

# reclaimID: Secure, Self-Sovereign Identities using Name Systems and Attribute-Based Encryption

Martin Schanzenbach, Georg Bramm, Julian Schütte

*Fraunhofer AISEC*

Garching near Munich, Germany

{schanzen,bramm,schuette}@aisec.fraunhofer.de

**Abstract**—In this paper we present reclaimID: An architecture that allows users to reclaim their digital identities by securely sharing identity attributes without the need for a centralised service provider. We propose a design where user attributes are stored in and shared over a name system under user-owned namespaces. Attributes are encrypted using attribute-based encryption (ABE), allowing the user to selectively authorize and revoke access of requesting parties to subsets of his attributes. We present an implementation based on the decentralised GNU Name System (GNS) in combination with ciphertext-policy ABE using type-1 pairings. To show the practicality of our implementation, we carried out experimental evaluations of selected implementation aspects including attribute resolution performance. Finally, we show that our design can be used as a standard OpenID Connect Identity Provider allowing our implementation to be integrated into standard-compliant services.

**Index Terms**—identity and access management, peer-to-peer, privacy, decentralisation, name systems, attribute-based encryption

## I. INTRODUCTION

Today, users are often required to share personal data, like email addresses, to use services on the web. As part of normal service operation, such as notifications or billing, services require access to – ideally fresh and correct – user data. Consider the use case of a user subscribing to a social networking service. After successful registration and providing the service provider with an email address, the service uses it to send notifications such as status updates from friends. At the time of notification delivery, the service needs access to the respective email addresses. However, services cannot interact with users that are offline.

To mitigate this issue, services store user data in a database upon registration or retrieve it from a third party Identity Provider (IdP). If the data is stored, it can become stale unless diligent users continuously update their data. If the data is retrieved from an IdP, both user and service must be able to rely on the IdP to provide fresh, authentic attribute data. Further, the IdP must be available whenever needed and ideally does not abuse usage patterns, for example for user profiling.

Sharing user attributes in the Web today is often done via IdPs to reduce data redundancy and to give services access to current, up-to-date information even if the user is currently offline. The most common approach is to use one of the two major IdPs: Google or Facebook. Together they

claim over 85% of the identity provider market<sup>1</sup>. The use of central service providers allows users to efficiently manage identity information and control access. The IdP service is responsible for enforcing access control decisions made by the user regarding identities and attributes. Consequently, the IdP has full access and control over the managed user data. Abuse of this power is theoretically limited by local laws and regulations [1]. But, they are often ignored or challenged [2] and centralised service providers are major targets for targeted advertisement businesses as well as hackers, including government actors [6]–[8].

From a security perspective, this setup of omniscient intermediaries is a significant threat to the users' privacy. Users must completely trust the IdP with respect to protecting the integrity and confidentiality of their identity in their interest. Various breaches of large IdPs such as the ones at Yahoo that revealed 3 billion user records to the public<sup>2</sup> have shown that these expectations are hard to meet at times. Finally, IdPs such as Facebook – and for a long time also Google – enforce a “real-name policy”<sup>3</sup>. Denying pseudonymous identity can be considered to be in direct violation to the human right to be forgotten.

In this paper, we present the design and a reference implementation of reclaimID. We address the issues elaborated above by not relying on a centralised IdP to serve attributes. In reclaimID, users manage their attributes in a name system and can selectively grant other parties access. The name system ensures that attributes can be accessed asynchronously whenever needed and provide integrity as well as authenticity guarantees. Access to attributes is authorized and enforced through the use of attribute-based encryption (ABE). We implemented this design using ciphertext-policy ABE and the GNU Name System [3], [4] to show that it can be practically realised. Further, we show how reclaimID can be integrated into a standardised authorization and authentication protocol in the form of OpenID Connect.

<sup>1</sup><http://www.gigya.com/blog/the-landscape-of-customer-identity-q2-2015/>, accessed 2017/02/20

<sup>2</sup>[https://en.wikipedia.org/w/index.php?title=Yahoo!\\_data\\_breaches&oldid=817379693](https://en.wikipedia.org/w/index.php?title=Yahoo!_data_breaches&oldid=817379693), accessed 2018/01/09

<sup>3</sup><http://www.businessinsider.de/facebook-changes-to-real-name-policy-2015-12>, accessed 2018/02/06

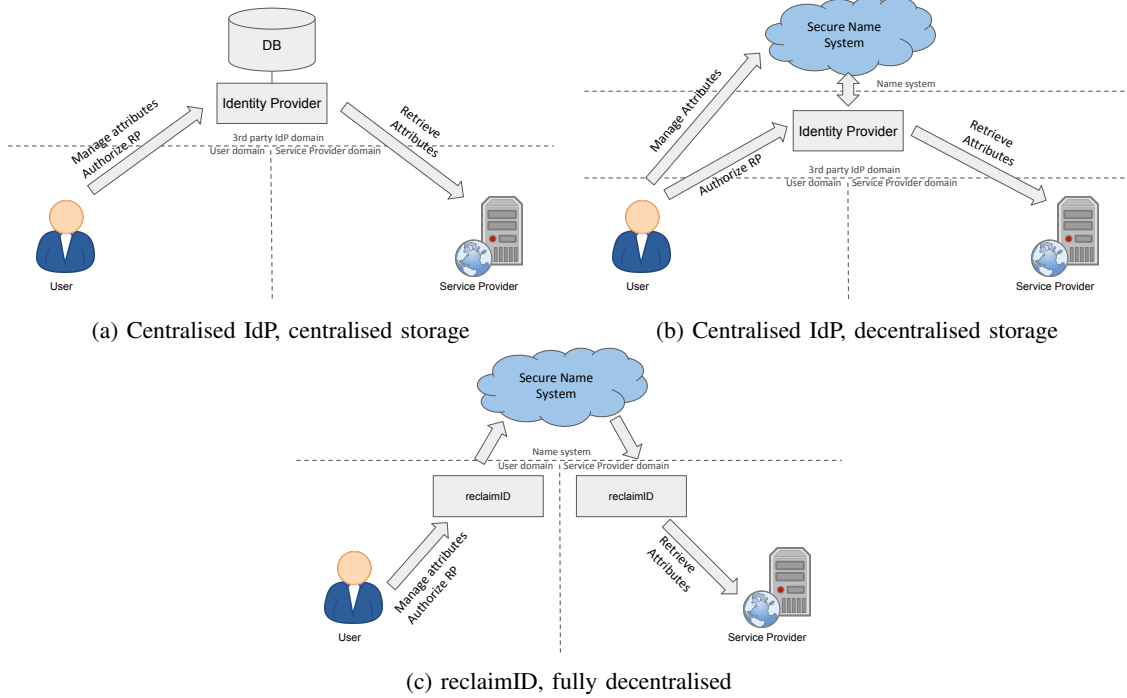


Fig. 1: Identity Provider Architectures

## II. APPROACH

Traditionally, user attributes are centrally managed at an IdP and shared using protocols such as OpenID Connect 1.0 (OIDC) or SAML 2.0. Requesting parties<sup>4</sup> (RPs), as well as users, must trust the IdP with respect to availability, integrity, and confidentiality of attributes. Figure 1a illustrates this setup.

An alternative to maintaining central user databases are decentralised approaches such as NameID [5]. In NameID, there is a central IdP service that reads identity information from the *Namecoin* name system. Name systems consist of disjoint namespaces that contain mappings from human-readable names to values. NameID equates namespaces with digital identities. Figure 1b illustrates the subtle difference between a identity management system with decentralised storage and traditional IdPs. However, as even in this setup requests are still relayed over a single IdP, both architectures rely on a central IdP service that consequently acts as a single point of failure and is omniscient to all interactions between entities. NameID additionally does not protect the confidentiality of information that is stored in Namecoin in any way. This is rendering the access control that a user may perform in the process of an authorization pointless.

The integration of identity management with recent name systems such as Namecoin or GNS has several advantages such as an out-of-the-box, close coupling of namespaces with cryptographic identities (i.e. public key pairs). Further, name systems are not much different from identity management systems and most of their protocols can be used as a basis.

Specifically, we observe the following equivalences and differences between name systems and identity management (IdM) systems.

- 1) Namespaces in name systems are equivalent to digital identities. Namespaces can be defined by or cryptographically associated with a public and private key pair.
- 2) In a namespace, resource records are self-issued attributes of a user. These are equivalent to attribute names and values in an IdM system.
- 3) Retrieving the attribute of an identity is equivalent to a name query in the respective namespace.
- 4) Records in the name system can be queried by anyone. In an identity management system however, the confidentiality of identity attributes must be protected according to a user policy, i.e. when using a name system as the underlying layer of an IdM, such protection must be added on top.

With our approach, we set out to address the shortcomings discussed above. By designing a decentralised identity management system that is user-managed and protects the users privacy we mitigate the requirement of a central IdP service that routes all requests. We propose reclaimID, a decentralised IdP service that provides all typically required functions such as creation, querying, updates, and revocation of user identities in a decentralised way. Similar to NameID, we use name systems as an underlying transport and storage medium. However, reclaimID does not rely on any centralised component, as illustrated by Figure 1c. The concept of reclaimID can be implemented on top of any name system, inheriting most of its security properties. Additionally, reclaimID includes an

<sup>4</sup>In OpenID Connect referred to as “relying party”

additional authorization layer using ABE on top of the name system to ensure confidentiality and to enable policy-based access control of user attributes.

We design reclaimID to satisfy all requirements existing approaches also satisfy: From a user perspective it must be possible to manage one or more identities including respective attributes. Further, the user must be able to selectively manage third party access to subsets of those attributes. From a requesting party perspective it must be possible to request access to attributes of a user. Those attributes must then be retrievable without a direct communication channel to the user.

#### A. Security Goals and Threat Model

Our goal is to ensure availability, authenticity, integrity, privacy and confidentiality of user attributes. The user must be able to control access to his attributes by selectively authorising requesting parties. As nation-wide manipulation of the domain name system, data leaks and the surveillance of global IdPs are a matter of fact today [6]–[8], our attacker model must consider an attacker with the ability to collect any data in transit between all participants, as well as manipulation of a limited but large number of nodes in the network. We also assume that the attacker is able to coerce participants into submission of data they own or have access to. However, we assume that the attacker is not able to break cryptographic primitives that are considered secure by the research community today. In the following we discuss the security properties reclaimID aims to satisfy in detail:

**Availability** By not relying on a centralised service that serves user attributes and allows the user to manage authorizations, our system provides higher availability guarantees especially in the face of powerful attackers such as nation states that may utilise “lawful” interception techniques. One major drawback of decentralised services with peer-to-peer connectivity is the possibility of high user churn, which could potentially render identity attributes unavailable. This would be problematic in use cases such as the social network provider, as the user attributes must be accessible by requesting parties even if the user is not online. To solve this problem, our design is based on decentralising the service that allows users to manage attributes and selectively share them. This is achieved by storing attributes in a name system where the user acts as the main authority over the data. User attributes are stored in a distributed fashion and can be queried even if the user’s peer is not online. It should be noted that the concrete availability guarantees heavily depend on the underlying name system. It is thus important to evaluate available name systems for an implementation of reclaimID. We discuss name systems in Section III-H and our particular choice as part of the implementation in Section IV.

**Authenticity and Integrity** Users store attributes through the use of the IdP in a namespace of a name system. We assume that by doing so, attributes are inherently self-signed using the private key associated with the identity of the user that owns the namespace. Requesting parties can thus verify that the attribute was indeed stored by the user and has not

been modified by any third party by simply verifying the signature. A more common use case is that a requesting party requires attributes that were issued by a trusted<sup>5</sup> third party. In the case of Google or Facebook, the IdP is implicitly the trusted issuer of all attributes. Our proposed reclaimID, similar to NameID, does not define the form factor of attributes and the user itself is implicitly the issuer of attributes. However, the use of attribute-based credentials (ABCs) is perfectly possible in both – for example through the use of X.509 certificates as attributes.

**Privacy and Confidentiality** We argue that given centralised IdP services, an attacker can coerce the service provider and then has knowledge over all connections between users and requesting parties. The attacker can learn, for example, which services the user accesses over time. Our design does not directly mitigate this issue as some name systems do not protect the confidentiality of queries and responses as well as metadata. Name systems such as Namecoin and GNS can offer such protection due to built-in “query privacy”. We will discuss the security properties of selected name systems later.

To generally ensure confidentiality of attributes, we propose to encrypt them using ABE before they are stored in the name system. Through the ABE-layer of reclaimID, requesting parties are issued user keys that only allow decryption of those attribute subsets that they are authorized to access. We propose to bootstrap an ABE system for every identity. This means that every user acts as its own, individual key authority, giving him full and exclusive authority over all user keys and attributes. This achieves both, confidentiality of user attributes in the otherwise public namespace of the name system, and fine-grained access control of requesting parties to these attributes. Revocation of access rights to attributes is achieved by the creation of new keys, deletion of the existing attributes and publication of re-encrypted attributes over the name system. A further advantage of enforcing access rights cryptographically through ABE rather than traditionally through a central trusted IdP is that individually encrypted attributes profit from the caching implemented in most name systems and significantly reduce network overhead for the retrieval of attributes (even to zero, if the cache is local). We note that as our attacker is able to coerce participants into submission of data this approach does not protect against the case where an authorized third party is attacked. However, unlike an IdP, an authorized third party likely does not have access to *all* the data of a single user and cannot observe access patterns of other services.

Another aspect to privacy is that a user might not want to disclose the attribute value to a requesting party. Instead, only certain properties should be disclosed. A common use case is age verification, where a user only wants to disclose that she is over a certain age. Advancements in the area of privacy-preserving attribute-based credentials (PP-ABCs) [9], [10] allow this through the use of interactive zero-knowledge proofs. In our design the interactivity requirement is not

<sup>5</sup>trusted by the consumer of the attribute, e.g. the requesting party

acceptable, but recent work in the form of non-interactive zero-knowledge proofs [11] can be used in reclaimID, if needed.

### III. DESIGN

In this section, we present the main design of reclaimID's protocols. We show how we combine a name system and ABE scheme to realise a privacy-preserving, decentralised IdP. In particular, we answer the question of how users can grant and revoke authorizations for requesting parties to access their attributes. Further, we discuss the protocol for authorized parties to retrieve and decrypt user attributes from reclaimID. Finally, we elaborate on the impact that the choice of name system as well as ABE scheme has on an implementation of reclaimID.

#### A. Preliminaries

Attribute-based encryption schemes come in the two main flavours of ciphertext-policy ABE (CP-ABE) and key-policy ABE (KP-ABE). As our access policies refer only to single labels in the name system, both variants are likewise suited. For a first implementation we chose CP-ABE. A thorough discussion on the various implications of the two types of ABE schemes and the various existing schemes that exist for them can be found in Section III-H. For the discussion of the system's design, both variants can be considered equally possible. In the following, we present three major components: A name system, a CP-ABE scheme and an IdP. We explain how in reclaimID the first two components are used to realise the third.

First, we define the high-level functions and procedures and objects for all components. Let an ABE scheme consist of the following functions:

$$\begin{aligned} \text{Setup}_{\text{ABE}}() &\rightarrow (msk_{\text{ABE}}, pk_{\text{ABE}}) \\ \text{Keygen}_{\text{ABE}}(msk_{\text{ABE}}, A) &\rightarrow sk_{\text{ABE}} \\ \text{Enc}_{\text{ABE}}(pk_{\text{ABE}}, pt, policy) &\rightarrow ct \\ \text{Dec}_{\text{ABE}}(sk_{\text{ABE}}, ct) &\rightarrow pt \end{aligned} \quad (1)$$

Where  $msk_{\text{ABE}}$  is the master secret key,  $pk_{\text{ABE}}$  the public parameters key and  $sk_{\text{ABE}}$  a derived user key in the ABE scheme.  $A$  is a set of tags, or attribute names that can be associated with a key  $sk_{\text{ABE}}$  using the function  $\text{Keygen}_{\text{ABE}}()$ . Here,  $policy$  describes the policy that is attached to a ciphertext  $ct$ . Finally,  $pt$  denotes the plaintext message. For encryption and decryption we define the functions  $\text{Enc}_{\text{ABE}}()$  and  $\text{Dec}_{\text{ABE}}()$ , respectively.

For the name system and IdP, we define  $pk_{\text{user}}, sk_{\text{user}}$  as the public and private key pair associated with an identity  $ID_{\text{user}}$ . We define the functions  $\text{Enc}()$  and  $\text{Dec}()$  as the asymmetric encryption and decryption functions for use with the identity key pair.

Name systems consist of *namespaces* that are owned by users or legal entities. Namespaces are managed by their respective owners and contain name-value mappings. In our use case, the owner of a namespace is an "issuer" of attributes in its respective namespace. As such, name systems inherently

provide a storage, resolution and delegation mechanism for self-issued attributes. Such name-value mappings are realised in name systems using *resource records*. In name systems this mapping must be cryptographically bound to the namespace owner, usually through digital signatures using public-key cryptography. Let a name system  $N$  consist of the following procedures

$$\begin{aligned} \text{Resolve}(ID_{\text{user}}, name) &\rightarrow R \\ \text{Publish}(ID_{\text{user}}, name, R) & \\ \text{Depublish}(ID_{\text{user}}, name) & \end{aligned} \quad (2)$$

$R$  is a record in a namespace and  $name$  is a name in a namespace owned by  $ID_{\text{user}}$ . While it may seem odd that a resolution takes a name *and* a namespace as argument it is not. Internally resolvers in a name system always query for names inside namespaces. It is common, however, that the resolver has specialised procedures that hide this from the user, e.g. by performing iterative lookups and segmenting a long name into multiple labels. The prime example here is DNS, which allows a user to resolve a "fully qualified domain name" (FQDN) by iteratively trying to find the authoritative namespace for a specific label. We assume that in the name system  $sk_{\text{user}}$  is used to create a record signature over the data in a record  $R$  in a namespace owned by  $ID_{\text{user}}$ . When  $R$  is published using  $\text{Publish}()$ , the signature is stored alongside the record. Consequently, we also assume that the record signature is verified when a record is retrieved using  $\text{Resolve}()$ . The respective public key can be used to uniquely identify the namespace and verify record signatures. A published record is no longer resolvable after calling  $\text{Depublish}()$  and the cached records expire.

Finally, let an *IdP* consist of the procedures:

$$\begin{aligned} \text{Store}(ID_{\text{user}}, attribute) & \\ \text{Delete}(ID_{\text{user}}, attribute) & \\ \text{Authorize}(ID_{\text{user}}, ID_{\text{rp}}, attributes) &\rightarrow ticket \\ \text{Revoke}(ticket) & \\ \text{Retrieve}(ID_{\text{rp}}, ticket) &\rightarrow attributes \end{aligned} \quad (3)$$

The procedures  $\text{Store}()$  and  $\text{Delete}()$  allow the user  $ID_{\text{user}}$  to manage attributes.  $\text{Authorize}()$  is the procedures used to authorize a requesting party  $ID_{\text{rp}}$  to access a set of attributes. This access can be revoked using  $\text{Revoke}()$ . The requesting party can use the  $\text{Retrieve}()$  procedures to access attributes it was granted access to.

We define an identity *attribute* as follows:

$$attribute = (name, value, version) \quad (4)$$

The *name* is an attribute identifier, such as "email". An attribute also has a *value* associated with it. The *value* may contain arbitrary data associated with *name* such as "john@doe.com". It may also contain more complex data structures such as credentials issued by third parties. The details of attribute values, however, are out of scope in our

design. The attribute *version* is relevant for revocation in the later sections of this chapter.

The *attributes* specified in **Authorize()** and **Retrieve()** are a set of attributes. A *ticket* is a handle of an authorization that is passed to the authorized requesting party so it can access the shared attributes. We define a ticket as follows:

$$ticket = (ID_{user}, ID_{rp}, names, rnd) \quad (5)$$

The ticket identities  $ID_{user}$  and  $ID_{rp}$  identify the user that issued the ticket and the requesting party, respectively. *names* is the list of attributes that the requesting party is authorized to access and *rnd* is a random label under which the user key  $sk_{ABE}$  for the requesting party is stored encrypted in the namespace of the identity. This ticket must be transferred in an initial out-of-band authorization process and is used by the requesting party to retrieve attribute data.

In the following, we always assume that given an identity, its public key  $pk_{user}$  and the associated ABE key material can also be retrieved. If a procedure is called by an identity, we also assume that we have access to the respective private keys  $sk_{user}$ ,  $sk_{rp}$  and  $sk_{ABE}$ . Before storing the first attribute, a user must bootstrap an ABE system. In this process, the user creates an ABE public parameters key  $pk_{ABE}$  and master secret key  $msk_{ABE}$  for one of her namespaces by executing **Setup<sub>ABE</sub>()**.

### B. Storage

In reclaimID, the encrypted value of an attribute is stored inside a resource record  $R$  in the name system. By publishing the resource record under the attribute name the user effectively issued an attribute to her identity. In Algorithm 1 we define the **IdP Store()** procedure.

---

#### Algorithm 1: Store

---

**input** : User attribute  $a$   
User identity  $ID_{user}$

- 1  $policy \leftarrow \text{Concat}(a.name, a.version);$
  - 2  $ct \leftarrow \text{Enc}_{ABE}(pk_{ABE}, a.value, policy);$
  - 3 **Publish**( $ID_{user}, a.name, ct$ );
- 

First, we use the concatenation procedure **Concat()** to build the ABE *policy* from the attribute name and version. The resulting policy can be interpreted as “To decrypt the ciphertext, a key associated with a tag representing the attribute in the respective version is required”. To create the record data that is stored in the name system, we encrypt the attribute value using the ABE encryption function **Enc<sub>ABE</sub>()**. The encrypted attribute value is published as a record under the attribute name using the name system function **Publish()**. We note here that internally name systems distinguish between different types of records. We therefore define the record type of records representing identity attributes to be “ID\_ATTR”. The record type does not serve any specific function except from allowing us to distinguish our records from, e.g. IP addresses. In our design, all attribute resource records must have this type set.

We also note here that records in name systems expire. An implementation must make a choice for an appropriate expiration time that allows to efficiently make use of the respective caching mechanism in the name system, if any.

### C. Authorization

To authorize a requesting party to access a set of attributes, the user must create an authorization-specific user secret key  $sk_{ABE}$  using the ABE function **Keygen<sub>ABE</sub>()**. For  $sk_{ABE}$  to be used to decrypt the respective attribute records of the shared attributes it must be associated with a specific set of *tags*.

There are two ways an authorized party can learn  $sk_{ABE}$ : Resolving it through the name system or via an out-of-band exchange, for example using a web-based authorization protocol. The latter is only possible in “synchronous” use-cases, i.e. when user and authorized party are both online. In use-cases where user or authorized party are offline,  $sk_{ABE}$  must be exchanged via the name system. We define the procedure for authorization in Algorithm 2.

---

#### Algorithm 2: Authorize

---

**input** : User identity  $ID_{user}$   
requesting party  $ID_{rp}$   
Set of attributes  $A$   
Master secret key  $msk_{ABE}$

**output**: A ticket  $t$

- 1  $tags \leftarrow \{\text{Concat}(a.name, a.version) \mid a \in A\};$
  - 2  $sk_{ABE} \leftarrow \text{Keygen}(msk_{ABE}, tags);$
  - 3  $ct \leftarrow \text{Enc}(pk_{rp}, sk_{ABE});$
  - 4  $rnd \leftarrow_R \mathbb{R};$
  - 5 **Publish**( $ID_{user}, rnd, ct$ );
  - 6  $names \leftarrow \{a.name \mid a \in A\};$
  - 7  $t \leftarrow (ID_{user}, ID_{rp}, names, rnd);$
  - 8 **return**  $t$ ;
- 

First, we generate a set of *tags* that correspond to the respective encrypted records the requesting party shall be authorized to access. After the  $sk_{ABE}$  is generated using the users’  $msk_{ABE}$ , it is encrypted using the public key  $pk_{rp}$  of the requesting party. Then, a random label *rnd* is generated under which the encrypted  $sk_{ABE}$  is published in the user namespace. The random label *rnd*, the user identity  $ID_{user}$ , the requesting party identity  $ID_{rp}$  and the attributes that the requesting party is authorized to access are assembled into a ticket  $t$ . Updates to  $sk_{ABE}$ , made necessary for example due to revocations, are published by the user and retrieved by the requesting party using the same random label *rnd*. Similarly to attribute records, we define key records to have a unique type of “ABE\_KEY”.

### D. Deletion

Removing attributes is not as simple as removing the respective records from the namespace. First, the attribute record may still be resolvable in the name system until the records expire and it is purged from the cache. Requesting parties that

are authorized to access this attribute then must be prohibited from accessing any future incarnations of this attribute. This is important as to not risk any unwanted side-effects where unauthorized parties may still be able to decrypt the attribute. For this, the attribute tag version must be incremented before a new attribute with the same name is issued. A reclaimID implementation must keep track of this state by either keeping the attribute with an empty placeholder value or by having a local database that contains the versioning information. This implementation detail is out of scope of the reclaimID design and we only define the procedure for deletion itself in Algorithm 3.

---

**Algorithm 3: Delete**


---

**input** : User attribute  $a$   
User identity  $ID_{\text{user}}$

- 1 **Depublish**( $ID_{\text{user}}, a.name$ );
- 2  $a.version++$ ;
- 3 **for each** ticket  $t$  issued by  $ID_{\text{user}}$  **do**
- 4      $A_t \leftarrow \{x \mid x \in A \setminus a \wedge x.name \in t.names\}$ ;
- 5     **Authorize**( $ID_{\text{user}}, t.ID_{\text{rp}}, A_t, msk_{\text{ABE}}$ );
- 6 **end**

---

The **Delete()** procedure starts off by de-publishing the respective attribute record from the namespace and then incrementing the attribute version. After, all authorized parties (i.e. all issued tickets) that have access to this attribute are re-authorized to access all attributes they had access to before *except* the deleted attribute.

#### E. Update

When the user modifies the attribute value the respective record in the name system must be updated accordingly. Naively, it is possible to simply combine a **Delete()** and a **Store()** call. But since we defined the **Delete()** procedure to increment the attribute version such an approach would require the user to reissue all existing ABE keys to the relevant requesting parties. Consequently, updating the attribute is simply a call to **Store()** after updating the attribute value. This update will only take effect *after* the identity record expires and only then will the updated value be resolvable by authorized parties. As the tag used to encrypt the attribute does not change, previously authorized requesting parties will be able to decrypt the updated record data with their existing keys.

#### F. Retrieval

To retrieve an attribute  $a$  of identity  $ID_{\text{user}}$  an authorized requesting party  $ID_{\text{rp}}$  must perform a lookup in the name system. The name to lookup is the attribute name, e.g. “email”. If the attribute exists, the response from the name system will contain the encrypted attribute value record  $R$ . As elaborated above,  $sk_{\text{ABE}}$  contains a set of tags that allows it to be used in the decryption of all attribute records that  $ID_{\text{rp}}$  is authorized to access. To retrieve  $sk_{\text{ABE}}$ ,  $ID_{\text{rp}}$  must first resolve the key

record under the name  $rnd$  in the identity namespace of  $ID_{\text{user}}$ . To do so,  $ID_{\text{rp}}$  must have received the label  $rnd$  out-of-band in a ticket as discussed in the previous section. Given  $sk_{\text{ABE}}$ , the requesting party can decrypt the attribute value using the CP-ABE decryption function **Dec**<sub>ABE</sub>( $\cdot$ ). We formally define the procedure for retrieval in Algorithm 4.

---

**Algorithm 4: Retrieve**


---

**input** : requesting party  $ID_{\text{rp}}$   
Ticket  $t$

- 1  $ct \leftarrow \text{Resolve}(t.ID_{\text{user}}, t.rnd)$ ;
- 2  $sk_{\text{ABE}} \leftarrow \text{Dec}(sk_{\text{rp}}, ct)$ ;
- 3 **for all** attribute names  $n \in t.names$  **do**
- 4      $R \leftarrow \text{Resolve}(t.ID_{\text{user}}, n)$ ;
- 5      $attribute \leftarrow \text{Dec}_{\text{ABE}}(sk_{\text{ABE}}, R)$ ;
- 6 **end**

---

Note that most name systems allow queries for attribute records to be executed in parallel, which allows the for-loop in the **Retrieve()** procedure to be parallelised.

#### G. Revocation

We define revocation – as opposed to deletion – as the process to revoke access of a specific requesting party to user attributes in reclaimID. Revocation schemes for ABE are often quite complex and inefficient. In our case we also have to take into account user key distribution and name system properties.

In fact, the performance impact caused by cryptographic operations is not as critical in our design for two reasons: First, regeneration of keys and re-encryption can be done locally in the background after it is initiated by the user. Second, from a requesting party point of view, even if access to a particular attribute is revoked there was a time in past where access was granted. So, revoking access on currently accessible data is not important in our design.

Revocation of access in reclaimID is used to prevent the decryption of an attribute record using a specific user key  $sk_{\text{ABE}}$  of a requesting party. Any attribute that the requesting party was authorized to access at any time in the past was most likely already retrieved and possibly even persisted locally. Consequently, it is not a goal to revoke access to the *current* attributes that were already published. The primary goal is to prohibit a requesting party from continuously accessing up-to-date attribute information *in the future*.

Our revocation scheme is enforced through attribute versioning. As elaborated in the previous sections, an attribute record is encrypted using a tag that is a concatenation of the attribute name and version. When access of a requesting party to an attribute is revoked, we simply increment the attribute version. Then, we again publish the encrypted attribute value to the name system.

Any other requesting parties also authorized to access the same attribute must be issued new user keys containing updated tags. The updated keys are published under the same respective labels  $rnd$  and can be resolved if needed. Using

this approach we can limit the amount of re-generated user keys to the number of requesting parties that share one or more attribute authorizations with the requesting party that had its access revoked<sup>6</sup>. Another advantage of this approach becomes evident when taking the first authorization of an RP into account: Initially, it suffices to create a new user key with the current attribute versions and transfer it to the RP. As the ciphertext does not need to be updated in this case, the attribute records currently in the name system can then instantly be decrypted by the RP. We define our **Revoke()** procedure in Algorithm 5.

---

**Algorithm 5:** Revoke

---

```

input : A ticket  $t_{rp}$  issued to RP
1 for each attribute  $a$  of  $t_{rp}.user$  do
2   if  $a.name \in t_{rp}.names$  then
3      $a.version++$ ;
4     Store( $t_{rp}.ID_{user}, a$ );
5   end
6 end
7 for each ticket  $t \neq t_{rp}$  issued by  $t_{rp}.ID_{user}$  do
8   if  $\emptyset \neq t.names \cap t_{rp}.names$  then
9      $A_t \leftarrow \{a \mid a \in A \wedge a.name \in t.names\}$ ;
10    Authorize( $t.ID_{user}, t.ID_{rp}, A_t, msk_{ABE}$ );
11  end
12 end

```

---

#### H. Implementation Considerations

As mentioned above, the concrete choice of ABE and name system used in an implementation partially determines the security properties of reclaimID. In the following we discuss the most important aspects that are relevant and must be considered by implementers.

1) *Name System*: While theoretically all name systems are suitable for our design, practically only name systems with strong security guarantees are reasonable choices for building a decentralised IdP. An insufficiently resilient name system might be subject to denial of service attacks, rendering the IdP useless. Also, bulk collection and enumeration of attributes is unwanted, as it exposes organizational and/or trust relationships.

Grothoff et al. [12] have categorized state of the art name systems according to their security properties including integrity and availability with respect to strong attacker models. According to the study, Namecoin and GNS exhibit security properties absent in most other name systems, such as resistance to man-in-the-middle manipulation, request and response privacy and censorship resistance. The authors conclude that the choice of name system is – in addition to security considerations – depending on organizational aspects of the ecosystems.

For instance, DNSSEC relies on the distributed design of redundant DNS servers as well as caching. Even more important is the fact that domain names in DNS are highly regulated, making it a semi-centralized system where only the technology is distributed. This organizational architecture degrades resilience in the face of strong attackers. Not to mention that DNSSEC further suffers from a design weakness that results in a privacy issue regarding namespace enumeration<sup>7</sup> for which a mitigation was proposed by Goldberg et al [14].

In peer-to-peer-based, decentralised name systems such as Namecoin and GNS this is not a problem. Namecoin is a blockchain-based name system, availability is addressed by having all records replicated by all participants in a local ledger. No central authorities are required to manage the structure of the name system since integrity is ensured through the consensus mechanism. Of course, it is trivial to enumerate namespaces in Namecoin due to the nature of a public ledger. Further, blockchain-based decentralised systems are still quite new and possibly prone to various attacks on the ledger itself [15], [16].

GNS, on the other hand, is a name system built on top of a distributed hash table (DHT). It prevents namespace enumeration and also features an efficient response caching mechanism. GNS is a petname system that inherently mitigates name squatting by not having globally unique names. Records are generally not considered confidential in a name system as its primary use-case is resource discovery. As such, Namecoin and DNSSEC do not protect the contents of resource records or namespaces in any way. In GNS, record, query and response data is protected. Grothoff et al. [12] discuss the respective mechanisms to achieve this. Records are, by default, encrypted using a symmetric key that can be derived from its record label and namespace. Further, record queries are protected through a “query privacy” utilising a similar approach. So unlike DNS, for example, queries cannot be trivially observed by third parties. This prevents an attacker from easily profiling interactions between users and service. It should be noted that blockchain-based name systems do not suffer from this particular problem as queries are basically just lookups in a local database.

Based on the above, we conclude that peer-to-peer-based approaches, such as Namecoin or GNS, should be preferred over semi-centralized, distributed systems such as DNS.

2) *ABE Scheme*: When it comes to ABE we have to decide between the two major flavors: Key-Policy ABE (KP-ABE) and Ciphertext-Policy ABE (CP-ABE). As discussed by Borgh et al. [17], [18] the use of CP-ABE is more intuitive than KP-ABE from the point of view of the encrypter. This is due to the fact that if keys are issued by a third party, it is non trivial for the encrypter to know who has access to the plaintext. In our case, the encrypter is the same entity as the key issuer so this limitation does not hold. Furthermore, policies in our proposed system are composed of a single

<sup>6</sup>As opposed to re-bootstrapping the whole ABE scheme and issuing/publishing new keys for all RPs

<sup>7</sup><https://dnscurve.org/espionage2.html>, accessed 2017/12/26

tag<sup>8</sup>. Thus, we are not dependent on having the ability to use conjunctive or disjunctive policies in either ciphertext or key material. Therefore the chosen ABE flavor is independent from the system design. In fact, the system properties would not significantly change. The only thing what would differ is the underlying encryption logic that is ideally never exposed by reclaimID anyway. One could argue that reclaimID would benefit from recent ABE scheme's like FAME [19] in terms of performance regarding the cryptographic operations. But FAME's performance benefits only take effect when policies become more complex. The reason for that is because in FAME, there is always a fixed number of pairing operations while in other approaches, such as BSW CP-ABE [20], the number of pairings is determined by the complexity of the policy. In addition to that, the space complexity of BSW CP-ABE ciphertexts and user keys is less than in FAME because it uses two group elements per attribute while BSW CP-ABE uses three group elements per attribute. FAME uses type-3 pairings, which are more efficient than the type-1 pairings used in the original BSW CP-ABE [21]. If more complex policies including significantly higher number of attributes are required, the use of the FAME scheme in an implementation should be preferred.

#### IV. IMPLEMENTATION

We have implemented<sup>9</sup> reclaimID on top of GNS, a name system that is part of the GUNet peer-to-peer framework<sup>10</sup>. Further, we use a slightly modified but functionally equivalent version of the CP-ABE implementation *libbswabe*<sup>11</sup>. We chose this implementation because of its general availability and because we presume that variations of this scheme can provide performance improvements. The use of *libbswabe* can be considered as a performance baseline with regards to our performance evaluations.

In the following section, we discuss what implications our choice of name system and ABE scheme have on the security properties of our implementation. Additionally, we present and discuss the results of performance tests that we carried out. Finally, we show how reclaimID can practically be integrated into an OpenID Connect 1.0 Identity Provider (OIDC IdP) to provide a standards-compliant way of using our design.

##### A. Security Properties

Many security properties of our reclaimID implementation stem from GNS:

**Availability:** GNS built on top of the  $R^5N$  [22] DHT.  $R^5N$  is designed to perform well in restricted-route environments with malicious participants. reclaimID directly benefits from the strong security guarantees of  $R^5N$ , such as high resilience and censorship-resistance. Records are replicated and stored redundantly in  $R^5N$  under a key that is generated by hashing

the namespace public key with the query name. Further, GNS is a petname system where users register names in their own local namespace. This is unlike DNS, for example, since DNS has a global unique root zone managed by a single organization that delegates sub-hierarchies to other organizations. The petname approach mitigates the name squatting problem where attackers register names in bulk before legitimate users.

**Integrity:** Other name systems, such as DNSSEC or Namecoin, sign either whole namespaces or single resource records. In GNS, record sets are aggregated by label and signed using a key derived from the namespace private key before being published in the DHT. The DHT has a built-in signature verification ensuring that only valid results are cached and returned.

**Privacy and Confidentiality:** Records in GNS are encrypted using a key derived from the query label and namespace public key before they are published into the network. In a similar fashion GNS realises query privacy. The namespace private key is derived using the label of a record to sign the records aggregated by label. The corresponding derived public key is published along with the record, thus allowing any peer to verify the validity of the record and avoid storing corrupted content. On the other hand, this does not leak information on the namespace owner. The query key for records under a specific name is constructed by hashing the name and the public key of the respective namespace. Even if the record data itself is unprotected, which is not the case in our design, a peer in GNS that stores a record or observes a query can only access its contents if the respective name and key are known to that peer. Finally, we additionally use the CP-ABE library *libbswabe* in accordance with the reclaimID design.

##### B. Performance Evaluation

Since our implementation uses GNS, which is built on top of the  $R^5N$  DHT, performance is a concern. We tested our implementation with regards to the following aspects:

- Median time to retrieve a user key
- Median time to retrieve an attribute
- Performance impact of caching in GNS on attribute retrieval
- Performance impact of the number of nodes in the network

Our test setup consists of a virtual host with 32 vCPUs at 2.3 GHz and 32GB of RAM. To determine the median time it takes to resolve a key and subsequently an attribute, we bootstrap a GUNet network  $N$ . Before every test run,  $N$  is re-bootstrapped to ensure that any caches are purged. In each run, we repeat one test 10 times. We define the test to consist of the following steps:

- 1) Randomly choose a node  $A$  that acts as a user and a node  $B$  that acts as a requesting party from  $N$ .
- 2)  $A$  and  $B$  create identities in GNS exchange public keys.
- 3)  $A$  creates and stores a test attribute  $a$  and authorizes  $B$  to access it.
- 4) Simulate an out-of-band handover of the respective authorization ticket  $t$  to  $B$

<sup>8</sup>An attribute name concatenated with a version number

<sup>9</sup>In <https://gnunet.org/git/gnunet.git> as part of the identity provider subsystem

<sup>10</sup><https://gnunet.org>, accessed 11/29/2017

<sup>11</sup><http://acsc.cs.utexas.edu/cpabe/>, accessed 2017/29/11



- 5)  $B$  retrieves the ABE user key  $sk_{ABE}$
- 6)  $B$  retrieves the attribute  $a$  and decrypts the attribute value.

Each time the test is repeated, we randomly choose a different node  $B \in N$  while  $A$  stays fixed. We measure the time it takes each  $B$  to resolve the user key (Step 5) and the attribute (Step 6). Between each test, we do not tear down or re-bootstrap the network so that we can investigate the impact of caching on retrieval times. After 10 tests, we tear down the network to conclude the test run.

We execute 1000 test runs to increase the reliability of our dataset. Further, as we want to investigate the impact of the size of the network, we ran our experiment for  $|N| = 50$ ,  $|N| = 100$ ,  $|N| = 150$  and  $|N| = 200$  nodes, respectively.

We expect the key and attribute retrieval times within a single test run to initially exhibit a high variance. However, successive attribute resolutions within a single test run by different parties are expected to be faster and show increasingly less variance due to caching kicking in. We do not expect the same behaviour for the key retrieval. In regards to the network size, we expect it to have a negative influence on retrieval times and the respective variance with increasing node count.

1) *Results:* Figure 2 is a comparison between median attribute retrieval times across all test runs in differently sized networks. The data suggests that the median times for attribute retrieval increases with the size of the network. At the same time, the retrieval times appear to converge, albeit slower with increasing node count.

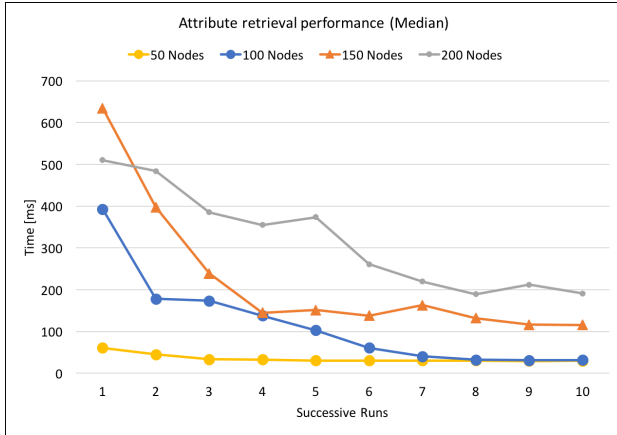


Fig. 2: Median attribute retrieval times across all test runs for network sizes of 50, 100, 150, and 200 nodes.

In Figure 3, we can see that the time it takes to resolve a user key varies with a median of around 200 ms. As expected, the variance is quite high throughout all 10 successive tests across all test runs. Since we randomly choose our nodes, performance largely depends on the routing and topological distance between  $A$  and  $B$ . Our experimental setup consists of a clique topology. However, considering no caching can be leveraged due to the individuality of the query, the observed performance is still practical. This observation matches with our expectations and suggests that the initial user key after

authorization should be transferred to the requesting party out-of-band and not via the name system. The fact that the initial resolution exhibits a particularly high variance and median retrieval time supports this as well.

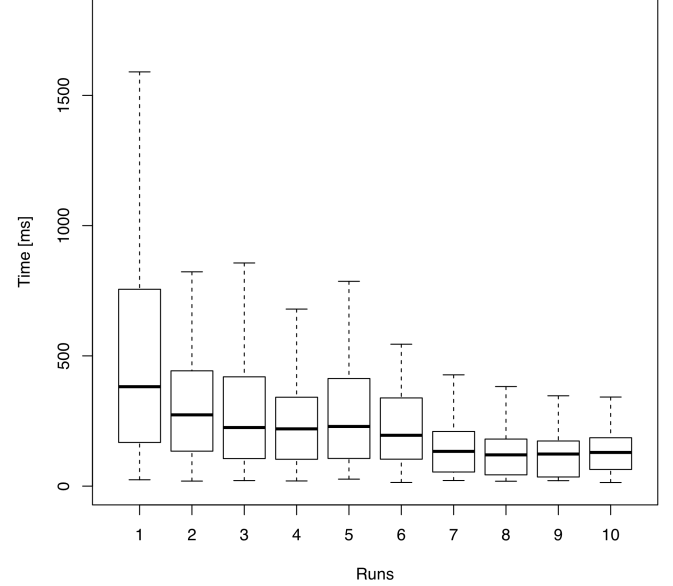


Fig. 3: User key retrieval performance of user keys for a network size of 100 nodes.

In Figure 4, we can see that the retrieval times for attributes also initially exhibit a high variance. But, retrieval times quickly converge to low median times at less than 100ms with a low variance. This dataset nicely illustrates the effects of attribute caching in the name system in our implementation. After the first few requesting parties are authorized to access attributes and have resolved the respective records, resolution times improve greatly. This confirms our expectation that we can leverage caching of queries and responses in GNS for attributes and it impacts the systems performance positively.

2) *Discussion:* Our results show that the implementation quickly converges into a reasonably well performing system. In asynchronous use cases, where user data is retrieved by requesting parties without user interaction after an initial authorization flow, resolution times of up to 100ms are acceptable. Increasing attribute counts should not negatively impact resolution performance as attributes can be resolved in parallel.

However, since key resolution will not benefit from caching in the current implementation, we recommend an initial out-of-band transfer of the key as part of a authorization protocol such as the OIDC authorization code flow. Further, keys may change from time to time, in particular due to a revocation initiated by the user. When the attributes are required for processing it might already be too late and the attribute can no longer be decrypted with the old key. It is reasonable for requesting

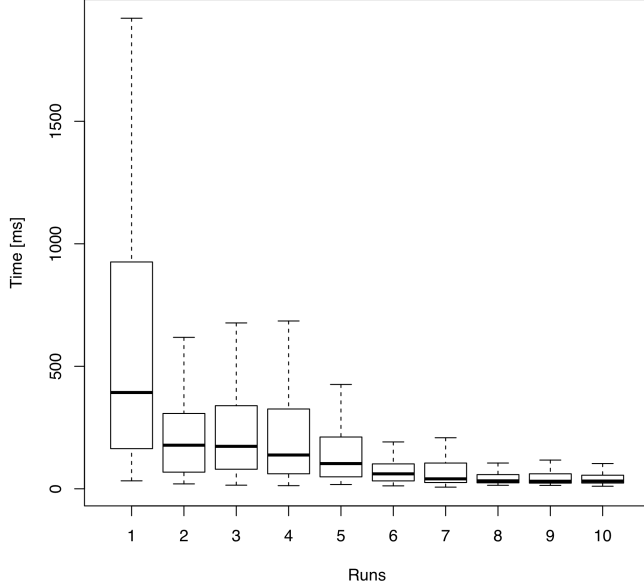


Fig. 4: Attribute resolution performance for a network size of 100 nodes.

parties to regularly resolve their respective key records, rather than only resolve the updated key.

### C. OpenID Connect Integration

As discussed in our design, an out-of-band exchange of the authorization ticket and possibly even the user key  $sk_{ABE}$  can be done using an authorization protocol. Instead of proposing our own protocol, we show that our implementation can be abstracted through an HTTP-based OIDC 1.0 compliant authorization flow [23]. Like the specification, in the following we assume that all HTTP exchanges between the user and the service are secured using TLS server authentication. In traditional OIDC deployments, a single service serves well defined endpoints to users and requesting parties. As our implementation is a decentralised service any participant can take both the role of a user as well as the role of a requesting party. For this reason, all participants run a local reclaimID instance that exposes the respective OIDC endpoints.

Let us consider our original social network use case illustrated as OIDC flow in Figure 5: We assume that a user Alice manages her user attributes – in particular her email address – using reclaimID through a web frontend running on her local machine (0). Consequently, the email record  $R_{sk_{rp}}$  containing Alice’s address is stored in GNS (1). She registers to a social networking service at a website (2). The website offers a “reclaimID” button that – when pressed – initiates an OIDC authorization code flow in which the service requests access to Alice’s “email” attribute. Alice presses the button and her browser is redirected to the OIDC authorization endpoint that is exposed by her local reclaimID installation

(3a). Alice consents to the authorization request (3b) and the authorization procedure as defined in Section III-C is executed (4) that stores the user key  $sk_{rp}$  in the record  $R_{sk_{rp}}$ . Next, the browser is redirected back to the website (5a,5b) along with an “authorization code”. We use the OIDC authorization code to piggyback a reclaimID ticket that includes the  $rnd$ .

The service exchanges the code at the OIDC token endpoint (6). This request triggers the retrieval of Alice’s email attribute as defined in Section III-F (7). The email attribute is wrapped inside a JSON Web Token and returned in the OIDC token response (8). The response additionally contains an opaque access token that can be used against the OIDC userinfo endpoint. When Alice is offline, a request to the userinfo endpoint simply reuses the ticket obtained in (5b) to obtain fresh user attributes from reclaimID (6).

1) *GNS naming and OpenID Connect*: Unlike DNS, GNS is a petname system. In DNS, there is a global unique root zone managed by a single organisation. The petname property in GNS comes in handy since the local IdP services for the user and the requesting party can both be addressed using, e.g. “identity.gnu”. For each entity, we can assume that the system is configured to map this name to the respective local hosts. In our design, this host also runs the respective local reclaimID service. When building HTTP-based authorization protocols on top our design, such as OIDC, this is useful as the specification presumes that there is a single service instance reachable under one domain name. Using GNS, this is actually the case while at the same time the service behind the name is decentralised.

## V. RELATED WORK

Work related to ours mainly targets decentralised and user-centric identity management. As the following discussion shows, this does not necessarily include a decentralised IdP, which we claim is an essential aspect in a truly user-centric system.

DP5 [24] is a privacy-friendly presence notification service that through the use of asymmetric pairing functions and pseudo-random functions provides a secure and private personal information retrieval (PIR). While DP5 addresses the most relevant privacy and security issues, it does not address availability as reclaimID does with a redundant DHT. A powerful attacker might coerce a DP5 service provider to discontinue its services without having to deal with any of the users trying to share information. Participants in reclaimID are not individually protected against such an attack, but the lack of central service instances mitigates the collapse of the whole system.

CONIKS [25] is an approach originally designed for user-centric key transparency. The authors propose a system that does not require a single, centralised third-party to monitor mappings from names to keys, such as from a domain name to its corresponding certificate. Rather, CONIKS allows users and services to participate in a privacy-preserving protocol that allows them to audit providers of such mappings for non-equivocation. In contrast to reclaimID, CONIKS relies

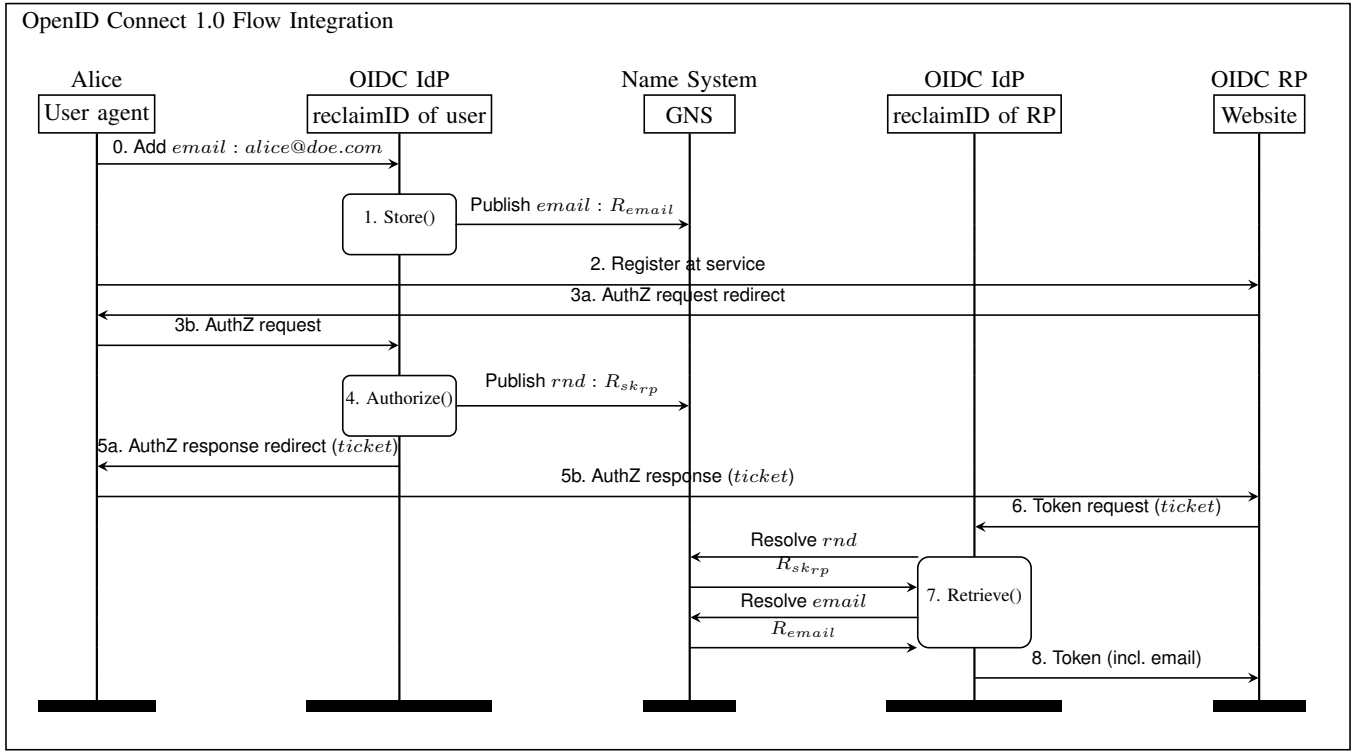


Fig. 5: An example authorization flow and attribute retrieval integrated into OpenID Connect (OIDC). Protocol steps 1,2,4,5 and 8 include the standard OIDC authorization Code Flow. Steps 3, 6 and 7 are interactions between the respective local reclaimID and GNS components.

on centralised IdPs that serve the respective key material. While misuse of those IdPs can be detected, COINKS does not prevent attacks on these central services. This issue is directly addressed by the authors behind ClaimChain [26].

ClaimChain also primarily addresses the key verification use-case through the use of hash chains and cross-referencing. Similar to reclaimID, this approach mitigates the need for trusted centralised IdPs and replaces them with a decentralised protocol and data structure. While their design proposes flexible, decentralised data structures, ClaimChain does not address the underlying transportation of such. This is in particular problematic with regards to offline access of claims, i.e. situations where offline users cannot be addressed directly by a requesting party. The authors of ClaimChain state that in this case, claims can be stored at online services for users to interact with them, which would again introduces a centralised component. Our approach addresses this issue by also covering the transportation layer and respective protocols for offline access of attributes.

NameID [5] is a recent decentralised IdM approach that uses identities and attributes located in the Namecoin [13] blockchain. Those identities and attributes are in complete control by the respective user and cannot be edited or deleted by the IdP. As a consequence, all identity attributes in NameID are inherently self-issued, meaning that there is no third party authority issuing or certifying attributes. Users authenticate by providing proof-of-possession of the respective wallet private

key a claimed identity is associated with. While NameID fully decentralises attribute management, this comes at the cost of privacy. All identities and attributes are managed in plain text in the Namecoin blockchain and are thus publicly readable to anybody. We address this issue with reclaimID by a cryptographic access control on attributes using ABE.

Another approach to decentralised IdM by Schanzenbach et al. [27] is also using a decentralised name system as backend for self-issued identity attributes. Unlike NameID the authors also show how the IdP service itself can be decentralised but their implementation does not feature OpenID Connect compatibility. Identity attributes that are shared with a requesting party are aggregated and published a single resource record per requesting party. However, most name systems are quite unresponsive to changes in namespaces and thus heavily rely on caching which most likely negatively impacts the systems performance.

The approach taken by uPort [28] is similar to ours in that it is focused on self-issued (in uPort called “self-sovereign”) identity data stored in a decentralised system. While we use the name system’s distributed storage (specifically in form of a DHT in case of GUnet), uPort stores public profile data in IPFS [29]. Identity data is managed using smart contracts on the Ethereum [30] blockchain. Currently this approach stores identity data in plain text in the IPFS which raises the same privacy concerns as NameID. uPort further requires a central service to share private identity attributes between user and

requesting party – something that uPort solves querying the decentralised name system.

## VI. CONCLUSION AND FUTURE WORK

We introduced reclaimID, a decentralised service for self-sovereign identity management. reclaimID differs from existing approaches in that it combines three main aspects: user-managed attributes without a central party, the complete dissolution of a central IdP into a decentralised query protocol, and privacy-preserving features such as ABE-based access control to sensitive attribute data.

By a practical implementation of reclaimID based on the name system GNS, we have shown that the approach is valid and achieves the functional requirements of an identity management system. In a series of experiments we evaluated the performance and scalability of our system with the result that in its current state, the implementation of reclaimID is able to serve small to medium applications with up to a few hundreds of participants in production. Performance optimizations and deployments to large scale testbeds such as PlanetLab are future work.

Through the use of ABE, reclaimID provides an access control layer to user attributes which would otherwise be stored world-readable, as shown by related work. We consider the ability to authorize access to user attributes not only essential for preserving the user's privacy, but also to enable new use cases which are currently "solved" by workarounds that again negatively impact privacy. For instance, particularly in the context of the Internet of Things, devices often need to be authenticated entities that exist in different security domains. Such domain-spanning authentication and authorization scenarios are becoming the rule rather than the exception in an environment full of interconnected devices, but the approaches to this challenge still follow the traditional pattern of adding "trusted" third parties (e.g. for device vendors) or complex cross-certification-like constructs. reclaimID allows a more elegant solution to this challenge by providing an inter-domain identity management infrastructure to establish trust relationships directly between entities, e.g. between a specific device of a vendor and an application by another vendor.

Besides further performance improvements to address near-realtime requirements, our future work will thus address the extension of reclaimID to support privacy-preserving attribute-based credentials.

## ACKNOWLEDGMENTS

This work was developed in the Fraunhofer Cluster of Excellence «Cognitive Internet Technologies»<sup>12</sup>

## REFERENCES

- [1] P. Gola, R. Schomerus, and C. Klug, *BDSG - Bundesdatenschutzgesetz : Kommentar*. Mnchen: Beck, 8. bearbeitete und ergnzte auflage ed., 2005.
- [2] C. N. de l'Informatique et des Libertés (French data protection authority), "Decision no. 2016-007 of january 26, 2016 issuing formal notice to facebook inc. and facebook ireland," January 2016.
- [3] M. Wachs, M. Schanzenbach, and C. Grothoff, "A censorship-resistant, privacy-enhancing and fully decentralized name system," in *Cryptology and Network Security*, pp. 127–142, Springer, 2014.
- [4] M. Wachs, M. Schanzenbach, and C. Grothoff, "On the feasibility of a censorship resistant decentralized name system," in *Foundations and Practice of Security*, pp. 19–30, Springer, 2014.
- [5] D. Kraft, "Nameid," 2017. <https://nameid.org/>.
- [6] R. Deibert, J. Palfrey, R. Rohozinski, and J. Zittrain, *Access contested: security, identity, and resistance in Asian cyberspace*. MIT Press, 2011.
- [7] B. Gellman and L. Poitras, "Us, british intelligence mining data from nine us internet companies in broad secret program," *The Washington Post*, vol. 6, 2013.
- [8] L. Hui and M. Rajagopalan, "At sina weibos censorship hub, chinas little brothers cleanse online chatter," *Reuters*, September, vol. 11, 2013.
- [9] G. N. A. R. Jan Camenisch, Anja Lehmann, "Privacy-preserving auditing for attribute-based credentials." *Cryptology ePrint Archive*, Report 2014/468, 2014. <https://eprint.iacr.org/2014/468>.
- [10] J. Camenisch and E. Van Herreweghen, "Design and implementation of the idemix anonymous credential system," in *Proceedings of the 9th ACM conference on Computer and communications security*, pp. 21–30, ACM, 2002.
- [11] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, "Snarks for c: Verifying program executions succinctly and in zero knowledge," in *Advances in Cryptology—CRYPTO 2013*, pp. 90–108, Springer, 2013.
- [12] C. Grothoff, M. Wachs, M. Erment, and J. Appelbaum, "Towards secure name resolution on the internet," *Computers & Security*, 2018.
- [13] Namecoin, "Namecoin is a decentralized open source information registration and transfer system based on the bitcoin cryptocurrency," February 2016. <https://namecoin.info/>, accessed 2016/02/23.
- [14] S. Goldberg, M. Naor, D. Papadopoulos, L. Reyzin, S. Vasant, and A. Ziv, "Nsec5: Provably preventing dnssec zone enumeration," in *NDSS*, 2015.
- [15] A. Maria, Z. Aviv, and V. Laurent, "Hijacking bitcoin: Routing attacks on cryptocurrencies," in *Security and Privacy (SP), 2017 IEEE Symposium on*, IEEE, 2017.
- [16] I. Giechaskiel, C. Cremers, and K. B. Rasmussen, *On Bitcoin Security in the Presence of Broken Cryptographic Primitives*, pp. 201–222. Cham: Springer International Publishing, 2016.
- [17] J. Borgh, "Attribute-based encryption in systems with resource constrained devices in an information centric networking context," 2016.
- [18] J. Borgh, E. Ngai, B. Ohlman, and A. M. Malik, "Employing attribute-based encryption in systems with resource constrained devices in an information-centric networking context," in *Global Internet of Things Summit (GloTS), 2017*, pp. 1–6, IEEE, 2017.
- [19] S. Agrawal and M. Chase, "Fame: Fast attribute-based message encryption,"
- [20] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Security and Privacy, 2007. SP'07. IEEE Symposium on*, pp. 321–334, IEEE, 2007.
- [21] S. D. Galbraith, K. G. Paterson, and N. P. Smart, "Pairings for cryptographers," *Discrete Applied Mathematics*, vol. 156, no. 16, pp. 3113–3121, 2008.
- [22] N. S. Evans and C. Grothoff, "R5n: Randomized recursive routing for restricted-route networks," in *NSS*, pp. 316–321, 2011.
- [23] N. Sakimura, J. Bradley, M. Jones, B. de Medeiros, and C. Mortimore, "Openid connect core 1.0 incorporating errata set 1," 2014. [http://openid.net/specs/openid-connect-core-1\\_0.html](http://openid.net/specs/openid-connect-core-1_0.html).
- [24] N. Borisov, G. Danezis, and I. Goldberg, "Dp5: A private presence service," *Proceedings on Privacy Enhancing Technologies*, vol. 2015, no. 2, pp. 4–24, 2015.
- [25] M. S. Melara, A. Blankstein, J. Bonneau, E. W. Felten, and M. J. Freedman, "Coniks: Bringing key transparency to end users," in *USENIX Security Symposium*, pp. 383–398, 2015.
- [26] B. Kulynych, M. Isaakidis, C. Troncoso, and G. Danezis, "Claimchain: decentralized public key infrastructure," *arXiv preprint arXiv:1707.06279*, 2017.
- [27] M. Schanzenbach and C. Banse, *Managing and Presenting User Attributes over a Decentralized Secure Name System*, pp. 213–220. Cham: Springer International Publishing, 2016.
- [28] uPort, "uport whitepaper," November 2017. [https://whitepaper.uport.me/uPort\\_whitepaper\\_DRAFT20170221.pdf](https://whitepaper.uport.me/uPort_whitepaper_DRAFT20170221.pdf), accessed 2017/11/12.
- [29] IPFS, "Ipfs," November 2017. <https://ipfs.io/>, accessed 2017/11/12.
- [30] Ethereum, "Ethereum," November 2017. <https://www.ethereum.org/>, accessed 2017/11/07.

<sup>12</sup><http://www.cit.fraunhofer.de>