NEXt generation Techno-social Legal Encryption Access and Privacy nextleap.eu

Grant No. 688722. Project started 2016-01-01. Duration 36 months.

DELIVERABLE D2.2

Federated Secure Identity Protocol

George Danezis[1], Bogdan Kulynych[2], Carmela Troncoso[2], and Marios Isaakidis[1]

[1]University College London
[2]IMDEA Software Institute

Beneficiaries: **UCL** (lead), IMDEA

Workpackage: **Science of Decentralized Architectures**

Description: This deliverable summarizes the work performed up to M12 withing task 2.2 regarding the development of a secure Federated Identity system. Identity systems enable communicating parties to establish correct cryptographic keys for their correspondents so that they can bootstrap secure communications. In this deliverable we envision a decentralized Identity System, where each user or device maintains repositories of statements regarding their own and their contacts' public keys. Its high integrity is maintained through hash chains, and equivocation is prevented through proofs of consistence and cross-hashing of the chains. We introduce the concept of cross-referenced hash chains (CRHC), and describe how they can be used to build a decentralized Identity System with support for address books, and describe the potential to provide strong privacy properties of such a system.

Version: Draft

Nature: Report (R)

Dissemination level: Public (P)

Pages: 29

Date: 2017-2-3

Project co-funded by the European Commission within the Horizon 2020 Programme.

1 Introduction

Following the Snowden revelations it became clear that, given the dependence of citizens, governments, and corporations on electronic communications, there is a strong need for highly secure end-to-end communications in which content cannot be accessed by third parties, to shield communications from mass surveillance systems, domestic or foreign.

Content confidentiality in end-to-end communication systems can be achieved by relying on asymmetric cryptography techniques to protect the transmitted data. These techniques work as follows: each user generates a pair of keys, namely the private decryption key, that is kept secret, and the public encryption key that can be shared publicly. Those who know the public key can use it to encrypt messages in such a way that only the owner of the respective private key can decrypt them. Public key cryptography can also be used for authentication and integrity purposes. The owner of the private signature key can digitally sign a message and those who know the public verification key can verify the signature and be assured that (i) the message has not been tampered with, and (ii) the message comes from the singature key owner.

Although asymmetric cryptography solves the confidentiality problem, it requires that users know each other keys, with high assurance. Public Key Infrastructures (PKIs) are systems which facilitate the operations that relate to public key exchange required by secure communications. PKIs create *trusted* bindings between an entity and their respective public keys, and provide means to recording, distributing and revoking public keys corresponding to principals.

Currently, these PKI infrastructures can be implemented in a number of ways that provide different levels of assurance, against different threat models. The most popular implementation, based on centralized certification authorities provides good availability and eases the key management operations (e.g. revocation or update). Yet, it places all trust on the PKI provider to maintain the integrity of the keys and its binding to users identities, and moreover places this provider in a privileged position to surveil users interactions. Recent proposals attempt to provide consistency via transparency logs, maintained and exchanged by the PKI providers and implemented as hash chains. Systems such as CONIKS make *equivocations* by an email provider about users' public key material detectable and accountable. Their adoption would be the next leap towards widespread end-to-end encryption transparent to the users; yet no email provider has deployed CONIKS as of today.

On the other side of the spectrum we find fully decentralized approaches, such as the Web of Trust, in which there is no single entity in which users put all of their trust, mitigating the concerns about such a central entity becoming corrupt, but that do not support easy updating and distribution of keys. Current implementations of web of trust also rely on centralized key servers to overcome those problems, which may observe requests for keys and deduce who is talking to whom – a serious meta-data privacy concern.

In this work we go beyond PKI, which is traditionally centralized, by considering the problem of providing an secure *Identity System* or *Address Book* – namely arbitrary high-integrity and authenticity bindings between names, as they are known by applications and users, and encryption or verification public keys necessary to secure communications. Decentralization, allows for increased robustness to adversarial central authorities, higher availability, as well as more options for supporting privacy.

The challenges of a modern Identity Systems as instantiated by Public Key Infrastructure

Modern communication protocols, such as the Signal protocol based on OTR [BGB04], do not encrypt all messages to the recipient's public key. Instead, to strengthen the security properties provided by the protocols, e.g., ensuring forward security, their approach is to derive symmetric short-term keys, called ephemeral keys, to encrypt messages. However, this does not remove the need to count on a PKI for authentication purposes.

However, no existing practical PKI provides adequate support to achieving the strong privacy properties that these modern protocols have as core goal in a decentralized manner. Either they rely on centralized authorities in a position of power to control the activities of users, or they lack flexibility to support fast updates of ephemeral keys.

In this report we outline the design of CLAIMCHAIN, a decentralized Identity System or PKI that relies on hash chains to ensure the integrity of keys, and on the novel notion of *cross references across chains* to further support the provision of evidence about the binding between keys and identities. In CLAIMCHAIN, each user or device has a claimchain, that acts as repository of statements regarding their own and their contacts' public keys. These claimchains are made available in a decentralized storage infrastructure where they can be queried so as to collect evidence about the correctness of the claims they hold, e.g., bindings of keys to identities, key revocations, etc. Our design provides support for different trust judgements that users may rely upon to verify the correctness of these claims. We also provide the flexibility for users to associate arbitrary attributes with names, and share them with others, making the system a generic decentralized address book.

This report is organized as follows. In the next section we revise existing solutions related to CLAIMCHAIN. Then, in Section 3 we provide an overview of our approach to building an Identity System, putting it in context with respect to existing PKI and describing the threat model and security properties it provides. We describe CLAIMCHAIN in detail in Section 5, we discuss in Section 6 how to enhance it to provide privacy protection, and finally we explain in Section 7 how CLAIMCHAIN can instantiate different identity management paradigms and how it can interoperate with existing solutions for secure end-to-end communication.

Positioning within NEXTLEAP

NEXTLEAP aims to understand the socio-technical extensions of decentralization into our digital rights. In that context, it poses questions around the philosophical foundations of the Internet, and studies the patterns that lead to the wide adoption of decentralized privacy enhancing tools. In a more applied approach, NEXTLEAP provides the technical insights on how to design and deploy systems that distribute trust, while assuring security and privacy properties via means of modern cryptographic techniques and mechanism design. Thus, this proposal for a Federated Secure Identity system contributes to NEXTLEAP's objective to conceive protocols and implement them as open source modules for grassroots developers to embed in their privacy tools.

The question of federated, secure identity is key to NEXTLEAP. The established approach of binding the notion of identity to arbitrary human-unreadable cryptographic keys does not mirror how we distinguish and trust individuals in our non-electronic interactions. Furthermore, identity management with cryptographic material is cumbersome in practice, frequently leading to key losses that get interpreted as identity tombstones. The existing practices for key discovery and validation leak the social graph of users and carelessly put significant trust in storage providers. At the same time, established reputation in systems such as social networks is not transferable, locking up users in proprietary platform silos.

Our reasoning for proposing practical ideas to address the above issues, is based on preliminary findings of NEXTLEAP field research. Through ethno-graphical studying of communities that work with decentralized communication protocols, it became apparent that secure messaging implementers tend to re-use existing protocols and software libraries, even if they are not officially standardized; that is the case, for example, with the Signal Axolotl protocol [ME16]. This further augments our commitment in pursuing close collaboration with open source communities so as to better understand their needs and for making sure that NEXTLEAP output is meaningful and usable to them, without getting impeded by time-consuming standardization processes. For that purpose, NEXTLEAP has identified projects that could integrate our protocols and has been actively engaging in their meetings and development processes. Consequently this gives us better insights on the unsolved problems of secure decentralized communications and the opportunity to have impactful contributions in real world tools, as described at the D5.1 NEXTLEAP report [KW16]. CLAIMCHAIN is trying to be complimentary to upcoming techniques for public key distribution, such as Autocrypt[1]. Further on, CLAIMCHAIN is maintaining compatibility with the existing APIs so as to be interoperable with current tools.

[1] https://autocrypt.org

The decentralization case study by Musiani and Ershomina [ME16] also highlighted the diversity of the secure messaging ecosystem, evaluating the degree of decentralization of the tools' underlying design. As a reference to designing decentralized privacy systems, NEXTLEAP has crafted a systematization report. On that, Troncoso et. al. share their insights on how decentralization is achieved, its benefits and drawbacks, with a special focus on how it affects the privacy properties of a system. The authors conclude that sound decentralized privacy designs require skills that lay around the intersection of distributed systems, cryptography, mechanism design, game theory and sociology [TDIH16]. CLAIMCHAIN is referring to the systematization report for evaluating the different instantiations of its decentralized components in regard to deployability and implicit privacy properties.

The first technical proposal of NEXTLEAP related to federated identity was UnlimitID [IHD16], a privacy enhancement to the established protocol for transferring membership, attributes and reputation across Internet platforms, that prohibits Identity Providers from impersonating users or tracking them as they are connected to third-party services. In the following months, CLAIMCHAIN will be revised based on user-centric feedback and comments from privacy enhancing tools developers to serve as the blueprint for implementing a decentralized identity system for establishing trust in a privacy-friendly way.

Going forward NEXTLEAP partners (WP5.1) have identified email as a key target for deploying our end-to-end encryption technologies that need to be supported by an identity system. More specifically the *Autocrypt* effort, which grew from NEXTLEAP's WP5 community building activities, embeds information about encryption keys into email headers, to build a decentralized key distribution system. Initially, this effort does not aim to authenticate keys, making it vulnerable to sophisticated man-in-the-middle attacks, and an Identity System is necessary to overcome this limitation. We discuss how CLAIMCHAIN could be instantiated within the architectural constraints of Autocrypt, in an extreme decentralized context, in Section 7.2.2.

2 Related work

This section provides a review of deployed PKI systems, that are the predominant centralized instantiations of Identity Systems, with a focus on those designs that aim in vouching for the authenticity of a binding between an email address and a PGP public key. Some systems follow a multi-tier strategy to assist in a best effort basis users to determine which of the public keys they have retrieved for an email address to use, if any at all. The last resort case, in which the user has to rely on a public key without validity assurances, is known as "trust on first use" (TOFU).

2.1 Validating bindings by trusting the PKI

SKS Keyserver: the established PGP PKI The default PKI most mail client agents will use for submitting and retrieving PGP public keys, is the SKS Keyserver[2]. Anyone can upload a binding without authorization and an SKS keyserver may hold many bindings for the same email to different public keys; therefore there are no guarantees about which of the bindings, if any at all, is authentic. The various SKS keyserver pools will synchronise their databases, thus making removing a binding very difficult. In order to protect from man in the middle attacks on the transport layer, as well as for assuring the confidentiality of the requests from a network observer, some SKS keyservers also handle HTTPS queries.

Trusting bindings via proof of email ownership The next generation of PGP PKIs introduced an access control mechanism based on proving ownership of the email address inbox; the PKI sends an email with a confirmation link that the user has to visit or reply to in order to confirm an operation. The PGP Global Directory[3] and the the Mailvelope key server[4] challenge anyone uploading a binding in a similar way. Bindings get published

[2]https://sks-keyservers.net
[3]https://keyserver.pgp.com
[4]https://keys.mailvelope.com

only after they have been confirmed and both of the services can maintain only one binding for an email address. Furthermore the PGP Global Directory and Mailvelope are more flexible with updating and removing bindings, even when a user has lost his private key. On the other hand, this model cannot protect users from malicious key servers who advertise fake bindings, malicious email providers who update the binding of a user without his consent, or adversaries who could intercept the verification emails via network traffic.

Trusted Notaries and Network Perspective auditing In faith that distribution of trust could be beneficial, federated PKI systems emerged. The Nyms Identity Directory[5] is building upon Trusted Notaries, PKIs comparable to Mailvelope (see previous paragraph) which upon verification of email ownership will also sign the binding with their certification key, and advises a Network Perspective auditing mechanism, according to which users can request from providers to request and cross-check the bindings they retrieve, so as to prevent malicious providers from lying about their users' bindings.

Nicknym[6], integrated into the LEAP Encryption Access project [SHKP16], is implementing the Nyms protocol in a multi-tier approach; the more assurances Nicknym can collect for a binding the higher trust is given to it. In addition, in pursuance of bypassing the X.509 Certificate Authorities, Nicknym is introducing the concept of the Federated Web of Trust (FWOT). Participating Trusted Notaries sign each other's certification key and their users are able to resolve the authenticity of a binding by following a friend-of-a-friend chain of signatures of notaries in an automatic way.

CONIKS CONIKS [MBB$^+$15] is aiming for seamless end-to-end email encryption by delegating to email providers to maintain auditable logs of their users' public keys. The purpose of the logs is the verifiability of the consistency throughout time of a binding of a user email address to the respective public key material. Effectively, equivocations about a user's public key by their provider are easily detectable and accountable.

The innovative elements of CONIKS that make it stand apart alternative proposals reside upon (a) the privacy properties by virtue of modern cryptographic mechanisms embedded in its data structures (b) capacity to scale and (c) being transparent to the users.

Under the hood, CONIKS logs could be seen as a modification to Certificate Transparency (CT) [LAK13] applied to PGP PKIs. They serve as an integrity measure that the provider is responding to all public key lookups throughout a period of time with the same result. Likewise CT, this is realized with a linked list of merkle trees hosted by each email provider, in which the leaf nodes correspond to the various email addresses of its users and encode/store their public keys. In order to verify the response of a key lookup, the provider presents a path in the latest merkle tree to either the leaf node that corresponds to the requested user (proof of inclusion), or to sibling nodes so as to prove that the requested user is not present in the merkle tree (proof of absence).

In prearranged intervals a CONIKS provider creates a new merkle tree encoding the current state of its users' public keys. That new tree also includes the hash of the previous tree, thus creating a hash chain similar to the Bitcoin blockchain [Nak08]. It should be clear though that the CONIKS blockchain/hashchain is independent to each email provider and incompatible to the Bitcoin or CT chains. Yet they are meant to provide the same function: an auditable append-only log that restricts modifications of past events. Furthermore, even if CONIKS providers publicly share their logs, they leak no information about the number of the users, their identities or their public keys. The mapping of a user to the corresponding leaf node is determined via a *verifiable random function (VRF)*, therefore can be computed only by the provider but can be verified with a short proof.

With regard to usability, CONIKS requires no manual operations by the users since the consistency verification checks can be integrated in mail clients. The authors do not advise a mechanism for cross-device key management or policies for lost or compromised keys. In addition, the state machine allows false positives that make difficult explaining to the users what they should do. More importantly, updating a user's public key is not an accountable action; the provider could as well have updated the key material of a user without their consent.

[5] http://nyms.io

[6] https://leap.se/en/docs/design/nicknym

CONIKS copes with these issues by providing a multilevel security scheme; tech-savvy users who have selected the strict pinning policy and lose their keys cannot make any updates to their corresponding merkle tree node.

Running CONIKS is a decision that falls solely to the email providers. The incentives for deploying an auditable and tamper-resistance PKI, that could enable end-to-end encryption by default, belong to the socio-economic sphere and, unfortunately, no providers have publicly expressed interest in doing so as of today[7]. At the same time, CONIKS found support in security-oriented messaging applications and could as well serve as a high-integrity mapping service for Bitcoin wallet public keys.

2.2 Social validation

The PGP Web of Trust The PGP Web of Trust (WoT) [Zim95] is not exactly a PKI itself, rather than a social trust establishment mechanism for the authenticity of PGP keys. It is complimentary to PGP and by nature decentralized. The concept is similar to the endorsements by Trusted Notaries in Section 2.1, however in the WoT any PGP key owner can vouch for the validity of a binding. Users use such vouches of their friends, and recursively those of the friends of their friends, to decide whether a binding is authentic.

The Web of Trust proved to be unusable in practice, reinforced by the difficulties in PGP keys management. Moreover it has been criticised with regard to privacy, since it is leaking the social graph of users.

Online Presence and Social Media Some PKI designs use attestations in social media profiles and other online directories to bootstrap a secure communication channel, under the basis that it is difficult for an adversary to gain access to an existing social media account, and impersonating entities by creating genuine-looking fake accounts is expensive.

In Keybase[8] users publish statements about their binding in various social media, website domains they control and the like, that others can check when validating the authenticity of the advertised keys. In addition, the Keybase provider data structure is very similar to the CONIKS chain of Merkle trees and when a new block is created, the hash of the head of Keybase's chain will be included in a Bitcoin transaction. This would prohibit a Keybase provider from equivocating about the latest state of their chain, since it is included in a public tamper-resistant log.

3 Approach

3.1 Problem statement

A public key infrastructure (PKI) instantiates an Identity System, and is a system that facilitates the management (storage, verification, distribution, etc.) of keys required to bootstrap secure communication [AL02]. Such infrastructure can be implemented in a number of ways. We now overview the existing options and their characteristics.

Often, since the paradigmatic example for PKI is the Certification Authority-based system backing up secure communications in the Internet, PKIs are assumed to be centralized. In this approach, one or more central authorities (so-called Certification Authority, CAs), usually organized as a hierarchy, can provide evidence about the correctness of *all* the keys in the system. A simplification of this scenario with only one CA is shown in Figure 1. Here When Alice provides a key to Bob, and in order to verify its correctness, Bob asks for evidence to the CA. In current deployments, such evidence is gathered in the form of, for instance, a Certificate Revocation List [CSF$^+$08] or an Online Certificate Status Protocol (OCSP) response [SMA$^+$13].

[7] https://freedom-to-tinker.com/2016/03/31/why-making-johnnys-key-management-transparent-is-so-challenging/
[8] https://keybase.io

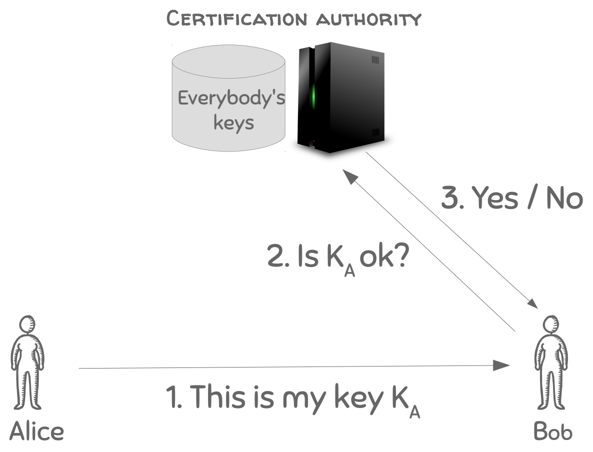


Figure 1: Centralized PKI

In a centralized PKI infrastructure it is easy to obtain good availability (only the CA has to have high availability), and deployment is easy since it fits very well the client-server paradigm. Yet, it requires full trust on the CA for confidentiality and integrity of the keys, and puts this authority in a position of privilege to perform surveillance on all events happening in the network.

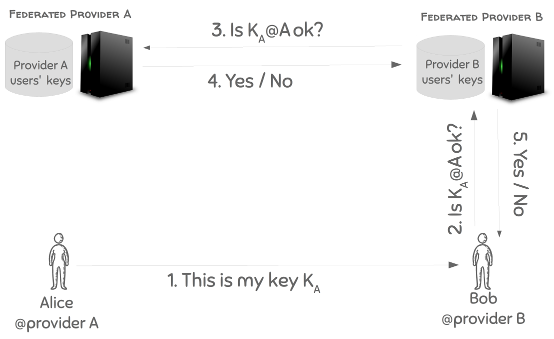
An alternative is, instead of having a unique source of authority that vouches for the correctness of all keys, have providers that can only attest to the correctness of the keys associated to them. Then, trust relationships are established at the provider level in such a way that users can verify each other keys by trust transitivity. This corresponds to the federated paradigm, illustrated in Figure 2. In this example Alice is associated to Provider A, and Bob is associated to Provider B. When Bob needs to verify Alice's key, he turns to his provider for evidence. Then the evidence can be collected in two possible ways: i) recursive (Figure 2(a)), where Provider B will gather evidence from Provider A in the name of Bob; or ii) iterative Figure 2(b)), where Provider B refers Bob to trusted Provider A to gather the evidence himself.

In a federated PKI, availability again relies on providers, thus, it is easy to achieve. In terms of confidentiality trust is more distributed than in the centralized case, but the providers are still in a position of power with respect to users.

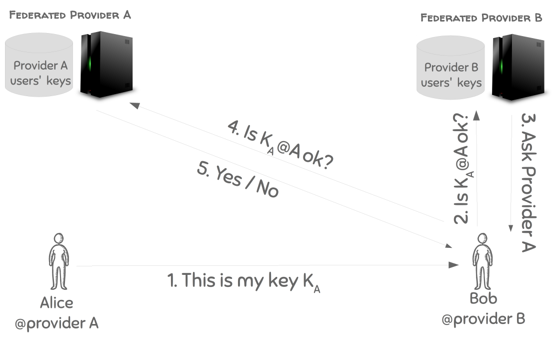
The third alternative is to fully decentralize the source of authority, like in the Web of Trust [CDF$^+$07] where the authenticity of the binding between a public key and its owner is attested to by other users that sign the certificate. Such attestation usually happens offline, i.e., users sign each others' certificates prior to the verification. This approach presents limitations in terms of flexibility and update of keys, which makes it not suitable for modern applications such as messaging where keys may need to be changed very often.

The decentralized verification process, however, could also be done online, as shown in Figure 3(a). In this case Bob verifies Alice's key by gathering evidence from other peers (in the example Charlie and David), and then uses the collected evidence to decide whether Alice's key is correct.

This decentralized approach presents advantages from a privacy/trust point of view, as there is no single entity in the system that can observe the actions from many users. On the other hand, full decentralization poses three fundamental problems: first, it requires the availability of other users to provide evidence; second, it is difficult to

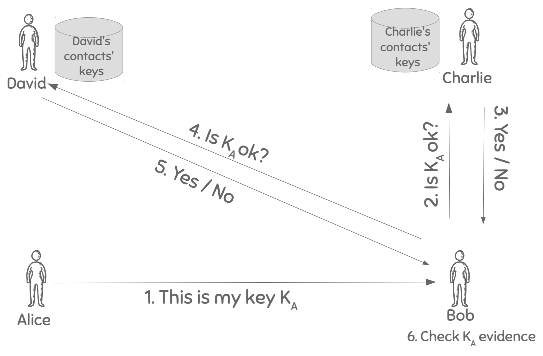


(a) Federated recursive

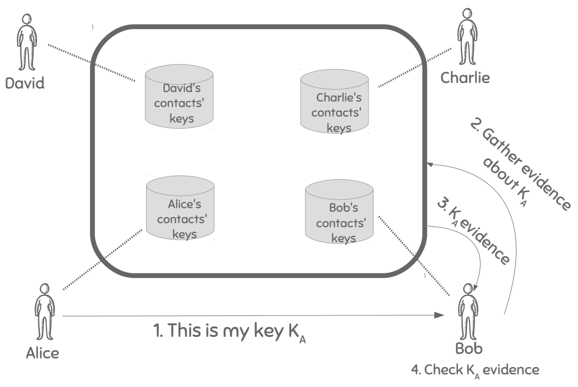


(b) Federated iterative

Figure 2: Federated PKI



(a) Decentralized



(b) Decentralized with high availability

Figure 3: Decentralized PKI

ensure that all actors in the system have a consistent view of the state; and third, it requires the setup of a policy to decide the veracity of the key upon receiving the evidence from peers.

In the remainder of this document we describe CLAIMCHAIN, an infrastructure that solves the first two problems, i.e., it enables the secure collection of evidence in terms of integrity and availability, in an efficient manner. For this purpose we rely on the existence of high-availability storage infrastructure that ensures that evidence of all users is always available for consultation (represented in thick line in Figure 3(b)). Such storage can be implemented in a centralized way, e.g., in a Cloud like Amazon Web Services, or in a distributed manner using, for instance, a DHT to index content, or local storage on peers combined with a gossip protocol. Additionally, CLAIMCHAIN can be configured to provide strong privacy guarantees towards the availability storage as well as to other users.

We now provide a concrete description of the threat model considered by CLAIMCHAIN and the security and privacy properties we aim at achieving; and we outline the main ideas behind our design that we develop in subsequent sections. We consider that the definition of the verification policy is an orthogonal problem to the one of secure evidence collection, and thus we limit the design of CLAIMCHAIN to support the evidence collection such that different policies can be implemented depending on the environment in which CLAIMCHAIN is deployed.

3.2 A hashchain-based solution

Our solution to the decentralized or federated Identity System problem is called CLAIMCHAIN. CLAIMCHAIN builds a per-device high-integrity store of evidence relating to each user's belief about their own keys, and other people key bindings. That store is stored both locally but also distributed over a decentralize key-value store ran by *storage providers*, for others to be able to access – potentially under some restrictions to preserve meta-data privacy. The evidence may be updated over time, to include new keys, new beliefs about binding between users and keys, or to revoke out of date or compromised keys.

The integrity of the store is preserved by using a set of high-integrity data structures, based on hash chains and Merkle trees. These allow users to verify the correctness of their own store, and well as succinctly share with others a witness (called a chain *head imprint*, or *head*) that summarises all the state of their evidence. Such heads may be stored with semi-trusted providers, gossipped between users, or attached to messages to provide peers with an up-to-date record of the evidence others hold in the network. Such remote heads may be sealed into a user's hash chain to signal to others they are aware of this state – a process we call *cross referencing*.

Once a user accepts a binding between a name and a hash chain *head* as genuine, and cross-reference their chain, they are able to discover keys that may be used to authenticate the referenced user or encrypt messages to them. Furthermore, future updates to the remote users' stores are self-certifying. However, accepting initially a binding between a name and a hash chain head is an activity that inherently requires a degree of trust in some evidence provided by third parties. CLAIMCHAIN is agnostic as to the way such bindings are resolved, and provides evidence that may be used to implement traditional centralized PKI, web of trust models, friend of friends models, as well as any other policy that users trust to accept new bindings.

CLAIMCHAIN aims at providing the following security and privacy properties. We stress that not all of them need to be achieved at the same time. Different configurations of CLAIMCHAIN result in different sets of achieved properties.

1. **Security properties.**
    - **Non-equivocation.** Users can not equivocate on head imprints and public keys of other nodes.
    - **Integrity of chain content.** No one can tamper with or modify chain content if the chain head imprint is known.
    - **Authorized and authenticated chain updates.** Only the authorised user may update the chain content.

- **Detectable impersonation attempts.** If there is an impersonation attempt, which is executed as a fork of a victim's chain, it may eventually be detected and the user should be able to convince others that their updates are genuine.
- **DoS attacks protection.** It should be "difficult" to abuse the communication channels inside the system on a massive scale.

2. **Privacy properties**
   - **Confidentiality of relationships.** No user or provider can learn the contacts of another user without authorization.
   - **Confidentiality of meta-data.** No user or provider can learn the meta-data stored in the storage of other users without authorization.
   - **Query privacy.** The content of queries performed on the storage providers should be private from other users and storage providers.
   - **Key privacy.** Passive or active harvesting of public keys should not be feasible.
3. **Interoperability**
   - **Mail clients.** It should be possible to integrate CLAIMCHAIN with existing mail clients via the GnuPG agent.
   - **Messaging systems.** It should be possible to integrate CLAIMCHAIN with with existing messaging systems, such as the Tor Messenger.
   - **Federated deployment.** Deployment options compatible with a federated network of providers should be available, without relying fully on those providers for authenticity.

CLAIMCHAIN can provide the security and privacy properties explained above under different adversarial models. From the point of view of a user there are three possible adversarial entities: i) the CLAIMCHAIN storage provider where users can potentially mirror their stores of evidence remotely, and thus has access to all the information comprised in this evidence; ii) other users that may query the storage provider about the state or content of different evidence stores, or that may include false evidence in their stores of evidence; and iii) third parties that have access to CLAIMCHAIN that can either eavesdrop any communication within the system or interact with CLAIMCHAIN's components. We consider the following cases for these three adversarial entities:

- **Honest.** The entity is fully trusted. Effectively this entity is not adversarial and there is no need to implement protection mechanisms against it.
- **Honest but curious.** The entity does not try to equivocate to users about the content of the storage, nor tamper with this content, but given the information it can access (e.g., eavesdropping on the communication or interacting according to CLAIMCHAIN's protocols) tries to learn information about the users. For instance, it may want to reconstruct the social graph by reading the information users store, or by analyzing their requests regarding information stored by other users.
- **Malicious**. The entity not only tries to learn information about users using the data that it can obtain through honest means, but may also deviate from the protocols to learn more information. Furthermore, it may also try to equivocate users about the storage content (e.g., a users' state) to impersonate other users, or to perform a Denial of Service attack.

We note that in the detailed description of the CLAIMCHAIN we provide in Section 5 there is another entity, the STM, that keeps track of the latest state of users' evidence stores. We note that, as we explain below, given that our approach for evidence storage relies on cryptographic protections for ensuring the integrity of the evidence stored and avoid its corruption, we do not consider that STM needs to provide the correct latest state, since this correctness can be validated independently. However, we note that a malfunctioning STM, may prevent the users from check other users evidence stores, effectively producing a denial of service. Finally, we note that given its role in the system this entity does not pose a threat for users' privacy.

Finally, we assume that the cryptographic primitives used are sound, the adversary is constrained in terms of computation power and cannot break the properties of cryptographic primitives, and we consider secure key generation and operating system security are out of scope. However, we do consider that users' cryptographic key material can be compromised, e.g., due to device theft, and provide mechanisms to avoid it.

3.2.1 Hashchains and The Blockchain

The original purpose of blockchain, as proposed in Bitcoin [Nak08], is to provide a *consensus* mechanism for a global, transparent, append-only log. This log is represented as a linked list of blocks, with each block imprinting a cryptographic hash of the previous one. We refer to the first block of the log as the *genesis block*, and to the last known block as the *head* of the chain. In order to append a new block, the participating nodes need to solve a computationally expensive task, a process called *mining*; the first node to find a solution creates and broadcasts a new block and gets a reward, which in turn acts as an incentive for nodes to keep mining. This consensus mechanism can be compared to a ballot, where each CPU cycle spent for solving the task corresponds to a vote. The consistency of the history of the log is assured because it is impossible to change a block without mining the succeeding blocks all the way to the head. Different versions of the chain spanning from a valid block are called *forks*; when forks occur, participating nodes accept the longest chain as the valid history of the log.

Furthermore, blockchains are designed to be a very efficient *data structure* for high integrity verifiable logs. The interior scheme of blocks, incorporating merkle trees, makes the verification of logged transactions a matter of verifying the hashes of a small subset of the tree nodes. In addition, auditors need to store only the hashes of the blocks in order to be able to check the consistency of the chain; nodes that were offline can take advantage of this and catch up with the head of the chain simply by retrieving the intermediate blocks since the last known good block.

The concept of blockchain has been applied to various problems as a feasible technique for decentralization: cryptocurrency ledgers, secure multi-party computation, smart contracts, identity management and so on. The majority of such blockchain-backed systems cannot cope with the limitations of the Bitcoin blockchain and have designed their own independent chains[LAK13, BCG+14]. Variations of the mining and broadcasting mechanisms, or of the structure of the blocks, intend to solve scalability issues, the waste of computational power, the delay in getting a transaction registered in the log, the need to log a generic datastructure, or to obscure logged data for privacy purposes [GM16, MBB+15].

In practice, most applications use the blockchain as a *global* log, accepted by all nodes as the system-wide state. This results in systems where a local action, such as an event occurring at one of the peers and that needs to be logged, must be broadcasted to and be processed by all mining nodes. Therefore the bandwidth, storage and computational requirements for all mining nodes are significantly increased.

Based on what we have learned from building decentralized systems, we can expect that systems which keep track of local changes on a node level, and eventually aggregate them when needed until a global state is reached, have an advantage at least in terms of scalability [GM16]. Efforts to "individualize" blockchains yet provide a way to perform transactions among them lead to the appearance of *sidechains* [9]. Still, sidechains are independent from each other and there is no provision for performing common operations.

3.2.2 Why hashchains to build a decentralized or federated Identity System

ClaimChain uses high-integrity data structures such as hash chains and Merkle trees to represent the stores of evidence of users, and update them. These data structures ensure that whoever has a short witness (usually

[9] https://blockstream.com

a single hash value) can verify the full contents of the store, even if it is remotely hosted on an untrusted storage provider.

We leverage this feature of hash chains to replicate the user's evidence on network or local peer storage providers, so that it may be shared with other users. We also make use of the fact that a short witness to the full state of a user's store can be efficiently exchanged to include references to other people's stores through cross referencing. This provides a high-integrity association between two users or two devices of the same users. Furthermore, since this association is evidenced in a user's store that may be shared with others, it may be used to help other users establish an association between names and keys, e.g. in the case of a friend trying to discover the key of a common friend.

The data structures we use allow for self-certified updates. This does aways completely with the need to trust storage providers to either verify past claims of users, or future claims. Given a head imprint associated with a remote user, these can be verified despite potential attempts by storage providers to distort the evidence.

The use of hash chains also allows after the fact detection of potential key compromises. In case a hash chain is compromised, and the ability to extent it is usurped by an adversary, we expect a 'fork' to appear: on one side the adversary will include new false claims, and on the other the genuine user would extend it revoking the compromised keys. In the very least this allows others to detect that the chain is compromised. We also propose additional mechanisms, using revocation keys, to allow the owner of the chain to assert their ownership of the store, and guide honest users towards the honest side of the fork.

Thus the use of hash chains and other high-integrity primitives allows users to reduce the integrity of their current set, and future set, of claims to be summarised by a short witness – the *head imprint*. Thus establishing a secure association between that head imprint and a name allows others to perform fast key updates and revocations, as well as using some of the evidence of other users to make trust decisions about new associations, using an arbitrary trust management rule set. For this purpose we foresee users cross referencing others' hash chains.

4 Preliminaries

Our system relies on properties of *hash functions* to reduce integrity checks of a large, growing, set of statements about names and keys, to a simple integrity check of a short fixed size sting of bits – the *head* of a *hash chain* or *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 users inexpensive sharing and replication, while achieving high degrees of non-equivocation, and allows for easy replication and federation of the Identity Information. Further, cryptographic primitives, such as *digital signatures* allow for self-authenticated updates, and *encryption* allows us to implement some privacy features.

4.1 Cryptographic primitives

Let $\lambda > 0$ be a security parameter, $\mathcal{M} \subset \{0,1\}^*$ a plaintext message space, and $\mathcal{C}_S \subset \{0,1\}^*$ a ciphertext space of an encryption scheme S. We are using the following primitives in our basic construction:

- Cryptographic hash function $H : \mathcal{M} \to \{0,1\}^\lambda$. The function H should act as an ideal *Random Oracle*.
- Asymmetric signature and signature verification algorithms $\mathsf{SIG}_{\mathsf{sk}} : \mathcal{M} \to \mathcal{C}_{\mathsf{SIG}}$ and $\mathsf{VER}_{\mathsf{pk}} : (\mathcal{M} \times \mathcal{C}_{\mathsf{SIG}}) \to \{0,1\}$. A key generation algorithm can be used to derive keys $\mathsf{GEN} : \lambda \to (\mathsf{sk}, \mathsf{pk})$. We assume the algorithm for signatures is secure against *existential forgery*.

We denote string concatenation as $\|$. We assume that the hash of a tuple $H((t_1, t_2, ..., t_n))$ is a $H(t_1 \;\|\; t_2 \;...\; \|\; t_n)$.

4.2 Authenticated hash chain from indexable skip lists

A hash chain, also called *blockchain*, is an ordered sequence of *blocks* $\mathbf{B} = \{B_0, B_1, ..., B_n\}$. Each block stores one or more ordered statements, which may be indexed by the index of the block. Whereas a classical blockchain has the underlying structure of a linked list, we are using a construction that is based on a deterministic skip list [MHKS14, Pug90]. Each block contains links to one or more past blocks. Our construction makes proofs of inclusion shorter in size and less expensive to check than a traditional linked list.

Each block $B_i = (X_i, F_i, \sigma_i)$ in our chain comprises a payload X_i, set of hashes of some subset of previous blocks $F_i = \{(j, H(B_j)) \mid j \in J(i)\}$ for a set of indices $J(i)$, such that $\forall j \in J(i) : j < i$, and a signature $\sigma_i = \mathsf{SIG}_{\mathsf{sk}_i}(H(X_i \parallel F_i))$ for some key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$:

$$B_0 = (X_0,\ \varnothing,\ \sigma_0 = \mathsf{SIG}_{\mathsf{sk}_0}(H(X_0 \parallel \varnothing))) \qquad \text{(genesis block)}$$
$$B_i = (X_i,\ F_i,\ \sigma_i = \mathsf{SIG}_{\mathsf{sk}_i}(H(X_i \parallel F_i))), i > 0 \qquad \text{(regular block)}$$

The inclusion of a signature per-block allows the authentication of subsequent blocks, subject to the appropriate keys being used. The selection, validity and revocation of those keys will be discussed later in this report.

As discussed earlier, an imprint of a blockchain is an important component in the design of our PKI, since it is a unique identifier of the blockchain's state at any particular moment in time. The imprint of the blockchain that has B_i as the latest block is simply its hash and index: $(i, H(B_i))$.

Each block contains the imprint of several previous blocks in the set $F_i = \{(j, H(B_j)) \mid j \in J(i)\}$ that allows someone that knows B_i to efficiently authenticate past blocks forming part of the chain. The indexes $J(i)$ are chosen to mirror the structure of a skip-list:

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

For example $J(127) \equiv \{64, 96, 0, 112, 120, 124, 126\}$ and $J(128) \equiv \{96, 64, 0, 112, 120, 124, 126, 127\}$. These sequences $J(i)$ have valuable properties:

- Notionally the numbers fall on the 'tick marks' of a binary ruler, one per height of tick-mark.
- The sequence for $J(i)$ ensures that given an index $j < i$, it contains a a number that is at least $(i-j)/2$ closer to j than i.
- The size of the sequence $J(i)$ is of length $O(\log i)$.
- Subsequent $J(i)$ and $J(i+1)$ share a large number of common elements.

Given those properties, we can achieve efficient proofs of inclusion in the sequence as we study in the next section.

4.2.1 Supported operations

We define a number of operations on the hash chain described above.

ADD$(B_i, X_{\{}i+1\}, \mathbf{sk}_i)$ Appends the payload $X_{\{}i+1\}$ to the existing chain B_i and returns the new block $B_{\{}i+1\}$. This operation takes time $O(\log i)$ in terms of all operations, or $O(1)$ terms of cryptographic operations only.

GET(B_i, j) Returns a block at index j, along with its evidence of inclusion $\pi_{i \to j}$. The computational cost of this operation is $O(\log(i))$ operations, and the size of the bundle of evidence is $O(\log^2 i)$.

EXTENDS?(B_i, B_j) This operation returns true if block B_i is an extension of the chain B_j for $i > j$. It is equivalent to checking that $\mathsf{GET}(B_i, j) = B_j$.

CHECK$(B_i, B_j, \pi_{i \to j})$ Returns true if the evidence $\pi_{i \to j}$ is a proof that B_j for $j < i$ is part of the chain B_i. This operations takes $O(\log i)$ cryptographic (hash) operations.

The bundle of evidence $\pi_{i \to j}$ is a set of blocks B_x including B_i and B_j such that the imprint of each block is included in another block in the evidence. Thus the blocks in the evidence form a chain from B_j to B_i that may be checked for integrity by checking the inclusion of their imprint in the subsequent block. Furthermore, such sequence of evidence is guaranteed to be of length $O(\log i)$ blocks maximum.

4.3 Sorted Merkle Trees

Hash chains provide an efficient way to prove that a block forms part of the chain at a particular previous position. However, we also require a data structure that allows for efficient proofs of inclusion as well as non-inclusion of specific data. Sorted Merkle Trees [Mer89] provide such facilities, and we briefly describe them here.

A Sorted Merkle Tree is the high-integrity equivalent of a sorted binary tree. It is composed of two types of structures: Leafs $L(x)$, holding a data item x, and branches $B(y, W_i, W_j)$, where W is the imprint of either a another Branch or a Leaf. The imprint of a Branch or a Leaf is defined as the hashed unambiguous representation of the Branch or Leaf using a secure cryptographic hash function.

The key invariant of the Sorted Merkle Tree concerns branches $B(y, W_i, W_j)$. Consider the set of all values held at Leafs hashed as part of the imprint W_i as V_i and all values hashed as part of W_j as set V_j. Then for all $x \in V_i$ it must hold that $x \leq y$, and for all $x \in V_j$ it must hold that $x > y$. Thus the leafs to the right of the branch hold values smaller or equal to x and to the left values greater than x.

It is easy for someone with a 'root' element of the tree to convince themselves that an element is included or not included in a leaf of the sub-tree. A proof of inclusion of a value v simply involves returning the leaf $L(v)$ and all Branches from that leaf to the root of the tree. Checking it simply requires recomputing the imprints of all elements at each level and checking for their inclusion in the branch a level above them. A proof of non-inclusion involves returning the proof of inclusion of the element preceding and following v in the Sorted Merkle Tree. Those two proofs can be checked to ensure no leaf $L(x)$ is attached to the tree. Those proofs take $O(\log n)$ computational effort, where n is the number of leafs in the tree, and are also of $O(\log n)$ size in the number of Branches or Leafs necessary to construct or check the proof.

Maintaining the tree balanced under multiple additions and deletions of elements is important to ensure the above time and space complexity bounds. To achieve this the sorted tree may be subject to additions or deletions of leaf values following the algorithms of AVL-Trees, Red-Black Trees, or any other balanced sorted tree strategy.

4.3.1 Supported operations

We define a number of operations given a Sorted Merkle Tree. By convention we consider the element R_i is the root of the tree:

ADD(R_i, x) Given the root of a Sorted Merkle Tree R_i this operation returns a new root R_{i+1} of a Merkle Tree containing the same set of elements as R_i with the addition of element x. This operation takes time $O(\log n)$ in the size n of the number of elements in the tree.

REMOVE(R_i, x) Given the root of a Sorted Merkle Tree R_i this operation returns a new root R_{i+1} of a tree containing all elements, except element x. As for ADD this operation take time $O(\log n)$ in n the number of elements in the tree.

IN(R_i, x) Returns true if the element x is in the tree with root R_i, along with a set of elements π_x that act as evidence of that fact. Otherwise returns false, and a set of elements $\pi_{\bar{x}}$ proving that fact. The evidence is composed of a $O(\log n)$ number of elements if the tree is balanced.

CHECKIN$(R_i, x, \pi_x$ **or** $\pi_{\bar{x}})$ Given the root of a tree R_i returns true if the proof of inclusion π_x or non-inclusion $\pi_{\bar{x}}$ of element x are valid, otherwise returns false. This check takes time $O(\log n)$.

5 Basic ClaimChain framework

We consider that there are three types of entities in ClaimChain Identity System: *nodes* (users of the system), *storage infrastructure* (SI), and *state tracking mechanism* (STM). Nodes have claimchains, one per device, high integrity mechanisms that store claims about themselves or other nodes. These claimchains are stored in SI, and a reference to their latest state is stored at STM. We assume that nodes can interact with SI and STM. Direct communication between nodes is not required for the functioning of the system, but as we explain in Sect. 5.3, it can be leveraged to implement the functionality offered by the STM.

ClaimChain primary goal is to enable nodes to reliably exchange cryptographic material with high-integrity binding to identities. Additionally, ClaimChain allows nodes to vouch in a verifiable manner for claims of various nature, that is, not only concerning key material, but identifiers in other systems, persistent messages, etc.

Our system assumes some common parameters are shared for all nodes and components of the subsystems. We leave certain parts of ClaimChain— the so-called *policies* of nodes — underspecified and only describe the limits to their flexibility given by our constructions. Policies should be instantiated by each node individually according to their preferences, since different policies yield different trade-offs in privacy and security properties.

In this section we present ClaimChain's *basic* framework, and we discuss in Section 6 and Section 7 possible enhancements and instantiations.

5.1 Claimchains

The basic building block in our Identity System is a *claim*: a commitment to a piece of information. This piece of information can refer to the node itself (e.g. key material), to other nodes (e.g., the state of other node), or any other information deemed relevant. Every node maintains an append-only log of *statements*, each statement being either a claim or a revocation of a previous claim. By design, the log contains detailed information about each statement needed for its verification (e.g., digital signature, cryptographic material, or other meta-data). A node also maintains auxiliary data structures that allow for efficient search in the statement logs: a set of all claims (*inclusion set*), and an *inverse claim index*. Together with the statement log, these data structures form a node's *claimchain*. We now explain each of the components in detail.

In the following, we use the terms imprint of a claimchain and imprint of a hashchain interchangeably, for claim-chain meaning the imprint of the corresponding hashchain $\mathbf{B}$ in the claimchain.

Statement A statement S_i, shown in Figure 4(a). It is represented as a record containing: the *statement type* K, the statement *content* D, an optional *pointer* P, and *statement metadata* M. The meaning of each of these fields is discussed in Section 5.1.2.

Statement log The statement log, which we denote as $\mathbf{B}$, is implemented as an authenticated hash chain based on skip lists. According to the definition given in Section 4.2, each block B_i in $\mathbf{B}$ contains imprints of some previous blocks F_i, a payload X_i, and a signature $\sigma_i = \mathsf{SIG}_{\mathsf{sk}}(X_i \mid\mid F_i)$ for some asymmetric key pair $(\mathsf{sk}, \mathsf{pk})$.

In ClaimChain, a block's payload X_i consists of four components (see Figure 4(b)):

- An ordered list of statements $\{S_0, S_1, \ldots, S_{m-1}\}$. We denote the j-th statement in block B_i as $B_{i,j}$, i.e., statements in the log can be *pointed to* by a tuple of indices (i, j).

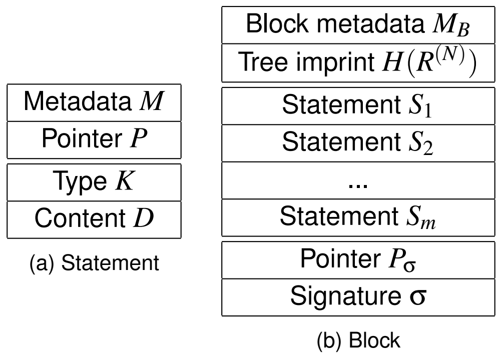


Figure 4: Logical structure of statements and blocks in the hash chain

- A *signature pointer* P_σ, which is the index of the statement containing the public key pk corresponding to the signature key sk used to sign the block.
- The latest root imprint $H(R)$ of the inclusion tree $\mathbf{T}$, which we introduce below.
- The *block metadata* M_B. We do not define any requirements on M_B, metadata can be any kind of data included in blocks and statements that is not crucial to the functioning of our basic framework. Metadata could be required for implementing nodes policies, e.g., it could include timestamps.

Inclusion set The set of all claims, denoted as $\mathbf{T}$, implemented as a sorted Merkle tree. Each leaf of the tree contains a hash $H((i,j) \mid\mid K \mid\mid D)$ for some claim (K,D) that is present in the log at index (i,j). We denote trees corresponding to the state of the claimchain at its i-th block as $\mathbf{T}_i$, and corresponding root as R_i.

Inverse claim index An index I is a lookup table that given the statement type and content (K,D) returns a pointer to the *latest* occurrence of the claim in the log. For the purpose of simplifying the discussion, we assume that the inverse index I is always correct.

5.1.1 Cryptographic key material

We consider that nodes can store and commit to the following cryptographic key material in their claimchains:

Verification keys Asymmetric key pair used to generate and verify block and statements signatures. We denote a verification key pair as $(\mathsf{pk}_\mathsf{V}, \mathsf{sk}_\mathsf{V})$.

Recovery keys Asymmetric key pair used for recovering from a loss of a verification key, denoted as $(\mathsf{pk}_\mathsf{R}, \mathsf{sk}_\mathsf{R})$. This key permits revoking a public verification key pk_V without having access to its secret part sk_V.

5.1.2 Statement types

We differentiate several types of statements that nodes can include in their logs. A type is defined by the kind of content D, the pointer in P, and the key used to sign the block containing the statement, as summarized in Table 1.

Initial verification key Own initial public verification key pk_V.

Initial recovery key Own public recovery key pk_R.

Verification key update Revocation statement for a previously committed own verification key. Contains the new verification key pk'_V and must be signed with the verification key being revoked sk_V.

Table 1: Statement types. (r,l) is an index of a non-revoked verification key. pk' refers to a new (updated) public key.

Statement type (K)	Content (D)	Pointer (P)	Block sk
Verification key	pk_V	–	sk_V
Recovery key	pk_R	–	sk_V
Encryption key	pk_E	–	sk_V
Verification key update	pk'_V	–	sk_V
Recovery key update	pk'_R	–	sk_R
Verification key recovery	pk'_V	(r,l)	sk_R
Revocation of claim	$H(X_{i,j} \mid\mid D_{i,j})$	(r,l)	sk_V
Block cross-reference	$\left(i, H(B_i^{(N)})\right)$	–	sk_V
Block self-reference	$(i, H(B_i))$	–	sk_V
Generic claim	$*$	$*$	sk_V

Verification key recovery Revocation of a verification key using the recovery key. The block containing this statement must be signed with the recovery key sk_R.

Recovery key update Revocation of a recovery key. The block containing this statement must be signed with the previous non-revoked recovery key sk_R.

Other claim revocation Revocation of any kind of previously committed claim. Contains the hash of the statement $H(B_{i,j})$ being revoked, and a reference (i,j) to the statement.

Cross-reference The imprint of a claimchain belonging to node N at block $B_i^{(N)}$.

Self-reference The same as cross-reference where the node N is self. Although cross-reference and self-reference statements have identical structure they can be interpreted differently, e.g., self cross references may constitute stronger evidence when making decisions about a claim.

Generic claim Any kind of claim about the node, other nodes, or anything else. Can be used by nodes in some individual policy instantiations, or for any other purpose outside of the scope of ClaimChain functionality. Some examples of generic claim content:

- Encryption key. A public key that can be used to encrypt messages to the claimchain owner.
- Signal prekey bundle. A set of Signal X3DH protocol prekeys, used to establish secure communication sessions [MP16].
- Links to claimchain owner identities at other services, e.g. email addresses, social media accounts.

We call verification key update, verification key recovery, recovery key update, and other claim revocation *revocation statements*. Of these, we call recovery key revocation and verification key recovery *recovery statements*. Note that we only allow one signature per block, hence recovery statements must go in separate blocks. The signature pointer P_σ in a block always contains a reference to the statement declaring public signature key pk that corresponds to the secret key sk used to sign the block, be it recovery or verification key.

5.1.3 Validation

We say that a claimchain is *valid* on a subpath $\pi = (B_{i_0}, B_{i_1}, ..., B_{i_k})$ of its statement log when all of the following conditions are met:

Block integrity For each two successive block pairs in π, B_k and B_{k+1}, the finger of B_{k+1} pointing to k matches the imprint of B_k: $(k, H(B_k)) = F^k_{(k+1)}$.

Key pointer Each block contains a pointer to either a recovery key or a verification key.

Block authenticity For each block $B_k \in \pi$, the signature σ_k can be successfully verified using the key pk referenced in B_k's P_σ. Moreover, pk must be the latest non-revoked public key of its kind up until block B_k.

Tree correctness For each two successive blocks B_k, B_{k+1}, the root of the inclusion tree R_{k+1} in B_{k+1} matches the root of a tree constructed adding the contents of B_{k+1} to the previous tree $\mathbf{T}_k$.

Recovery statement priority If a block B_k contains a recovery key revocation statement, or a verification key recovery statement, it is the only statement in the block.

Correct key usage Blocks containing recovery statements are signed using recovery keys, all other blocks are signed with verification keys.

Revocation consistency If a claim at $B_{i,j}$ is revoked in statement $B_{i',j'}$, it cannot have been revoked before $B_{i',j'}$, i.e., claims can only be revoked once.

Key revocation/update correctness Claims referring to verification or recovery keys cannot be revoked, but must be always updated.

Verification key recovery consistency The verification key recovery statement has a pointer to a non-revoked verification key.

A path in a claimchain can be validated on average in linear time in the length of the path using a two-pass algorithm for validating the claimchain. The first pass traverses the statement log top to bottom and checks integrity, authenticity, and recovery statement priority, optimistically assuming the correct usage of the signature keys. The second pass traverses the path bottom to top, checking the correctness of trees, revocation consistency, and correct key usage.

5.2 Claimchain storage infrastructure

We assume there exist high-availability centralized, decentralized, federated or local key-value stores, that are uniquely identifiable and accessible by the nodes. These stores are entities that form CLAIMCHAIN's *storage infrastructure*, SI, and each store has a unique identifier $\mathrm{ID}_{\mathrm{SI}}$. An SI entity in its most basic form provides the following key-value store API:

GET.OBJECT(k) Returns the object at key k.

UPSERT.OBJECT(k,v) Writes object v to the storage under key k.

Keys in SI can be arbitrary strings in $\{0,1\}^\lambda$. We store every hash chain block and inclusion tree node as objects using their own hashes as keys. For inverse claim indices, we augment the key to include the hash of the genesis block of the claimchain containing the indexed claim. The reason is that, as claimchains reference each other blocks, claims (K,D) may not be globally unique. More formally, for a claim (K,D) included at (i,j) in a hash chain $\mathbf{B}$, the key to store the pointer is $H\left(H(B_0) \mid\mid K \mid\mid D\right)$, and the object is the pointer (i,j). We note that both those type of keys are collision resistant – meaning an adversary cannot find a different value than the genuine one for a given key.

These API functions then can be used in implementations of operations on hash chains and Merkle trees to retrieve block and node content from the storage infrastructure. This allows to refer to whole claimchains by heads of their hash chain, since every block contains data that is sufficient to retrieve other blocks, the inclusion tree nodes, and inverse index from an SI entity.

The simplest lookup service without authorization or authentication (any CLAIMCHAIN node can write and read any objects) is enough for ensuring all of the desired security properties (see Section 3.2). Additionally, CLAIMCHAIN can incorporate other protection mechanisms to increase the privacy properties, such as:

- Authorization and access control. Restrict certain nodes from accessing claimchains or parts of claimchains of certain other nodes.
- (ε-)PIR mechanism. Protect privacy of queries.
- Transparency mechanism. Allow nodes to audit SI entity actions, and reliably detect equivocation attempts.

We discuss these possible additions in Section 6.

We also note that the key-value stores may be implemented in a totally decentralized, federated or peer-to-peer manner. Since the key corresponding to each value is difficult to forge (due to the properties of hash functions), two key-value stores may be merged without expecting any conflicts – entries will either have disjoint keys, or keys for which the values are identical. Thus a distributed hash table may be used to store the key-value pairs. Alternatively, user key-value pairs may be stored with their provider. Finally, a providerless solution is also possible, by which each user maintains a local key-value store, that is updated through gossip with other users (subject to a privacy policy), or by receiving out-of-band evidence about other user's keys and values. The ability to authenticate all data in the store, gives flexibility, since evidence does not have to be from an authoritative source to be included in the key-value store.

5.3 Claimchain state tracking mechanism

Besides the storage infrastructure, ClaimChain assumes the existence of a state tracking mechanism, STM, that, given ID_N returns the current imprint of node N's claimchain, $\left(n, H(B_n^{(N)})\right)$, together with the identifier ID_SI of the SI entity where the claimchain can be accessed.

STM supports two operations:

GET.IMPRINT($\mathbf{ID_N}$) Returns an imprint $\left(n, H(B_n^{(N)})\right)$ and ID_SI for node N with identity ID_N.

SET.IMPRINT($\mathbf{ID_N}$, $\left(k, H(B_k^{(N)})\right)$, $\mathbf{ID_{SI}}$) Write or update the latest imprint and SI identifier for node N with identity ID_N.

For use in messaging, the STM can be implemented in-band during node to node communication if senders attach their own imprints and SI identifiers to message meta-data. In general, STM can exist as a special service resolving user identities to SI identifiers and claimchain imprints, as hybrid STM and SI service resolving identities and object hashes to objects directly, or as simple out-of-band sharing of the STM data.

In a federated setting we expect, besides communication peers, for users' providers to be maintaining such services on behalf of their own users. Alternatively such a service could also be decentralized by requiring a quorum of STMs to return a consistent latest imprint, before others accept it as valid.

5.3.1 Local address book

For efficiency reasons, we assume that each node maintains their own local *address book*, containing the latest imprints obtained from STM. The address book, denoted as A, is a mapping of other nodes' identifiers to the latest seen imprints of their claimchains, SI identifiers, and some meta-data M:

$$\text{ID}_N \mapsto \left((n, H(B_n^{(N)})), \text{ID}_{SI}, M_N\right) .$$

Unlike claimchains, address books are not meant to be published. We also note that peers may locally mirror, replicate or update key-value stores. Due to the specific relation between keys and values, this replication can take place without a need for consensus, since for each key only one fixed value is admissible (or at least can be found).

5.4 Operation

The basic framework allows nodes to verify and validate the claimchains of other nodes, retrieve and prove existence of given claims in claimchains, and publish own claimchains.

5.4.1 Claim inclusion test

We now describe the process to check if a given claim $C = (K, D)$ is non-revoked in the claimchain. Given a claimchain with hash chain $\mathbf{B}$, inclusion tree $\mathbf{T}$, and reverse claim index I, the process is as follows:

1. Obtain the reference $(i, j) = I(C)$. If C is not in the index, the result is N/A
2. Obtain the reference $(i', j') = I\left((K = \text{“}revocation'', D = H(C))\right)$. If (i', j') is in index, check if $H((i', j') \,||\, \text{“}revocation\text{”}$ is in $\mathbf{T}$. If yes, the result is revoked. Otherwise, proceed to the next step.
3. Check if $H((i, j) \,||\, K \,||\, D)$ is in $\mathbf{T}$. If yes, the result is non-revoked.

Alternatively, if I is not trusted (or non-existing), one may need to traverse the whole log to find the index of claim C.

The inclusion result can only be trusted to be authentic if the claimchain is authentic. The next section describes a protocol to verify this fact.

5.4.2 Claimchain verification

The claimchain verification allows to detect corruption of the claimchain, as well as impersonation attempts. Nodes use this process to decide if they want to believe in authenticity of another node N_* claimchain with imprint $\left(b, H(B_b^{N_*})\right)$. The possible outcomes of the verification process are:

Invalid The claimchain in question is corrupted, and possibly, has been tampered with.

Stale The claimchain in question has advanced past b by the time the node has started the verification. The node retrieves the imprint from STM, and attempts verification again.

Inconclusive There is not enough information for the node to come up with a decision.

Not trustworthy The node chooses not to believe that authenticity of the claimchain.

Trustworthy The node chooses to believe the authenticity of the claimchain.

Given a claimchain imprint $\left(b, H(B_b^{N_*})\right)$ and ID_{SI}^b, e.g., obtained via STM, and, if existent, the previously verified imprint $(a, H(B_a^{N_*}))$, and ID_{SI}^a, e.g., retrieved from the local address book, the verification process consists of three steps:

Step 1 – Validation First, perform three preliminary checks: i) assert that B_a is in the address book, otherwise move to step 2, ii) Then, assert $\text{ID}_{SI}^a = \text{ID}_{SI}^b$, otherwise the verification decision is not defined in the basic framework, and iii) assert that $B_a = \text{GET}(B_b^{N_*}), a)$, otherwise output invalid claimchain.

If the preliminary checks succeed, the node validates claimchain on path $\pi = (B_a^{N_*}, B_{a+1}^{N_*}, ..., B_b^{N_*})$, using the two-pass algorithm. If the validation does not succeed, the claimchain is corrupted, thus output invalid. If the validation is successful, proceed to the next step.

Step 2 – Evidence collection In this step, *evidence* about the authenticity of the claimchain being validated is collected from other nodes $N_1, \ldots, N_m$. The set of other nodes from which collect evidence is defined by each node's *evidence collection policy*, which is a function:

$$B_b^{N_*} \mapsto \{B_{n_1}^{N_1}, B_{n_2}^{N_2}, \ldots, B_{n_m}^{N_m}\},$$

where each of the evidence blocks $B_{n_i}^{N_i}$ is the latest authentic known state for N_i's claimchain. The evidence collection policy may recursively run the verification process for any of the component $B_{n_i}^{N_i}$ prior to returning the final evidence set. We note that the evidence collection policy could be NOOP.

Step 3 – Evidence processing For each evidence claimchain i, a node tests the inclusion of a cross-reference claim $\big($K = "cross-reference", D = $(b, H(B_b^{N_*}))$ $\big)$ in the claimchain. This produces a result $d_i \in \{$N/A, revoked, non-revoked $\}$.

The vector of results $\mathbf{d} = (d_1, d_2, \ldots, d_m)$ is then passed to an *evidence processing policy*, which is a function

$$\big(\mathbf{E}, \mathbf{d}\big) \mapsto d,$$

where $E = (B_{n_1}^{N_1}, B_{n_2}^{N_2}, \ldots, B_{n_m}^{N_m})$ is the evidence, and the end outcome $d \in \{$stale, inconclusive, not trustworthy, trustworthy$\}$.

5.4.3 Committing to the claimchain

The nodes commit to any statements in their claimchains when they deem needed, with some exceptions. Nodes choose the SI entities ID$_{SI_1}$, ID$_{SI_2}$, ..., ID$_{SI_m}$ they want to publish their claimchains with. To do this, nodes use claim addition and revocation operations that are specific to each ID$_{SI_i}$. Upon committing, nodes share their new block imprints and ID$_{SI_i}$ through state tracking mechanism.

Committing a cross-reference statement is a special case. It is defined by claimchain verification process outcome, that is, a node commits a cross-reference statement containing an imprint if and only if the verification outcome for the imprint is "trustworthy".

6 Enhancing ClaimChain security and privacy properties

In the previous sections we have introduced a basic version of ClaimChain that can be used to implement a high availability decentralized or federated Identity System. In this section we describe how different instantiations of the building blocks and processes provide different security properties.

6.1 Content Privacy

Our design of ClaimChain is agnostic to which authentication or authorization mechanisms are used to enable updates of the information stored in the SI or the STM. In fact, ClaimChain could operate without such control mechanisms, as the high integrity mechanisms, together with the cross references, enable to detect tampering of the evidence store content.

However, even though updating does not require access control, claimchains encode sensitive information. This information can be obtained directly by reading the data stored in the claimchain,e.g., personal keys, other nodes keys, correspondences to other identities of a node, or inferred from these data, e.g., a node's social graph can be deduced from the keys stored in the keychain. Thus, nodes may wish to protect the content of the blocks from the SI or from other nodes.

We outline below different trust models for ClaimChain and possible implementations to enforce access control to a chain block's contents.

Trusted Storage Infrastructure. The least privacy preserving option is to not protect the content from the Storage infrastructure and, at the same time, fully trust this entity to perform access control (using any classical mechanism) to the blocks in the claimchain, i.e., with ensuring that only authorized nodes can retrieve blocks from a given chain. While this protects information from other nodes, it requires revealing the list of authorized nodes to the SI, effectively revealing nodes social graphs to this entity.

Semi-trusted Storage Infrastructure. A second option is to trust the Storage Infrastructure to see the content of the blocks, but still trust it to enforce access control for other nodes. Protecting the content from the SI can be done in two ways: i) requiring the use of encryption, and thus the distribution of the decryption key to the set of nodes authorized to read the blocks; or ii) by using several (non-colluding) SIs to store "shares" of the blocks, and letting other nodes know which providers store shares. Both implementations still require the SI to know the list of authorized nodes that can access information in a given claimchain, i.e., it is necessary to reveal the social graph to the SI. In the first case this is needed to implement revocation of access permission, while in the second case not only enables revocation purposes but also ensures that only authorized nodes can obtain enough shares to recover a block.

Fully untrusted Storage Infrastructure. If nodes do not wish to trust CLAIMCHAIN's SI with their social network, they can implement encryption-based access control policy in which blocks are encrypted with different keys depending on who should have access to them. A key distribution mechanism is needed to make the corresponding decryption keys arrive to the authorized nodes. While this is the best scheme from a privacy perspective, developing and deploying a key management mechanism that efficiently distributes the keys its a challenge in itself.

Federated Storage Infrastructures. In a federated setting where users are associated with providers, they may chose to use their providers are key-value stores. In those cases the provider may provide appropriate access control to the store to enforce a policy preference of its users. This is distinct from the Trusted Storage Infrastructure, since each provider only has a partial view, namely, the statements associated with their own users. Additionally, users may chose their provider on the basis of the assurance they have in relation to the privacy of their statements.

End-User Storage Infrastructures. Finally, a completely peer-to-peer approach may be followed to implement storage infrastructures providing total end-user control over the privacy of their claims. Users may opt for only using a local key-value store to maintain their own, or other statements. Furthermore, they may chose according to an arbitrary policy of their choice to share specific claims with others. Special care needs to be taken for their hash chains themselves to leak no information, such as by including commitments to their statements, and revealing their contents selectively to communication partners or other contacts.

6.2 Contacts Privacy

In order to prevent the SI from inferring a node social graph it is not sufficient to keep the content of its claimchain secret. Furthermore, it must be possible to keep nodes' evidence collection patterns hidden from the SI. Otherwise, the SI could infer relationships among nodes by analysing which claimchains are accessed. We now outline several methods to ensure such privacy.

A possible solution to keep contacts private is to have nodes accessing the SI use an anonymous communication channel [DMS04, KLDF16]. However, given that relationships among users are persistent, and that timing patterns can be used to link accesses to the SI from a particular node, such defense is prone to intersection attacks [DS04, Ray00] that could result in the SI being able to find nodes' set of contacts.

Alternatively, nodes could not be anonymous but collect evidence from their contacts along with requests for evidence from other dummy contacts so as hide their actual relationships [KYS05, RF10]. This approach is very appealing, as it can be deployed at the client side without requiring any collaboration from the provider or a third party. However, it presents two problems. First, constructing dummy requests that are indistinguishable from real queries is extremely difficult [BTD12, CG09], and more often than not these dummy requests can be filtered out and do not provide any protection. Second, even if one can construct dummies that are indistinguishable from real accesses, it has been demonstrated that dummy access patterns have to be carefully designed to not leak information. For instance, Toledo et al. [TDG16] show that naive dummy policies, such as the random accesses proposed in Certificate Transparency [LAK13] are not secure.

A secure option is the use of Private Information Retrieval (PIR) [CGKS95, Gol07], that enables the retrieval of records from a database without revealing which actual record is being retrieved. Pure PIR schemes require a number of operations linear in the number of records in a database and thus incur in prohibitive cost at database owners when the database grows. In order to keep PIR-based SI efficient, ε-PIR variants can be used [TDG16]. These variants, composed with an anonymity system can achieve arbitrary good levels of privacy, which can be selected depending on the particular scenario in which CLAIMCHAIN is deployed.

We note that these methods *must* be combined with the access control mechanisms described in the previous section to guarantee content privacy. So that the access control does not reduce the privacy provided by the private collection of evidence we propose in the next section a privacy-preserving authentication and authorization mechanism.

6.3 Privacy of authentication

A subset of the Storage Infrastructure designs employ access control mechanisms. For example, in the Trusted Storage Infrastructure scenario mentioned above (Section 6.1), the SI maintains a list of nodes that need to authenticate in order to retrieve blocks from a specific chain. This opens a window for curious SI entities to passively collect metadata about user actions with implications to their privacy, e.g. leaking information about when an encrypted communication channel between two individuals is established. To defend against that, we combine the Contacts Privacy techniques with UnlimitID [IHD16], a Federated Identity Management system that makes it difficult to link an authorized action to the real identity of the user.

UnlimitID, conceived in the context of NEXLEAP, is a privacy enhancement to the dominant authorization protocol in the Web, the OAuth [Har12]. The main goal of UnlimitID is to protect from curious Identity Providers that passively keep track of of which services their users are using and how. Effectively, it allows users of an Identity Provider to anonymously authorize to third-party services and to create pseudonymous accounts with them, without revealing their real identity. These pseudonymous accounts can encode the user attributes, yet are unlinkable to each other and to the account at the Identity Provider. This is achieved via the use of anonymous attribute-based credentials based on algebraic MACs [CMZ14]. The anonymity guarantees provided by UnlimitID are proportional to the number of active users on the Identity Provider that share the same values in their credentials, and so should increase for large Identity Providers.

We advice privacy-friendly Storage Infrastructure entities to avoid using traditional authorization mechanisms, such as username and password authentication, that bind an action to the identity of a user. They should rely on anonymous credentials instead. More specifically, constructs such as the anonymous credentials based on aMACs [CMZ14] are very efficient and versatile; they allow for selective attribute disclosure and can be locally blinded and used for an arbitrary number of times without leaking any information. We also suggest that the anonymous credentials encode an expiry date that is fixed among users, or that they can be blacklisted [TAKS07].

6.4 Storage non-equivocation

Finally, we would like to mention that CLAIMCHAIN could be combined with a transparency log in which the SI (or the STM) record evidence of operations made by users. Like in CONIKS [MBB^{+}15], this provider-oriented log provides a means for nodes to verify that providers return up to date information and do not try to equivocate nodes by denying having some information.

7 ClaimChain Instantiations

In this section we discuss how ClaimChain can implement traditional Identity Systems, as those presented in Section 3.1. We also discuss how our scheme can be provide support to existing secure messaging solutions.

7.1 Implementing identity management paradigms with ClaimChain

In order to decide whether a claim should be believed, we have stated that evidence provided by other nodes has to be collected, and validated according to a so-called verification policy. By changing which are the nodes that provide such evidence, and what are the conditions necessary to consider such evidence as sufficient proof that a claim is truthful, we can instantiate well-known identity management paradigms in ClaimChain.

Certification Authority-based PKI. In this Identity Management approach there exists one (or several) Certification Authority (CA) that has the ability to certify any node's state, traditionally by digitally signing nodes' public keys, in such a way that when any other node in the system sees the signature it believes the validity of the statement.

ClaimChain can easily implement this scenario by considering that there is one (or several) "super nodes", equivalent to the CA whose claimchain contains evidence about the state of any other node in the system. When a node ID$_A$ wants to verify a claim made by a node ID$_B$ it only needs to validate ID$_B$'s state in the CA claimchain. Once the state is verified, ID$_A$ can proceed to verify if the claim made by ID$_B$ is in ID$_B$'s claimchain and it is not revoked.

Federated identity. Instead of a central Certification Authority, in a Federated identity management scenario different groups of nodes are associated to different STM entities, which we will call here Identity Providers (IPs). Nodes only trust statements issued by their own IP, and providers only validate claims about their own nodes. Furthermore, IPs establish trust relationships among themselves, in such a way that users from an IP can validate claims about users from other IPs via the transitive property.

In ClaimChain, such scheme can be instantiated by considering that IPs have claimchains that attest the state of their users' claimchains, i.e., include statements about their users' claimchain heads; and also reflect their trust relationship with other IPs by including the state of their claimchains. Since in ClaimChain there is no interaction between nodes, it is natural to choose the iterative alternative so that there is no need for providers to carry out any operation.

To implement this paradigm, together with the latest state of its claimchain, ID$_B$ must also provide the latest state of her provider ID$_{IP_B}$'s claimchain. Then, ID$_A$ would first check that her provider ID$_{IP_A}$ trusts ID$_B$'s provider by checking that ID$_{IP_B}$'s head is in ID$_{IP_A}$'s claimchain. If this is the case, then ID$_A$ can validate the state of ID$_B$ using IDIP_B's chain.

Nyms Identity Directory and LEAP Nicknym. The above scenario can be extended to support the Nyms Identity Directory design, with Identity Providers acting as Trusted Notaries (TNs) that upon checking the authenticity of a binding (e.g. via an email verification protocol) will also attest the state of the claimchains of users from other providers as if they belonged to them.

The function of Network Perspective auditing is possible via the iterative evidence collection strategy (see Figure 3(b)). If a node ID$_A$ wants to check that the Identity Provider of node ID$_B$ has served the right state of ID$_B$'s chain, ID$_A$ will request from her provider to also fetch the head of the ID$_B$'s provider claimchain and make sure that is the same with the one she had received. As for the the Nicknym Federated Web of Trust, it can be implemented with providers also including in their claimchain the latest state of the claimchains of other providers they trust.

ClaimChain	PGP
Head imprints	Public key fingerprints
SI	Keyservers
STM	Keyservers
Initial verification key	Certification key
Recovery key	Certification key certified by the initial certification key
Cross-reference	Certification of another person's key
Self-reference	Certification of one's own key
Generic claim	Signed message

Figure 5: Correspondence between ClaimChain and PGP notions

Web of Trust. PKI infrastructures can be implemented in a decentralized manner, in which claims are not certified by a recognized Authority (e.g., CAs or IPs) but nodes rely on other nodes to attest the validity of those claims.

ClaimChain naturally implements this paradigm. Here, nodes that attest for other nodes' claimchains states by including their heads in their own claimchains Thus, when a node ID$_A$ wants to verify a claim made by a node ID$_B$ it finds out which other nodes attest for ID$_B$'s state, for instance ID$_C$ and ID$_D$ (note that ID$_C$ and ID$_D$ have been chosen as evidence providers as a matter of example, and in practice they would depend on ID$_A$'s social policy which is orthogonal to the operation of ClaimChain). Then ID$_A$ gathers evidence about the state from ID$_C$'s and ID$_D$'s claimchains. If the evidence processing indicates that the state of ID$_B$ is correct, ID$_A$ can check whether the claim is in ID$_B$'s claimchain, and if this is the case be convinced that the claim is valid.

7.2 Relation to existing mechanisms for secure email

7.2.1 Pretty Good Privacy (PGP) identity management

There is certain correspondence of ClaimChain notions to those in PGP and PGP Web of Trust explained in Section 2, which are summarized in Table 5.

ClaimChain's Head imprints can be seen as equivalent to PGP's public key fingerprints. An imprint can be used to verify a claim regarding the binding of a key and an identity, and the same applies to fingerprints and public keys. Claimchains, however, can be used to verify other types of claims, including key updates and revocations, and in general any generic statements. This functionality does not have an exact counterpart in PGP.

The roles of both SI and STM are performed in PGP by keyservers. A difference between both approaches is that in ClaimChain, SI and STM can be implemented in many ways, see Section 5.2, avoiding the need to trust a central storage such as the keyserver.

Another important difference to PGP is that ClaimChain allows for completely automatic handling of updates. The claimchain verification does not require any input from owners other than setting evidence processing policy. The updates including cross-references of other claimchains can happen automatically as well. This can result in better usability, since users can be relieved from manually certify other keys. Also, ClaimChain cryptographic-based properties enables automation of the checks in email or messaging clients.

7.2.2 In-band key discovery: Autocrypt

New initiatives try to avoid the usability pitfalls of both social-based and key-server based PGP by embedding key discovery within messages exchange. An example is Autocrypt[10] that has recently been rebranded *autoencrypt*, a protocol whose goal is to provide a key discovery mechanism that does not require an infrastructure to find the binding between identities and keys. Autocrypt proposes to distribute the key material by embedding it directly into the headers of email messages.

The Autocrypt key discovery mechanism is only secure against passive eavesdroppers. Email providers or any intermediate entity with access to the message could perform active attacks to either modify keys or downgrade the communication channel to not use encryption. A possible solution to this is to let users detect such tampering using out-of-band verification. ClaimChain can be used as a complement for Autocrypt as a mechanism to verify the integrity of keys, and, more importantly, the integrity of the binding to an identity. This can be done by using the ClaimChain validation process for the keys discovered through Autocrypt.

We also note that the key discovery mechanism in Autocrypt can be leveraged to include claimchain imprints along with keys, thus implementing a distributed in-band STM. This would allow Autocrypt users to automatically verify the transferred keys, not requiring the out-of-band fingerprint verification.

Instead of relying on a separate infrastructure to support Autocrypt, ClaimChain may also be embedded 'in-band' as part of the Autocrypt architecture. Autocrypt aims for backwards compatibility with traditional email, where Mail User Agents (MUAs) store email, and transfer it to others using the Simple Mail Transfer Protocol (SMTP), and retrieve it from providers using IMAP or the simpler POP3 protocols.

The embedding may work as follows: each MUA implements a local and private storage provider component, mapping keys to values, as required by ClaimChain. Data relevant to the local MUA, such as it own chain or chains of contacts are thus kept locally. When a user sends an email, the imprint of their claimchain may be included, as well as other key-value pairs as part of either special SMTP headers or attachment to the normal email. Such headers and attachments may also be encrypted to protect contact privacy. Upon receiving such an email, a recipient MUA includes the key-value pairs sent into its store, and if needed attempts to authenticate the imprint received according to the ClaimChain semantic rules. As a result, there is no need for a centralized storage provider, and MUAs collectively implement a decentralized and partially replicated information store for their own users. Note that this decentralization also alleviates the privacy concerns with respect to a centralized store both regarding content and access patterns.

The embedding of ClaimChain into Autocrypt maintains safety: our design assumes that the key-value store is untrusted, therefore including any well formed claims into the store cannot violate our integrity properties. Since store keys are a hash of their values, they are self-certifying, and an adversary cannot easily find a different value for a specific key. Thus merging incoming key-value pairs into a local store is safe, and there is no need for locking or consensus to ensure the eventually consistence of stores. In that respect the local stores implement a conflict-free replicated data structure that is safe. Potential bogus key-value pairs cannot influence the verification process of ClaimChain since it relies on digital signatures and hash-chains to authenticate valid data.

From the point of view of privacy, this embedding provides great flexibility to users. Their MUA may choose to share, or not, particular claims with selected recipients. At a minimum enough information could be shared to establish the integrity of the MUA's own chain. This would allow recipients to follow and authenticate key updates from the sending MUA. However, in the context of group communications, such as email to multiple recipient or mailing lists, information about other participant's chains may also be included to detect inconsistencies and forks – denoting an man-in-the-middle attack attempt. Techniques based on cryptographic access control to specific statement can also be deployed to provide even greater flexibility.

This decentralized embedding does not preclude additional support services that enable a more centralized or federated key-value store. Email providers may choose to act as an on-line key-value store for their own users.

[10] https://autocrypt.readthedocs.io/en/latest/

This does not expose them to the certification liability (a problem in CONIKS), since the store they serve is self-certifying and authenticated as to its integrity in an end-to-end manner. Other services may include mailing list listservs robots that broadcast known key-values or totally independent services. The conflict-free property of the storage provider allows all those sources to be combined to attempt to establish the veracity of any claims.

References

[AL02] Carlisle Adams and Steve Lloyd. *Understanding PKI: Concepts, Standards, and Deployment Considerations*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2002.

[BCG+14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 459–474, 2014.

[BGB04] Nikita Borisov, Ian Goldberg, and Eric A. Brewer. Off-the-record communication, or, why not to use PGP. In *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society, WPES 2004, Washington, DC, USA, October 28, 2004*, pages 77–84, 2004.

[BTD12] Ero Balsa, Carmela Troncoso, and Claudia Díaz. OB-PWS: obfuscation-based private web search. In *IEEE Symposium on Security and Privacy*, pages 491–505. IEEE Computer Society, 2012.

[CDF+07] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer. OpenPGP Message Format. RFC 4880, November 2007.

[CG09] Richard Chow and Philippe Golle. Faking contextual data for fun, profit, and privacy. In Ehab Al-Shaer and Stefano Paraboschi, editors, *ACM Workshop on Privacy in the Electronic Society, WPES 2009*, pages 105–108. ACM, 2009.

[CGKS95] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *36th Annual Symposium on Foundations of Computer Science*, pages 41–50. IEEE Computer Society, 1995.

[CMZ14] Melissa Chase, Sarah Meiklejohn, and Greg Zaverucha. Algebraic macs and keyed-verification anonymous credentials. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 1205–1216. ACM, 2014.

[CSF+08] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280, May 2008.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In Matt Blaze, editor, *Proceedings of the 13th USENIX Security Symposium, August 9-13, 2004, San Diego, CA, USA*, pages 303–320. USENIX, 2004.

[DS04] George Danezis and Andrei Serjantov. Statistical disclosure or intersection attacks on anonymity systems. In Jessica J. Fridrich, editor, *6th International Workshop on Information Hiding*, volume 3200 of *Lecture Notes in Computer Science*, pages 293–308. Springer, 2004.

[GM16] Matthew D. Green and Ian Miers. Bolt: Anonymous payment channels for decentralized currencies. *IACR Cryptology ePrint Archive*, 2016:701, 2016.

[Gol07] Ian Goldberg. Improving the robustness of private information retrieval. In *IEEE Symposium on Security and Privacy (S&P 2007)*, pages 131–148. IEEE Computer Society, 2007.

[Har12] Dick Hardt. The oauth 2.0 authorization framework. 2012.

[IHD16] Marios Isaakidis, Harry Halpin, and George Danezis. Unlimitid: Privacy-preserving federated identity management using algebraic macs. In *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society*, pages 139–142. ACM, 2016.

[KLDF16] Albert Kwon, David Lazar, Srinivas Devadas, and Bryan Ford. Riffle: An efficient communication system with strong anonymity. *PoPETs*, 2016(2):115–134, 2016.

[KW16] Holger Krekel and Max Wiehle. Identifying communities, use cases, development flows, 2016.

[KYS05] Hidetoshi Kido, Yutaka Yanagisawa, and Tetsuji Satoh. An anonymous communication technique using dummies for location-based services. In *Proceedings of the International Conference on Pervasive Services 2005, ICPS '05*, pages 88–97. IEEE Computer Society, 2005.

[LAK13] B. Laurie, Langley A., and E. Kasper. Certificate transparency. RFC 6962, June 2013.

[MBB$^+$15] Marcela S. Melara, Aaron Blankstein, Joseph Bonneau, Edward W. Felten, and Michael J. Freedman. CONIKS: bringing key transparency to end users. In Jaeyeon Jung and Thorsten Holz, editors, *24th USENIX Security Symposium, USENIX Security 15*, pages 383–398. USENIX Association, 2015.

[ME16] Francesca Musiani and Ksenia Ershomina. Initial decentralization case-studies, 2016.

[Mer89] Ralph C Merkle. A certified digital signature. In *Conference on the Theory and Application of Cryptology*, pages 218–238. Springer, 1989.

[MHKS14] Andrew Miller, Michael Hicks, Jonathan Katz, and Elaine Shi. Authenticated data structures, generically. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 411–424, 2014.

[MP16] Moxie Marlinspike and Trevor Perrin. The X3DH key agreement protocol. Revision 1, Open Whisper Systems, November 2016.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[Pug90] William Pugh. Skip lists: a probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6):668–676, 1990.

[Ray00] Jean-François Raymond. Traffic analysis: Protocols, attacks, design issues, and open problems. In Hannes Federrath, editor, *Designing Privacy Enhancing Technologies, International Workshop on Design Issues in Anonymity and Unobservability*, volume 2009 of *Lecture Notes in Computer Science*, pages 10–29. Springer, 2000.

[RF10] David Rebollo-Monedero and Jordi Forné. Optimized query forgery for private information retrieval. *IEEE Trans. Information Theory*, 56(9):4631–4642, 2010.

[SHKP16] Elijah Sparrow, Harry Halpin, Kali Kaneko, and Ruben Pollan. LEAP: A next-generation client VPN and encrypted email provider. In Sara Foresti and Giuseppe Persiano, editors, *Cryptology and Network Security - 15th International Conference, CANS 2016, Milan, Italy, November 14-16, 2016, Proceedings*, pages 176–191, 2016.

[SMA$^+$13] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. RFC 6960, June 2013.

[TAKS07] Patrick P. Tsang, Man Ho Au, Apu Kapadia, and Sean W. Smith. Blacklistable anonymous credentials: blocking misbehaving users without ttps. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007*, pages 72–81, 2007.

[TDG16] Raphael R. Toledo, George Danezis, and Ian Goldberg. Lower-cost $\in$-private information retrieval. *PoPETs*, 2016(4):184–201, 2016.

[TDIH16] Carmela Troncoso, George Danezis, Marios Isaakidis, and Harry Halpin. Internet science vocabulary and discussions, 2016.

[Zim95] Philip R Zimmermann. *The official PGP user's guide*. MIT press, 1995.