key, and the invariant of the structure is that any Leaf nodes in the left sub-tree will have keys smaller than the pivot, and any Leaf nodes to the right have keys equal or larger than the pivot. Internal nodes store the hash of the node representing the left and right sub-tree. This effectively creates a Merkle tree in which the hash of the root node is a succinct authenticator committing to the full sub-tree (subject to the security of the hash function).

A proof of inclusion of a key-value pair in the tree involves disclosing the full resolution path of nodes from the root of the tree to the sought leaf. Similarly, a proof of non-inclusion involves disclosing the failing resolution path from the root of the tree to the leaf nodes that are not the sought key-value. We note that, for a single key, only one value is to be stored. Any violation of this invariant may be detected when the proof of inclusion or exclusion are checked – thus the creator of the tree does not need to be trusted to enforce this invariant.

High-integrity skip list. Traditional hash chains only contain the hash of the previous block, requiring linear verification time to authenticate blocks in the past. By including a selection of hashes of past blocks (including the one immediately preceding the block) the cost is reduced to $\mathcal{O}(\log(i-j))$ where i is the index of the latest block, and j is the index of the past block to be verified as belonging to the chain. Concretely, a block includes some subset of hashes F_i to previous blocks $F_i = \{(j, H(B_j)) | j \in J(i)\}$ for a set of indices $J(i)$, such that $\forall j \in J(i) : j < i$. The indexes $J(i)$ are chosen to mirror the structure of a deterministic skip-list [10]:

$$J(i) \equiv \{\forall t \in \mathbb{Z}^*. \quad i - 1 - ((i-1) \mod (2^t))\}.$$

Notionally the indexes $J(i)$ fall on the 'tick marks' of a binary ruler, one per height of tick-mark preceding block index i. For example $J(127) \equiv \{64, 96, 0, 112, 120, 124, 126\}$ and $J(128) \equiv \{96, 64, 0, 112, 120, 124, 126, 127\}$.

Given an index $j < i$, $J(i)$ is guaranteed to contain a a number that is at least $(i-j)/2$ closer to j than i. This ensures that when authenticating block j departing from block i, it is possible to retrieve a past block that is at least half the distance between blocks i and j. This block will also contain past hashes that can be used to recursively authenticate any past block in $\mathcal{O}(\log(i-j))$ jumps.

3.1.1. Block & chain structure. The core building element of a ClaimChain, denoted as $B_i = (X_i, \sigma_i)$, is a *block*: a data structure that consists of some *payload* X_i, along with a public-key digital signature on that payload, $\sigma_i = \mathsf{Sign}_{\mathsf{sk}^t_{\mathsf{SIG}}}(H(X_i))$ for a signing key pair $(\mathsf{pk}^t_{\mathsf{SIG}}, \mathsf{sk}^t_{\mathsf{SIG}})$. The payload contains information that can be used by authorized readers to access claims. This per-block signature enables the authentication of subsequent blocks: a block B_i must have a valid signature under the verification key indicated in the payload of block B_{i-1}. The genesis block of the ClaimChain is 'self-signed' with a key pair designated in the initial payload.

A block in a ClaimChain serves as full snapshot of the owner's *state*, i.e., it is a self-contained commitment to the owners' belief about her own state and state of other people, at a given time. For auditability reasons we link chronological sequences of blocks into hash chains. From a genesis block B_0, we build the chain as follows:

$$B_0 = \left(X_0,\ \sigma_0 = \mathsf{Sign}_{\mathsf{sk}^0_{\mathsf{SIG}}}(H(X_0))\right)$$
$$B_i = \left(X_i,\ \sigma_i = \mathsf{Sign}_{\mathsf{sk}^{i-1}_{\mathsf{SIG}}}(H(X_i))\right), i > 0$$

The block payload X_i has the following fields:

Version. A version number associated with the code to interpret this ClaimChain.

Block index. The sequential number of this block, i.e., its position in the chain. The index of the genesis block is 0.

Timestamp. The unix epoch, at a granularity pre-defined by the version, when this block was created.

Nonce. A fresh cryptographic nonce that is used to 'salt' all cryptographic operations within the block. It ensures that information across blocks is not linkable.

Metadata. The identity and keys associated with this block (see Section 3.1.2).

Block map. Within each block we include the root of a Merkle Tree representing a non-equivocable high-integrity key-value map. This data structure is used to store both the claim map (explained further in Section 3.1.3) and the capabilities map (Section 3.1.4) for the block. This tree has two key properties: i) a key can only be resolved to a single value, and ii) given its root it enables the generation, or verification, of efficient proofs of inclusion or exclusion of specific claims or capabilities.

Pointers to previous blocks. Traditional hash chains and blockchains only contain the hash of the previous block, requiring linear verification time to authenticate blocks in the past. Blocks in a ClaimChain include hashes of blocks beyond the previous one to enable verification to be faster than linear. By including a selection of hashes of past blocks (including the one immediately preceding the block) the cost is reduced to $\mathcal{O}(\log(i-j))$ where i is the index of the latest block, and j is the index of the past block to be verified as belonging to the ClaimChain. The size of the pointers is of length $O(\log i)$, for block index i, and grows slowly as the chain grows.

3.1.2. ClaimChain metadata. Each ClaimChain block, including the genesis (initial) block, contains metadata necessary to identify the owner, read her claims, encrypt data for her, and verify the integrity of the chain. Note that metadata is stored in the clear and can be read by anyone.

We list the kinds of information now:

Identities. For instance, an email address, Twitter handle, Signal number, POTS number, web page, of the owner of the chain. These are used to extract labels (explained in the next section) that characterize this chain, and guide other users to find the appropriate ClaimChain to obtain cryptographic keys when trying to communicate with a partner.

Cryptographic material. This includes the keys that are necessary for the operation of the ClaimChain, and the keys that are useful for applications. The former includes the current signing key of the owner, $\mathsf{pk}_{\mathsf{SIG}}$, that is used to authenticate new blocks of the ClaimChain; the current key to compute verifiable random functions, $\mathsf{pk}_{\mathsf{VRF}}$, used to support non-equivocation as explained below; and a Diffie-Hellman key for key derivation, which we call the ClaimChain encryption key, $\mathsf{pk}_{\mathsf{DH}}$, used to implement private claims.

3.1.3. Claims and the block claim map. Claims within a block are assertive statements that carry information the owner of the chain wants to share at that particular point in time. In the context of implementing a PKI for messaging, claims can be statements about key material owned by the owner of the ClaimChain, key material owned by another person, cross-references of other ClaimChains (i.e., claims about the state of other ClaimChains), and statements about the ClaimChain itself.

Each claim is indexed by a *label* that is a well-known identifier associated with the identity (e.g., person, network end point) that the claim refers to. As an example, a claim may be labelled as 'alice@gmail.com' if it refers to a belief the ClaimChain owner has about Alice's gmail account's key or associated ClaimChain.

Private claims. All claims in a ClaimChain block are encrypted and indexed using a verifiable random function (VRF), under the correspondent public key $\mathsf{pk}_{\mathsf{VRF}}$ published in the ClaimChain metadata, and 'salted' using the nonce published in the same block. At a high-level, consider a claim $\mathsf{claim\_body}$ with label $\mathsf{claim\_label}$, that is to be included in a ClaimChain block with nonce, using $\mathsf{pk}_{\mathsf{VRF}}$. We first derive a key for this claim, by generating a value k using a verifiable random function as $k, \mathsf{proof} = \mathsf{VRF}_{\mathsf{pk}_{\mathsf{VRF}}}(\mathsf{claim\_label} \parallel \mathsf{nonce})$, where proof is needed to for verification of the VRF value. The index l of the claim is derived using this shared secret as $l = H_1(k)$, and an encryption key is derived as $K = H_2(k)$ where H_1 and H_2 are cryptographic hash functions. The body of the claim and proof are then jointly encrypted as: $C = \mathsf{Enc}_K(\mathsf{proof} \parallel \mathsf{claim\_body})$. The tuple (l, c) represents the encrypted claim.

A ClaimChain block includes the list of tuples (l, c) of all claims encoded using block's $\mathsf{pk}_{\mathsf{VRF}}$ and nonce. We consider the l component to be a 'look-up key' of the claim in that block (in the sense of a key-value store), and the c component to be the 'value' of the encrypted claim. The tuples are stored in a data structure that provides a 'map' interface – which we call the *claim map*. We instantiate this structure as a sorted Merkle tree, adapted to provide non-equivocation.

Given a VRF value k for a $\mathsf{claim\_label}$, a tuple (l, c), the block nonce and the public key $\mathsf{pk}_{\mathsf{VRF}}$ associated with the block, anyone may verify that (l, c) is a valid encoding of claim with $\mathsf{claim\_label}$. First, we check the claim look-up key l by re-deriving it using k and H_1. Second, the decryption key K may be derived, and used to decrypt c to recover the proof and $\mathsf{claim\_body}$. Finally, the proof may be used in conjunction with the public key $\mathsf{pk}_{\mathsf{VRF}}$, and the $\mathsf{claim\_label}$, to verify the validity of the given VRF value k. Users must always verify that a tuple (l, c) is valid encoding for a claim label before using the decrypted claim.

This scheme for encoding and decoding of claims offers two distinct security advantages: i) a valid claim only has a unique VRF value k for a given block nonceand $\mathsf{pk}_{\mathsf{VRF}}$, and thus a unique look-up key l. As a result, this key may be used to support non-equivocation for claims relating to this label, as we discuss below; and ii) without knowledge of the VRF value k, the tuple (l, c) leaks no information about the claim label or the claim body – preserving privacy. Such construction preserves both the privacy of claims and of the social graph, i.e., who makes claims about whom. Additionally, we implement cryptographic access control by sharing with some users, and not others, the appropriate VRF values, as we described in the following section.

Public claims. We note that owners may want to also share claims that are readable by anybody with the access to the block. One way to implement this is to simply store the value VRF values in the clear, e.g., in the metadata section of the block.

3.1.4. Access control and the capabilities map. ClaimChains use cryptographic access control to restrict reading access to claims. Additionally, this prevents non-authorized users from inferring that a claim for a specific label is present within a block.

Abstractly, we consider an access control matrix (ACM) [11] where subjects represent potential readers of claims; objects are labels of claims; and the only access right is 'read'. We assume that the owner of a ClaimChain, through an appropriate user interface or other mechanism, specifies the ACM. The objective of our system is then to ensure that only reads allowed by the policy represented by this matrix can be performed – i.e. only users with a 'read' privilege should be allowed to detect that a claim for a label is present in the claim map of a block, and decrypt this claim.

We implement this mechanism using cryptographic controls to regulate access to the VRF values (which serve as access tokens) associated to each label. We assume that for each subject with rights in the ACM, the owner of the ClaimChain has access to their up-to-date Diffie-Hellman public key. We also assume that the subject, at the time of reading, will have access to a Diffie-Hellman public key associated with the ClaimChain containing the claims to be read. We use those keys to derive pairwise encryption keys between the owner of the ClaimChain and reading subjects, under which we encrypt the VRF values of the labels these readers have access to in the ACM.

Each access right is encrypted separately under a look-up key and cryptographic key specific to the reader and the label to be read, effectively ensuring that a reader can only find claims for which she knows the label. For each subject-label pair, her set of reading rights is given by a number of single-label *capabilities*. We derive a subject-specific capability look-up key $l^A = H_3(\mathsf{nonce} \parallel s \parallel \mathsf{claim\_label})$ and similarly capability encryption key $K^A = H_4(\mathsf{nonce} \parallel s \parallel \mathsf{claim\_label})$, where s is a shared DH secret. Then we encrypt the VRF value for the specific label using K^A as $p^A = \mathsf{Enc}_{K^A}(k)$. The pairs (l_A, p_A) for all subject-label pairs are stored in the *capabilities map*.

A user wishing to access a claim within a ClaimChain uses the information in the *capabilities map* to retrieve the VRF value for the label of interest, within the block. This VRF value is then used to derive the look-up keys and decryption keys necessary to retrieve and decrypt the claim itself in the *claims map*. We note that the VRF value cannot be verified before the claim has been retrieved, since the proof necessary is not included in the capability map.

3.1.5. Object store abstraction. For the purpose of flexibility of ClaimChain deployments, one useful abstraction to have is self-certifying key-value stores, whose keys are hashes of corresponding values. We call this abstraction an *object store*. We assume that all objects related to ClaimChains – like blocks and tree nodes – reside in such key-value stores. We note that objects in a ClaimChain block may be either stored in a block payload directly, or may be referenced using their cryptographic hash representation. In the latter case, these objects are stored along the blocks as "encrypted blobs", and need to be available to readers in the same object store.

We consider the object stores have the following key properties, which are crucial in supporting flexible on-line, off-line, centralized or decentralized operations for ClaimChains:

Self-certification. Given an object store it is easy to verify its integrity, by checking the invariant that all keys are the hashes of the respective objects they map to.

Conflict-free merge. Given two valid object stores it is easy to merge them, without leading to conflicts or inconsistencies, by simply constructing a store with the union of their key-value pairs. This operation can be performed recursively to merge multiple stores.

Tolerance to partial views. An incomplete object store may result in failed attempts to authenticate ClaimChain data structures, or failure to check inclusion or non-inclusion of claims. However, an incomplete store cannot lead to an erroneous inference on the authenticity, inclusion or exclusion or any chain or claim.

Flexible distribution. Object stores may be replicated across infrastructure, and mirrored locally. They may also be implemented in a sharded manner on-line to increase performance, guaranteeing that partial off-line operation of partial stores cannot lead to errors.

We note that concrete designs must opt for a specific instantiation of this object store abstraction. For instance, it can be implemented on top of a publicly accessible on-line key-value store even without authorization or authentication. A provider-less, off-line instantiation is also possible. In this scenario users of a system employing ClaimChains maintain local object

stores, that can be updated through gossiping with other users, or by receiving out-of-band evidence about other user's ClaimChain status. The store self-certification and conflict-free merge properties give flexibility to this design option, since it permits that evidence that does not come from an authoritative source to be included in the store.

4. A decentralized PKI based on ClaimChains

We now describe how ClaimChains can be used to support the deployment of a decentralized Public Key Infrastructure.

Overview & Semantics. First, we give claims in ClaimChains a precise meaning that enableÑÚ clients to accept or reject bindings of identities to keys.

Consider a ClaimChain cc_A with metadata A, and a claim binding name B to another ClaimChain cc_B (which we denote as $B \rightarrow cc_B$). Using the SecPAL language [12] formalism we can say this chain encodes the following two SecPAL statements:

- "cc_A says $A \rightarrow cc_A$" (self-claim)
- "cc_A says $B \rightarrow cc_B$" (cross-claim)

ClaimChains ensures that those basic claims about self or other users can be relied upon with high integrity. The structure of the chain further ensures that previous claims can be retracted, while new claims can be included in new blocks. Furthermore, the capability-based access control system ensures that only authorized principals get access to claims in a given block.

Effectively, this makes ClaimChains suitable to build a PKI to support modern public key cryptography, from authentication to end-to-end encryption. In such scenario principals dynamically create and update ClaimChains, embedding metadata about their own identities and keys into them, and certifying the status of others' ClaimChains. By gathering evidence from trusted contacts (see below), users can obtain and validate keys associated to identities of their interest.

Trust in claims. We consider that an owner of a ClaimChain implicitly trusts all statements in her own chain to be true. However, this is not necessarily true for claims that are present in other users' ClaimChains. For those to be trusted, a *social verification* process needs to take place. From a theoretical perspective, *social verification* is simply an algorithm that takes claims (either self-claims or cross-claims) and then resolves them into name-ClaimChain trusted bindings. This algorithm may be specified in a formal language, with well defined semantics and allowing for efficient resolution, such as SecPAL; or, it can simply be an ad-hoc algorithm specified in any computer programming language.

We use the term social verification to emphasize that the rules under which users accept claims of other users *cannot* be universal or derived through deduction. Instead, each user (or her software) has to define personal rules that specify under what conditions claims in other chains can be trusted, be it self-claims or cross-claims. Some examples of such *social verification* rules that can be implemented using ClaimChains as evidence repositories:

Trust in certification authorities. A user may choose to believe all statements that are included in one of the ClaimChains, maintained by certification authorities. This is equivalent to the model employed by web browsers which consider that a TLS certificate is authentic if it is signed by a known certificate authority.

Traditional Web of Trust. A user may chose some of its 'friends' as being trusted, and thus accept all claims that are included in their ClaimChains. This decentralizes the role of certification, in that different users may chose who has authority to make statements about others.

Threshold / social trust schemes. A user may accept a binding between a name and a chain, or a self-claim, if a certain number of other designated users certify that claim. This reduces the risk of incorrect bindings derived from a single poor choice of whom to trust.

4.1. ClaimChain-based PKI deployment options

We now describe two possible deployment options for a ClaimChain-based PKI, which assume that users have a social verification procedure in place.

A ClaimChain-based PKI may be deployed in a context where users have the ability to query an on-line service to i) retrieve the latest state of another user's ClaimChain; ii) access all capabilities that they may access on such remote ClaimChain; and iii) retrieve all claims unlocked by those capabilities. This service may be as simple as a key-value store (implementing the object store abstraction), with an additional operation returning latest block for a queried identity (we call this *head distribution*).

In this model, users need to upload all updates of their chains to the on-line service providers. Other users need to perform look-ups to discover if ClaimChains of interest have been updated. Of course, users may locally mirror some of the object store to avoid performing duplicate queries.

Such an on-line deployment setting could be used by providers to track the latest state of users' ClaimChains, and be ready to serve their latest block upon demand. These providers need to perform work linear in the number blocks uploaded to the store to maintain those indexes. Also, these providers need to be trusted to provide the latest updates to each chain. This requirement can be relaxed by consulting bindings from multiple non-colluding providers.

In some cases, clients may have to operate off-line, without being able to query on-line services, and hence

they require a fully decentralized operation. A key example is the case of email communications, where emails may be composed or read off-line. Furthermore, it is considered good practice for Mail User Agents (MUAs) to not connect to services other than the email provider's for any checks. In this setting, users can maintain a local off-line storage for their own ClaimChain. Users update their own local object store when updating their chain; and use the 'natural' email communication with others to propagate the new state of their ClaimChains, and optionally others' ClaimChains, to their communication partners. At a minimum we foresee users including in all their email communications the hash of their latest head, and potentially their latest block. This allows a user's communication partners to keep track of this user's ClaimChain, and updates to her signature and encryption keys.

Note that ClaimChains can also support selective dissemination of claims to reduce the bandwidth overhead associated with each message. For instance, only a selection of the capabilities and claims that are of relevance to the recipient, together with the latest block may be included with every message.

A key example relates to transferring only information useful to allow for introductions via email: a user Alice, writing an email to Bob and Charlie, may include in their message claims that would allow Bob and Charlie to check their own claims are in Alice's chain, and potentially to start relying on Alice's binding (if they trust Alice) to authenticate each others' identities. In that case Alice would include the latest blocks of Alice, Bob and Charlie, and also the capabilities and claims in her latest block that allow Bob and Charlie to establish that she has included their latest blocks are corresponding to their names.

We call this process of embedding messages the information about one's own contacts (i.e. latest blocks of the contacts) in the messages, *in-band gossiping*.

In the case of email, the ClaimChains blocks and shards of the local object store can be encoded as email headers to be transferred transparently alongside messages, as proposed in Autocrypt.

Alternatively, an attachment with ClaimChain-related information, interpretable by compliant clients, may be included in a multi-part SMTP envelope.

Finally, we note that on-line object stores can also be used in this decentralized setting. In such a case, users only need to share the heads of their ClaimChains (decentralized head distribution) and some information allowing receivers to access their object store. Receivers then can retrieve all the blocks and tree nodes from the specified object store using only the ClaimChain head.

Deployment trade-offs. We now compare the deployment options in terms of the privacy properties and assumptions, performance, and effectiveness of key propagation in the system. See Table 1 for a summary of the comparison.

The fully decentralized setting, in which chains are stored locally and only transmitted upon demand offers in principle the best privacy properties. For the system to satisfy correctness and privacy, users need only rely on (i) their friends being honest on not transferring the private information and access tokens (VRF values k), and (ii) their social circle consisting of friends or other information providers being diverse, non-malicious, and reliable enough, so that the social validation process can detect if a queried identity-ClaimChain binding is correct or not.

Although the trust assumptions are minimal, this setting has the most costly performance requirements. At each point in time, users need to send their full chains ($O(n)$) to all of their recipients ($O(b)$), and often parts of others' ClaimChains for gossiping. We assume that the size of gossiped information is bounded by $O(m \cdot n)$, i.e., the full chains of the m users in the system. The outgoing bandwidth therefore is equal to the size of the own and gossiped chains, $O(n + m \cdot n) = O(m \cdot n)$. On the receiving end, users need to download $O(m \cdot n)$ data, from at most $O(b)$ senders, so the incoming bandwidth is $O(b \cdot m \cdot n)$. All users locally store their own chain and, for validation requirements, all gossiped information they have received, $O(n + m \cdot n) = O(m \cdot n)$. These are worst-case asymptotical estimations. In Section 6, we see that requirements are much smaller in practice.

Moreover, in this setting, the effectiveness of key propagation is strongly dependent on users' patterns of communications. For instance, users that frequently communicate with each other, are likely to have up-to-date knowledge of each others' keys; while users sending sporadic emails to someone that they never communicated with, and who is outside of their social circle, are unlikely to know this recipient's key.

Adding the on-line object store providers greatly improves the availability of ClaimChains, and of key propagation, but it introduces additional requirements: these providers need to have high availability, and they are trusted to not exploit access patterns to infer private information about the users.

When the distribution of heads is decentralized, the use of on-line storage providers significantly reduces the outgoing bandwidth, from $O(m \cdot n)$ to basically $O(1)$ – users only need to send heads of their own ClaimChains. Furthermore, the local storage may also be outsourced, only requiring to locally keep own ClaimChain head, and heads of other people ($O(m)$).

Finally, a provider that distributes the latest ClaimChain head for a given identity, as well as the block content, has identical trust requirements to those of on-line storage provider. The advantage is that it has an immediate benefit of 100% encryption key propagation. Indeed, even if the sender has never communicated with recipients, she can learn their ClaimChains by querying one or more on-line providers. The social validation policy then needs to ensure that the responds are correct. As for the storage and bandwidth costs, since gossiping

Table 1. Performance comparison of deployment options. (b: branching factor of the social graph, m: number of users, n: upper bound of the chain size in the system at a point in time.)

	Available object store provider		
	On-line deployment	**Decentralized head propagation**	**Fully decentralized operation**
Outgoing bandwidth	$O(n)$	$O(n)$	$O(m \cdot n)$
Incoming bandwidth	$O(b \cdot n)$	$O(b \cdot m \cdot n)$	$O(b \cdot m \cdot n)$
Local storage	$O(m)$	$O(m)$	$O(m \cdot n)$
Proportion of enc. messages	100%	See simulations (Section 6)	See simulations (Section 6)

is not needed, the incoming bandwidth is reduced to $O(b \cdot m)$.

5. Evaluation

In this section we evaluate the security, privacy, and performance of ClaimChains.

5.1. Experimental setup

We have implemented a prototype of ClaimChains in Python.[8] This implementation uses the [redacted] library for elliptic curve cryptography operations, which internally relies on OpenSSL[9] C library. For the implementation of verifiable skip-list and sorted Merkle tree operations, we use the [redacted] library, which is written in pure Python.

We run our experiments on an Intel Core i7-5600U CPU @ 2.60GHz machine using CPython 3.5.2. For simplicity, each chain block metadata field (see Section 3.1.2) only includes cryptographic public keys.

Cryptographic primitive instantiations. For symmetric encryption, we use AES128 in GCM mode. For public key cryptography, we use ECDSA, ECDH, and CONIKS VRF scheme [2], with a NIST/SECG curve over a 224 bit prime field. As a basic hash function we use SHA256, with H_1 through H_4 instantiated as SHA256 hash of a message with some prefix prepended. All the lookup keys on the claim map are truncated to 8 bytes, which ensures absence of collisions for up to 2^{32} entries in the map. The size of the per-block nonce is set to 16 bytes, using the standard Linux `urandom` device as PRNG.

5.2. Core operations performance

In this section we present results regarding the computation time of the core ClaimChain operations, as well as the storage required to keep and transmit the structures.

Core operations timing. We measure the performance of the core ClaimChain operations: encoding and decoding of claims and capability entries (see Sections 3.1.3 and 3.1.4). To this end, we encode and decode 1000 claims each with a random 32-byte label and 512-byte content, and for each claim we encode and decode a corresponding capability entry to a random DH public key. The choice of label and content size is representative of the PKI use case of ClaimChains. 32-byte labels can accommodate email addresses or other user identifiers (32 ASCII characters fit most email addresses in the dataset we use in our experiments in Section 6); and consider claim content to be ClaimChain heads (hashes) or public key material. For the sake of evaluating the worst-case scenario we choose 512 bytes for the content, since this can fit a 2048-bit RSA key if keys had to be directly included in the ClaimChains.

Table 2. ClaimChain basic operations timing

	avg (ms)	st. dev.
Single-label capab. lookup key computation	0.12	0.02
Single-label capab. decoding	0.14	0.02
Single-label capab. encoding	0.14	0.00
Claim encoding	1.51	0.16
VRF computation	1.46	0.16
Claim decoding	2.48	0.29
VRF verification	2.44	0.29

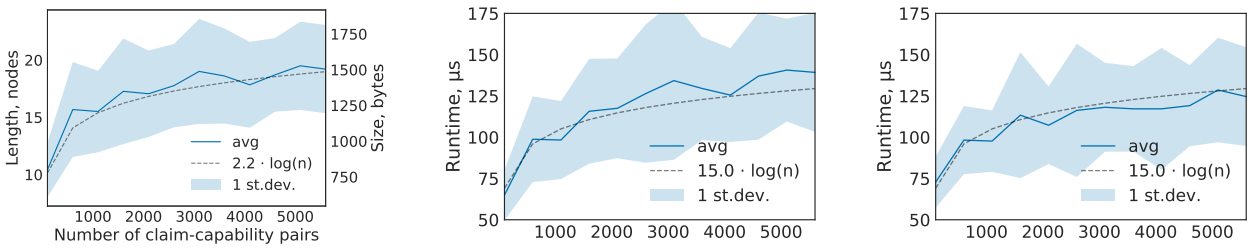
In Table 2 we report average and standard deviation of computation time over 1000 executions for each operation. One can see that time to encode and decode claims is mostly the time of VRF computation and verification. Encoding and decoding of capabilities, and computing the capability look-up keys each take under 0.15 milliseconds, making the computation time basically negligible. The worst computation time is under 2.5 milliseconds for decoding a claim.

Constructing the claim map. The most computationally expensive operation that ClaimChain owners perform is constructing the claim map. This operation is performed when a new block is constructed. For the measurements, we pick a number of claims N, then simulate N readers by generating their DH public keys, and encode one capability per claim to a random reader. In the PKI setting, this corresponds to having N cross-references readable by N contacts. We pick several numbers N of claim-capability pairs from 100 to 5000, and for each number we construct a sorted Merkle tree with the encoded entries, repeating 20 times.

In Fig. 2(a) we report the average time and variance (over 20 experiments) to build the tree. The operation takes under 0.3 seconds for 5000 claim-capability pairs. In a PKI for messaging, we expect average users to have much fewer than 10,000 map entries in their

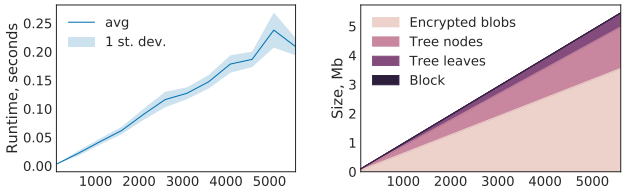
8. Our implementation and reproducible experiments will be made available upon publication.

9. https://openssl.org



(a) Size of inclusion evidence size (b) Computation time of inclusion evidence (c) Verification time of inclusion evidence

Figure 1. Storage and performance measurements of inclusion evidence for a single entry



(a) Claim map construction time (b) Storage size breakdown

Figure 2. Total storage size and claim map construction time

ClaimChains. For example, in our further simulations of fully decentralized setting in Section 6, the number of items in maps normally does not exceed 1000. Hence, the time of tree construction seems suitable for scalable deployments of ClaimChains.

Inclusion evidence measurements. In the fully decentralized deployment (Section 4.1), we assume that along with the block, owners send resolution paths to relevant claims and capabilities in the block map tree. These paths serve as evidence of inclusion of claims and capability entries in the tree. We measure the time to compute inclusion evidence for a single map entry, and to verify a single piece of inclusion evidence, depending on the number of entries in the owner's claim map. We use the same setting as in the previous experiment, assuming that the tree is already constructed for each number of claim-capability pairs. For every N, we choose at random 200 look-up keys from the claim map, and we measure the evidence computation and verification times, as well as the evidence size.

Fig. 1(a) shows the size measurements, and Fig. 1(b) and 1(c) show the timing results. We see that evidence computation time, verification time, and evidence size are all logarithmic in the number of entries in the map. Even though asymptotic complexity is logarithmic, users may trade off the computation time on the ClaimChain owner side for the bandwidth size by sending the whole tree, whose size we report below.

Storage size. To estimate the memory requirements for the owner to store a ClaimChain, we have measured the size of a block, as well as of its parts: the claim map tree, and the values in the claim map ("encrypted blobs"). The size of the claim map and size of all encrypted blobs depends on the number of entries in the map, and the size of encrypted blobs depends on the size of claims.

We use the same approach as the previous two experiments: varying the number of claim-capability pairs in the claim map within the range. We show the size breakdown depending on the number of claim-capability pairs in Fig. 2(b). The block size is constant, and can only grow if security parameters change (size of cryptographic public keys, or hash length increases), or additional metadata about the owner is added. We see that the overhead of tree nodes is significant. Yet, it is under 2 Mb even for 5000 claim-capability pairs, which suggests that sending the whole tree may be a feasible trade-off for avoiding the computation of evidence paths in a decentralized setting.

5.3. Security and privacy evaluation

In this section we formally define the security and privacy properties provided by ClaimChains, and outline sketches of proofs that our constructions satisfy these properties.

5.3.1. Privacy. We define ClaimChain privacy as a combination of three properties: *reader anonymity*, *capability unlinkability*, and *claim indistinguishability*. We now formally describe each property as a cryptographic game.

Reader Anonymity. Reader anonymity ensures that the capabilities and claims contained in a ClaimChain do not leak the identity of those allowed to read those claims – guaranteeing that only authorized readers can learn contacts of the ClaimChain owner.

The reader anonymity game ReadAnon (see Exp. 1) accepts, as all of the following privacy games, a universe of possible readers $\mathcal{R}$ and claims $\mathcal{C}$, and an existing ClaimChain *cc*. The adversary chooses two challenge readers r_0 and r_1, as well as a set of other readers R, an

arbitrary set $C \cup \{c\}$ of claims to extend a chain cc, and the mapping M between readers R and claims C. A bit b is chosen at random and the ClaimChain cc', is extended from cc with claims C and $\{c\}$, claim-reader mapping M, and a capability to read claim c for either reader r_0 or reader r_1. The adversary is provided with the resulting ClaimChain, and all DH secret keys of all the readers R, but not r. She has to infer the bit b, guessing which challenge reader was used to produce cc'.

Experiment 1: $\mathsf{ReadAnon}(\mathcal{A}, \mathcal{C}, \mathcal{R}, cc)$

$C, R, M \leftarrow \mathcal{A}(\mathcal{C}, \mathcal{R})$
$c, r_0, r_1 \leftarrow \mathcal{A}(\mathcal{C}, \mathcal{R} \setminus R), \quad \text{s.t. } r_0 \neq r_1$
$b \xleftarrow{\$} \{0,1\}$
$C' := C \cup \{c\}; M' := M \cup \{(c, r_b)\}$
$cc' \leftarrow \mathsf{Extend}(cc, C', M')$
return $\mathcal{A}(cc, cc', \{\mathsf{sk}_{\mathsf{DH}}^{\bar{r}} \mid \bar{r} \in R\}) = b$

Theorem 1 (Reader Anonymity)**.** *For any PPT Adversary $\mathcal{A}(\cdot)$, an arbitrary universe of claims $\mathcal{C}$ and readers $\mathcal{R}$, and any ClaimChain cc, construction of ClaimChains satisfies that the adversary wins* ReadAnon *game with negligible advantage over random guessing:*

$$\Pr[\mathsf{ReadAnon}(\mathcal{A}, \mathcal{C}, \mathcal{R}, cc) = 1] - \frac{1}{2} \leq \mathsf{negl}(\kappa)$$

Proof. (Sketch) We note that the adversary does not know the DH secret key of the owner of the chain or either of the challenge readers. Thus, when the chain is created, the derived shared secret between the owner and any of the two readers, is indistinguishable to the adversary by decisional Diffie-Hellman assumption. Since the encoded capability (c, r_b) is encrypted with key derived from the shared secret, it is is computationally indistinguishable from a random string based on the CPA security of an encryption scheme. □

Capability Unlinkability. Capability unlinkability ensures that an adversary cannot infer which claims a given reader can access.

In this game, CapUnlink (Exp. 2), the adversary chooses a set of readers R, a set of claims C to extend a chain cc, and a mapping between the readers and claims M. Adversary also picks a reader r not in R, and two distinct claims c_0 and c_1 from the universe of all claims. A bit b is chosen at random and the ClaimChain is extended with claims R, mapping M, and depending on the bit, one of two capabilities, (c_0, r) or (c_1, r). As previously, the adversary is then given the resulting ClaimChain, and secret keys of all readers R, but not r. She has to infer the bit b, guessing which challenge capability was added to the chain.

Theorem 2 (Capability unlinkability)**.** *For any PPT Adversary $\mathcal{A}(\cdot)$, an arbitrary universe of claims $\mathcal{C}$ and readers $\mathcal{R}$, and any ClaimChain cc, construction of ClaimChains satisfies that the adversary wins* CapUnlink *game with negligible advantage over random guessing:*

Experiment 2: $\mathsf{CapUnlink}(\mathcal{A}, \mathcal{C}, \mathcal{R}, cc)$

$C, R, M \leftarrow \mathcal{A}(\mathcal{C}, \mathcal{R})$
$c_0, c_1, r \leftarrow \mathcal{A}(\mathcal{C}, \mathcal{R} \setminus R), c_0 \neq c_1$
$b \xleftarrow{\$} \{0,1\}$
$C' := C \cup \{c_b\}; M' := M \cup \{(c_b, r)\}$
$cc' \leftarrow \mathsf{Extend}(cc, C', M')$
return $\mathcal{A}(cc, cc', \{\mathsf{sk}_{\mathsf{DH}}^{\bar{r}} \mid \bar{r} \in R\}) = b$

$$Pr[\mathsf{CapUnlink}(\mathcal{A}, \mathcal{C}, \mathcal{R}, cc) = 1] - \frac{1}{2} \leq \mathsf{negl}(\kappa)$$

Proof. (Sketch) As in the reader anonymity game, the capability (c_b, r) is indistinguishable from random to an adversary not knowing the secret key of reader $\mathsf{sk}_{\mathsf{DH}}^{r}$. □

Claim Indistinguishability. Finally, claim indistinguishability ensures that a ClaimChain does not leak which claims it contains.

In the claim indistinguishability game ClaimInd (see Exp. 3) the adversary chooses sets C, R, M, and two challenge claims c_0 and c_1. A bit b is chosen at random and the ClaimChain cc is extended with either claim c_0 or claim c_1. Unlike in the previous games, the adversary is then given the encryption secret keys (K) for all claims in C, but not c_0 and c_1. Given the resulting ClaimChain and the keys, she has to infer the bit b, namely which challenge claim was used to extend the chain.

Experiment 3: $\mathsf{ClaimInd}(\mathcal{A}, \mathcal{C}, \mathcal{R}, cc)$

$C, R, M \leftarrow \mathcal{A}(\mathcal{C}, \mathcal{R})$
$c_0, c_1, r \leftarrow \mathcal{A}(\mathcal{C} \setminus C, \mathcal{R} \setminus R), c_0 \neq c_1$
$b \xleftarrow{\$} \{0,1\}$
$C' := C \cup \{c_b\}; M' := M \cup \{(c_b, r)\}$
$cc' \leftarrow \mathsf{Extend}(cc, C', M')$
return $\mathcal{A}(cc, cc', \{K_{\bar{c}} \mid \bar{c} \in C\}) = b$

Theorem 3 (Claim Indistinguishability)**.** *For any PPT Adversary $\mathcal{A}(\cdot)$, an arbitrary universe of claims $\mathcal{C}$ and readers $\mathcal{R}$, and any ClaimChain cc, construction of ClaimChains satisfies that the adversary wins* ClaimInd *game with negligible advantage over random guessing:*

$$\Pr[\mathsf{ClaimInd}(\mathcal{A}, \mathcal{C}, \mathcal{R}, cc) = 1] - \frac{1}{2} \leq \mathsf{negl}(\kappa)$$

Proof. (Sketch) Note that the adversary does not know the secret key $\mathsf{sk}_{\mathsf{VRF}}$ of the owner of the chain. The claim encryption key K is derived from a VRF value k and block nonce, using a hash function. She can guess K with probability no greater than she can forge the VRF value and proof, which is negligible. Hence, under the CPA security of an encryption scheme, the resulting ciphertexts of the claims c_0 or c_1 are indistinguishable to the adversary. □

5.3.2. Non-equivocation. A key property provided by ClaimChains is *non-equivocation*: given a claim in a ClaimChain block, it is not possible for the owner to show different values of this claim to different users. We define a cryptographic game, called NonEq involving a polynomially bound adversary $\mathcal{A}(\cdot)$ and the procedures used to create lookup keys, and to extend and read the tree containing the capabilities map.

In this game, NonEq (see Exp. 4), the adversary builds an arbitrary structure that she passes as a ClaimChain. She then constructs two distinct claims c_0, c_1 with the same label claim_label, encoded to potentially two different readers r_0 and r_1. The adversary succeeds if these claims can both be verified as belonging in the ClaimChain, by the checking algorithm.

Experiment 4: $\mathsf{NonEq}(\mathcal{A}, \mathcal{C}, \mathcal{R})$

$cc, \mathsf{claim\_label}, (r_0, c_0), (r_1, c_1) \leftarrow \mathcal{A}(\mathcal{C}, \mathcal{R})$, s.t. $c_0 \neq c_1$
$d_0 := \mathsf{Check}(cc, \mathsf{claim\_label}, r_0, c_0)$
$d_1 := \mathsf{Check}(cc, \mathsf{claim\_label}, r_1, c_1)$
return $d_0 \wedge d_1$

Theorem 4 (Non-equivocation)**.** *For any PPT Adversary $\mathcal{A}(\cdot)$, an arbitrary universe of claims $\mathcal{C}$ and readers $\mathcal{R}$, construction of ClaimChains satisfies that the adversary wins NonEq game with negligible probability:*

$$\Pr[\mathsf{NonEq}(\mathcal{A}) = 1] \leq \mathsf{negl}(\kappa)$$

Sketch: Non-equivocation relies on two properties of ClaimChain building blocks. First, VRF scheme guarantees that for a given claim_label known by the reader, and VRF key pair $(\mathsf{sk}_{\mathsf{VRF}}, \mathsf{pk}_{\mathsf{VRF}})$, the adversary can only find one possible VRF value k (recall that $k, \mathsf{proof} = \mathsf{VRF}_{\mathsf{sk}_{\mathsf{VRF}}}(\mathsf{claim\_label} \parallel \mathsf{nonce})$), and therefore, can only derive one look-up key $l = H_1(k)$. Second, given the look-up key l and a hash of the non-equivocable Merkle tree root MTR, it is infeasible for the adversary to present two different resolution paths that start in the same root but end in leaves with different content, without breaking the collision resistance property of the hash function used. By union bound, the probability of adversary breaking either of this properties is negligible.

5.4. Integrity and authenticity

Block integrity. For a block B having hash digest $H(B)$, it is unfeasible to find another block B' with claims, metadata, pointers to previous blocks, or any other data from the block B dropped, rearranged, or otherwise modified, such that $H(B) = H(B')$.

Proof. Trivial reduction to *second preimage resistance of hash function H.* □

Extension to chains. For a chain $\mathbf{B} = \{B_1, B_2, ..., B_n\}$ with head $H(B_n)$, it is unfeasible to find another chain $\mathbf{B'} = \{B'_1, B'_2, ..., B'_m\}$ with claims, metadata, any other data from any block B_i of chain $\mathbf{B}$, and blocks themselves dropped, rearranged, or otherwise modified, such that $H(B_n) = H(B'_m)$.

Proof. Consequence of hash chain construction and block integrity. □

Authenticity. Given a block $B = (X,\ \sigma)$, where X is the payload of the block and σ its signature, and associated verification key pairs $(\mathsf{sk}_{\mathsf{SIG}}, \mathsf{pk}_{\mathsf{SIG}})$, an adversary cannot forge a new block.

Proof. Trivial reduction to selective unforgeability of the signature scheme. □

6. Key propagation in a fully decentralized setting

A fully decentralized deployment of a ClaimChain-based PKI offers the best privacy properties among the options we discuss in Section 4.1. It is, however, comparatively more expensive in terms of storage and bandwidth. Moreover, effectiveness of key propagation in this setting is largely unpredictable, since it depends on the users' communication patterns. In order to evaluate how well encryption keys propagate and what is the overhead in terms of storage and bandwidth, we use the Enron dataset[10] [13], [14]. This dataset is an archive of email directories of 147 employees of Enron, containing around 500,000 emails in total (about 230,000 after removing duplicates and non-readable email addresses). It is often used in research to simulate realistic relationships and patterns in messaging applications [15], [16].

In our simulations, every email sender in the dataset maintains their own ClaimChain. Each of the senders embeds in every sent message data which contains their ClaimChain, and may also contain parts of ClaimChains of other people. Upon receiving a message, users record all the obtained ClaimChain data in their *gossip storage*.

For each of our experiments we use a subset of consecutive 10,000 emails from the whole log. To simulate the use of ClaimChains, we loop through the emails in this batch of 10,000 chronologically, updating ClaimChains of senders and receivers after each sent email.

At the beginning of each experiment, we generate encryption keys for all users, and add an empty block to their ClaimChains only containing these encryption keys and ClaimChain key material. In our simulations, a sender only updates her ClaimChain either when there are any queued updates (changes to a claim, new claims, or updated capabilities) relevant to recipients of the current message, or when her own encryption key is rotated. For the purpose of illustration we consider that users rotate their keys every 50 outgoing emails. This choice affects the number of emails encrypted under a stale key. If users rotate their keys less often, this number decreases, and otherwise increases.

10. http://www.cs.cmu.edu/~enron/

We use the proportion of encrypted emails as a measure of effectiveness of key propagation, since enabling the encryption of end-to-end communications is the end goal of PKIs. An email can only be encrypted when the sender has received some parts of ClaimChains of all the recipients, directly or through gossiping. If the sender has not learned about a key of at least one of the recipients prior to sending, the email is sent in plaintext. For each sent email we record its encryption status: whether it has to be sent as plaintext, or it can be encrypted, or encrypted with a stale key.

Throughout experiments we measure users' local chain storage size and gossip storage size, by recording these for each receiver upon them receiving and processing a message. For every sender, we also record the size of ClaimChain data added to every message.

6.1. Simulated settings

Public setting. In this model users gossip to their correspondents all the information available to them. This means that they include latest heads of their contacts as public claims in their ClaimChains. Then, when sending an email, along with own latest ClaimChain block, users send evidence paths for all public claims in their chains. In turn, recipients parse the attached evidence and update their friends' entries accordingly. This scenario is close to the operation of the PGP Web of Trust, in which users always attach their public key along with signatures on the keys of their friends.

Private setting. In this model senders implement access control to their ClaimChains. In our experiments, we consider that access control lists are built in an incremental way, via "introductions". Every time there is an email with more than one recipient, all recipients earn the capability to see each other's updates in the sender's ClaimChain. Such capabilities are persistent over time, i.e., once a recipient has been granted access to read information about a contact, in subsequent emails she will be able to access claims about the contact even if that contact is not in the email.

6.2. Results

Effectiveness. To measure the effectiveness of propagation of key material at different points in time, we group emails into batches of 1,000 (i.e., ten batches per experiment of 10,000 emails). In each batch we compute the distribution of emails by their encryption status.

In the first set of measurements, we only report on emails sent and received by users within the Enron user set (147 employees). This data represents complete view of communications within a group in a corporate setting. In all 10 experiments, we observe that the number of encrypted emails increases over time, although there is no clear trend, and variation is high. For the private setting in the 4th, 7th, and 10th batch, on average 52% (± 37[11]), 52% (± 39), 44% (± 37) of emails respectively are encrypted. The overall percentage of encrypted emails in the private setting is 46% (± 19). A significant portion of emails are encrypted under stale encryption keys of recipients, due to our chosen key rotation policy that forces senders that are very active to change keys very often.

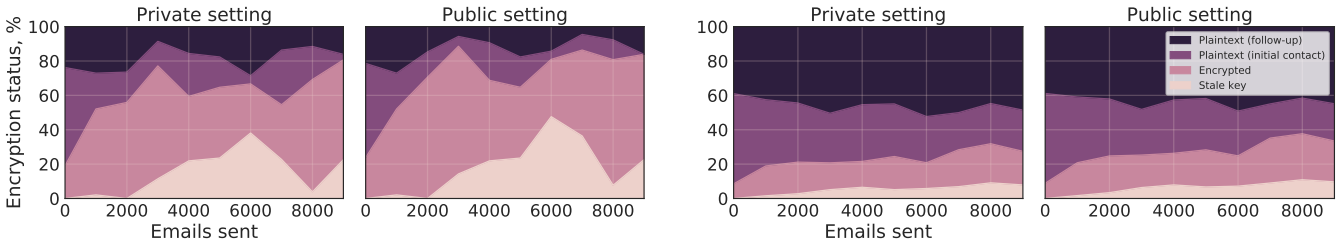
We present results of one of the experiments in Fig. 3(a) (left). Variability in the results throughout the experiments, as well as within the experiment, demonstrates that success of encryption in the decentralized setting is largely unpredictable.

The number of encrypted emails in the public setting is always greater or equal than in the private setting. The reason is that in this setting users gossip all the information available to them, whereas in the private setting they only share the subset that is relevant to current recipients of each email. We can see an example of this in Fig. 3(a) (right), where we show the results from simulations in the public setting for the same interval in the log as in the previous graphic. Indeed, the plot shows that in every batch the proportion of encrypted emails is greater for the public one. The overall proportion of encrypted emails in this interval is 52% in the public setting, and 46% in the private. We see below in this section, however, that even though public setting is more effective, it has very significant costs in terms of privacy, storage and bandwidth.

In the second set of measurements, we use all the emails to compute encryption status distributions, including those where some of senders or recipients are not in the Enron user set. Compared to the previous measurements, the proportion of encrypted traffic here is significantly lower, total average being 19% (± 9) in the private setting. Since in this case simulations are not restricted to communication within a clique of users, many senders and recipients are outside of the group of 147 employees, or even are not Enron employees. Many emails can not possibly be encrypted, since a big portion of them have at least one recipient that the sender has not learned anything about prior to sending the email (*"Plaintext (initial contact)"* area in the plots). In Figure 3(b) we show these measurements in the same interval as previously. In this interval, the overall proportion of encrypted emails is 24% in the public setting, and 20% in the private.

Storage and bandwidth costs. To estimate network bandwidth costs, for each email in a partial log of 10,000 we record the size of the ClaimChain data being sent. This data consists of all blocks of a sender's chain that were not sent to current email recipients before, tree inclusion paths to relevant claims, and claims themselves. In Figure 4(a) we show the bandwidth size distribution in four batches of emails (0-2500, 2500-5000, 5000-7500,

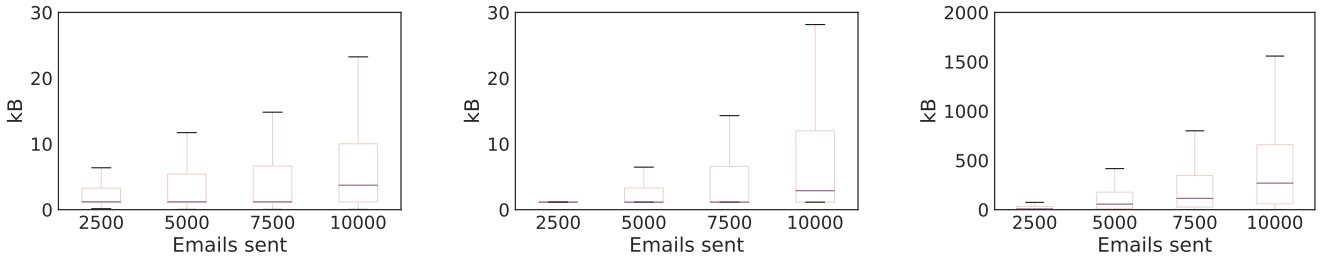
11. By $\pm x$ we denote 95% Student t-distribution confidence interval in terms of percentage points



(a) within the Enron user set

(b) All email traffic

Figure 3. Encryption status of email traffic. *Plaintext*: Initial contact – the email sender has not learned a key of at least one of the email recipients, and it is the first time contacting this recipient; Follow-up – subsequent email sent to a recipient for which the sender does not have a key. *Encrypted*: Stale key – at least one of the email recipients has changed her encryption key since it was learned by the sender; Encrypted – all keys are up to date.



(a) Outgoing bandwidth costs

(b) Size of the local user storage

(c) Size of the gossip storage

Figure 4. Storage and bandwidth measurements in the private setting

and 7500-10,000) across all experiments. The bandwidth costs rise as the simulation progresses, since chains grow over time. We observe that there is a lot of variation in growth, caused by the different behaviours in the dataset: ClaimChains of users that send or receive more emails grow faster than chains of others. In the private setting, the costs are normally not more than 30 kB per email. In the public setting, the costs are much higher, up to 500 kB per message in the first batch, rising to 2 MB by the end of simulations.

To measure client memory requirements, we record the size of storage used by each recipient upon receiving and processing every incoming email. We separately measure *local storage* – the space taken by user's own ClaimChain: all the blocks, tree nodes, and encrypted blobs, and *gossip storage* – the space taken by information that users have received from their correspondents through gossiping. In the private setting, local storage is mostly under 30 kB, and gossip storage size remains under 2 MB by the end of simulations. As with the bandwidth measurements, we show in Figures 4(b) and 4(c) the size distributions in four batches across all experiments. We see rising trends, and large variation that can again be explained by non-uniform behaviour of users. In the public setting, the local storage can take up to 4 MB, and gossip storage up to 20 MB.

Comparison of public vs private. In our experiments we observe that, as expected, public gossiping is more effective at propagating the key information than selective sharing. In fact, the public setting provides an upper bound for key propagation in our fully decentralized setting. Nonetheless, in the measurements visualized in Fig. 3, we observe that the advantage of public gossiping is relatively small compared to more privacy-friendly selective sharing: 6 p.p. increase in the total number of encrypted emails within the Enron user set, and 4 p.p. increase for all traffic. At the same time, the amount of shared information is vastly larger in the public setting, difference being in orders of magnitude, both for outgoing bandwidth (30 kB vs. 4 MB), and for gossip storage size (2 MB vs. 20 MB). These simulations show that gossiping of all contacts to everyone seems to not achieve significant improvements, while sacrificing users' privacy. This suggests that the selective revealing of contacts, enabled by ClaimChain cryptographic mechanisms, can offer a better trade-off between privacy and utility than the traditional Web of Trust-like sharing model.

Finally, our results show the key propagation performance of in-band gossiping is quite poor. Even within a clique of Enron users, only an average of 46% emails are encrypted successfully after simulating 10,000 emails. These results suggest that improving privacy by fully removing highly available centralized entities from the

PKI may result in a big impact on its performance.

7. Discussion: integration, security, and privacy

In this section we discuss aspects that must be taken into account when integrating ClaimChains within a system.

Hiding access patterns. Even though the cryptographic mechanisms in ClaimChain ensure the confidentiality of the content, usage patterns may reveal information to an adversary observing interactions (e.g., an on-line object store provider). For instance, the fact that Alice queries Bob's ClaimChain reveals that they know each other. To preserve privacy, accesses to the ClaimChain store can be performed using private information retrieval (PIR) [17] that can hide who accesses whose chains. Recent work has shown the PIR schemes may be efficient enough for large-scale deployment [18].

Transferability of access control tokens. As is the case with traditional capability-based access control systems, access control tokens in ClaimChain-based PKI (VRF values) are transferable. Transferability entails implicit trust assumptions on users, since malicious users may pass the access tokens to parties that were originally not authorized to read the claims. Research is needed to investigate whether designing a scheme in which access tokens are non-transferable, non-equivocable, and private at the same time, is possible.

Key revocations and chain compromise. The design of ClaimChains ensures that compromises are evident as forks of hash chains. We have not, however, discussed the best way to deal with detected compromises, nor with key revocations. As social validation policies, such protocols are context-dependent. Nevertheless, it is worth to note that ClaimChains support generic claims, which could include statements useful for this purpose, e.g. signals to other users about detected forks or revoked keys.

Interoperability with the host transfer medium. Some aspects of the use of ClaimChains in a decentralized deployment, can conflict with key operations of the system piggybacked to transmit them. For instance, take the introduction policy we use in our experiments, where the sender of an email provides to each recipient read access to information about other recipients. Assume a sender has communicated with a set of recipients, with all the messages being encrypted. If at some point a message from the sender comes in plaintext, yet the set of public recipient has not changed, then all recipients learn it is likely that a hidden (BCC) recipient was added.

Usability. Work is needed to understand the user experience and usability aspects of ClaimChains. PGP has long been criticized [19] for its usability issues, hindering its truly widespread usage; problem shared by CONIKS [20].

8. Conclusion

In this paper we have presented ClaimChains, a high-integrity mechanism for storing repositories of claims about users' and their contacts' state. We have introduced the concept of cross-referencing of hash chains, and described how cross-referencing can be used to build a decentralized PKI that enables users to establish trust in identity-key bindings, with better privacy properties than current deployments. The high integrity of ClaimChains, and the prevention of equivocation regarding bindings, is maintained using authenticated data structures, concretely hash chains and Merkle trees; and privacy is ensured using cryptographic access control and unlinkability mechanisms.

The computational, storage and bandwidth of ClaimChains depend on the setting in which they are stored and distributed. Yet, our empirical evaluation shows that its overhead is pretty small, and suitable for messaging applications. The different deployment options offer trade-offs between privacy, availability, and effectiveness of key distribution, for establishing end-to-end encryption. Our simulations on real data suggest that full decentralization, though having good privacy properties, comes at a high cost in terms of effectiveness.

Acknowledgments

This research is funded by NEXTLEAP project within the European Union's Horizon 2020 Framework Programme for Research and Innovation (H2020-ICT-2015, ICT-10-2015) under grant agreement 688722.

References

[1] C. Adams, S. Farrell, T. Kause, and T. Mononen, "Internet x.509 public key infrastructure certificate management protocol (cmp)." RFC 4210, 2005.

[2] M. S. Melara, A. Blankstein, J. Bonneau, E. W. Felten, and M. J. Freedman, "CONIKS: bringing key transparency to end users," in *24th USENIX Security Symposium, USENIX Security 15* (J. Jung and T. Holz, eds.), pp. 383–398, USENIX Association, 2015.

[3] P. R. Zimmermann, *The official PGP user's guide*. MIT press, 1995.

[4] B. Laurie, L. A., and E. Kasper, "Certificate transparency." RFC 6962, June 2013.

[5] M. Etemad and A. Küpçü, "Efficient key authentication service for secure end-to-end communications," in *International Conference on Provable Security*, pp. 183–197, Springer, 2015.

[6] E. Kokoris-Kogias, L. Gasser, I. Khoffi, P. Jovanovic, N. Gailly, and B. Ford, "Managing identities using blockchains and CoSi," in *9th Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs 2016)*, 2016.

[7] E. Syta, I. Tamas, D. Visher, D. I. Wolinsky, P. Jovanovic, L. Gasser, N. Gailly, I. Khoffi, and B. Ford, "Keeping authorities "honest or bust" with decentralized witness cosigning," in *IEEE Symposium on Security and Privacy, SP 2016*, pp. 526–545, IEEE Computer Society, 2016.

[8] M. Ali, J. C. Nelson, R. Shea, and M. J. Freedman, "Blockstack: A global naming and storage system secured by blockchains.," in *USENIX Annual Technical Conference*, pp. 181–194, 2016.

[9] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[10] J. I. Munro, T. Papadakis, and R. Sedgewick, "Deterministic skip lists," in *Proceedings of the Third Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms.* (G. N. Frederickson, ed.), pp. 367–375, ACM/SIAM, 1992.

[11] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman, "Protection in operating systems," *Commun. ACM*, vol. 19, no. 8, pp. 461–471, 1976.

[12] M. Y. Becker, C. Fournet, and A. D. Gordon, "Secpal: Design and semantics of a decentralized authorization language," *Journal of Computer Security*, vol. 18, no. 4, pp. 619–665, 2010.

[13] B. Klimt and Y. Yang, "Introducing the Enron Corpus," in *CEAS 2004 - First Conference on Email and Anti-Spam*, 2004.

[14] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney, "Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters," *Internet Mathematics*, vol. 6, no. 1, pp. 29–123, 2009.

[15] G. Danezis and C. Troncoso, "You cannot hide for long: de-anonymization of real-world dynamic behaviour," in *Workshop on Privacy in the Electronic Society, WPES 2013* (A. Sadeghi and S. Foresti, eds.), pp. 49–60, ACM, 2013.

[16] J. Shetty and J. Adibi, "Discovering important nodes through graph entropy the case of enron email database," in *3rd international workshop on Link discovery, LinkKDD* (J. Adibi, M. Grobelnik, D. Mladenic, and P. Pantel, eds.), pp. 74–81, ACM, 2005.

[17] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, "Private information retrieval," in *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pp. 41–50, IEEE, 1995.

[18] R. R. Toledo, G. Danezis, and I. Goldberg, "Lower-cost $\in$-private information retrieval," *PoPETs*, vol. 2016, no. 4, pp. 184–201, 2016.

[19] A. Whitten and J. D. Tygar, "Why johnny can't encrypt: A usability evaluation of pgp 5.0.," in *USENIX Security Symposium*, vol. 348, 1999.

[20] M. Melara, "Why making johnny's key management transparent is so challenging." https://freedom-to-tinker.com/2016/03/31/why-making-johnnys-key-management-transparent-is-so-challenging/. Last accessed: October 3, 2017.