Decentralized Authentication for Self-Sovereign Identities using Name Systems (DASEIN)

Christian Grothoff Martin Schanzenbach Annett Laube
Emmanuel Benoist Pascal Mainini

October 21, 2018

Abstract

The GNU Name System (GNS) is a fully decentralized public key infrastructure and name system with private information retrieval semantics. It serves a holistic approach to interact seamlessly with IoT ecosystems and enables people and their smart objects to prove their identity, membership and privileges — compatible with existing technologies.

In this report we demonstrate how a wide range of private authentication and identity management scenarios are addressed by GNS in a cost-efficient, usable and secure manner. This simple, secure and privacy-friendly authentication method is a significant breakthrough when cyber peace, privacy and liability are the priorities for the benefit of a wide range of the population.

After an introduction to GNS itself, we show how GNS can be used to authenticate servers, replacing the Domain Name System (DNS) and X.509 certificate authorities (CAs) with a more privacy-friendly but equally usable protocol which is trustworthy, human-centric and includes group authentication. We also built a demonstrator to highlight how GNS can be used in medical computing to simplify privacy-sensitive data processing in the Swiss health-care system. Combining GNS with attribute-based encryption, we created re:claim, a robust and reliable OpenID Connect-compatible authorization system. It includes simple, secure and privacy-friendly single sign-on to seamlessly share selected attributes with Web services, cloud ecosystems. Further, we demonstrate how re:claim can be used to solve the problem of addressing, authentication and data sharing for IoT devices.

These applications are just the beginning for GNS; the versatility and extensibility of the protocol will lend itself to an even broader range of use-cases.

GNS is an open standard with a complete free software reference implementation created by the GNU project. It can therefore be easily audited, adapted, enhanced, tailored, developed and/or integrated, as anyone is allowed to use the core protocols and implementations free of charge, and to adopt them to their needs under the terms of the GNU Affero General Public License, a free software license approved by the Free Software Foundation.

Contents

Chapter 1

Introduction

Identity and access management (IAM) in today's connected society is complex. Each person needs to remember usernames and passwords for a variety of Web sites or other on-line services. The rise of Internet of Things (IoT) will only make this challenge harder, as authentification against IoT devices is even more complex: How can a person address an IoT device? How can a device recognize its owner? Today, for both questions, each and every device manufacturer provides a different solution.

In this document, we present a general solution for managing identity of users and devices in a connected world. We propose the use of the GNU Name System (GNS) [43], which has been first developed as an alternative to the Domain Name System (DNS) by the GNU project. While the DNS system – the name system which Internet users are universally familiar with today – suffers from a wide range of security, privacy and political issues, its usability is the backbone of all Internet services today. GNS is designed to maintain most of the user experience of DNS, while replacing the DNS protocol with a more decentralized, more secure and privacy-preserving protocol. Furthermore, instead of insisting on a zone tree, GNS relaxes the relationship between zones to that of a directed graph, thereby making it possible for users to name objects at zero registration cost.

In this report, we detail how to use GNS as the foundation for a broadly applicable authentication and identity management system. This includes use cases for *mutual* authentication, authorization and (personal) data sharing. With GNS, it is possible for users to manage identities, identity attributes and credentials without the need for a centralized service provider. Users and services can securely share attributes or credentials simply over the name system in a secure, self-sovereign manner.

The structure of this report is designed to highlight the benefits of GNS-based identity and access management:

- **State of the art:** In Chapter 2, we elaborate on the technical background for GNS and the underlying network technology and cryptography. We further show how state-of-the cryptography in the form of attribute-based encryption (ABE) can be used to enhance the existing properties of GNS, allowing users to securely and selectively share personal data.

- **Usable:** In Chapter 3, we demonstrate the usability of authentication and authorization systems built on GNS in a variety of use cases including network service authenticaion (Section 3.1), the Internet of Things (Section 3.2), personal data sharing (Section 3.3) and health insurance (Section 3.4).
- **Reliable and secure:** We evaluated the reliability of GNS, with a focus on the availability of the network protocol and present the results in Chapter 4. In Chapter 5, we define the security properties and adversary model of GNS, and Chapter 6 discusses GNS's privacy properties.
- **Versatile & compatible:** We present a selection of use cases in Chapter 7, and detail which existing technologies GNS is compatible with in Chapter 8.
- **Affordable and scalable:** We performed a detailed cost evaluation and including performance measurements of GNS operation in Chapter 9.
- **Open:** We show that the solution is unencumbered free and open source software in Chapter 10.

In the remainder of this chapter, we give an overview over how the use of GNS benefits users, things as well as organizations in identity and access management by leveraging the unique properties of GNS.

1.1 For users

In GNS, identities are similar to subdomains in the DNS system. Owners of subdomains have complete control over their domain namespace. Further, users can delegate responsibility for parts of their namespace. For example, the responsible for the name `ch` (we call that person: owner) can delegate to another identity (i.e. public key) for any label inside of the `ch` domain. They could add `bfh` inside the `ch` domain for the BFH, creating `bfh.ch`. The resulting user experience is deliberatly similar to what DNS does, minimizing the learning curve for users.

A major difference to DNS is that the structure of the names created by GNS is not a tree, but a directed graph. The same public key may be referred to as `bfh`, `bfh.ch`, or `bfh.fr`. It can also be referred to as `bfh.musterman`, if it was added to the name space by the owner of `musterman`.

On most of the Internet applications (Web sites and/or smartphones apps) the users need to register and to create a username and password. For uniqueness purpose, Web sites often use email address as username. This has two advantages: give a unique identifier, and propose a way to reset the password. This gives also the opportunity to firms to contact prospects through email.

We propose to instead use GNS for authentication in Internet applications. At the registration, users give a GNS identity (which is automatically unique). They do not require any username, since they can login using the key pair: The server sends a challenge encrypted with the public key, the user uses the private key to solve the challenge.

Using GNS, we have created re:claim, an OpenID-compatible protocol that allows users to selectively share attributes with Web sites. Users manage their identities using namespaces and manage the respective attribute credentials as records. The attribute value, for example an email address, is encrypted by the user using a secret key and an attribute-based policy. The user authorizes a requesting party to access a set of attribute values by issuing an ABE user key. The user key contains the set of attribute names the requesting party requested and allows decryption of the attribute record values. The use of ABE allows us to enforce access control on attributes that are stored in a name system, where neither the user nor a third party is able to enforce access control decisions. To accelerate user acceptance of re:claim, we have also implemented an OpenID Connect layer. This allows Web sites to use the standardized OpenID protocol to interface with re:claim and request access to attributes. Further, users are provided with a recognizable authorization flow that is already used in "Social Login" systems provided by Facebook or Google.

This way, typing e-mail addresses, shipping addresses, phone numbers or even passport numbers into forms again and again is a thing of the past! We note that only explicitly authorized Web sites will have access to the attributes, and that users can control which attribute(s) they want to share with which Web sites. Users can keep their attributes up-to-date by editing the master version which is stored on their own device(s). Users can revoke access to attributes at any time. Web sites should not even have to store the attributes, as if they still have access they can get the current information via a GNS lookup. This way, Web sites do not have outdated information, do not unnecessarily store private user data, and users have the convenience of not having to type in their private data repeatedly. Section 2.2 includes a detailed technical presentation of re:claim and in Section 3.3, we present a user survey of the re:claim system that we used to improve the usability of the system.

1.2 For things

The user focused approach elaborated above can be extended to devices: The user `musterman` wants to use the device `camera`. He will add the camera in his name space and generate `camera.musterman`. The name `camera.musterman` will be unique and refer explicitly to this very device. Technically, this name will refer to a public key corresponding to the private key of the device. In this way, the device is uniquely defined. The camera may have a sticker with a QR-code for the owner to scan its public key.

The camera will then publish access information in a record set, typically under the empty label (or `@` in DNS bind notation). In this record set, it might publish its IPv4 and IPv6 addresses (as `A` or `AAAA` records) as well as a `PSK` record to identify its *owner*. This PSK record will refer to a shared key used to identify the camera's owner. Then the applications on the camera requiring authentication of the user can use the shared key for access control. As long as either the zone's public key or the label are secret, publishing the record set with the shared key in the GNS zone will maintain the confidentiality of the shared key.

Going beyond the simple addressing use case, we have applied re:claim to the Internet-of-Things (IoT) in an application involving controlled sharing of sensor data. Our "thing" is a sensor board that allows its owner to share the sensor data stream with requesting parties such as Web services. As with the user-focused approach, we tested the IoT use case with users and present the results in Section 3.2.

1.3 For organizations

In a normal PKI environment a central role is played by Certificate Authorities (CA) that are needed to bind the identity of a person with the corresponding public key. GNS is a decentralized system and does not inherently require to have such authorities, but supports their existence.

When GNS is used without registration authorities, anybody can create a zone and give it any nick name, say `donaldtrump`. So, there is no way to be certain that this zone is the one of the president of the United States. While anybody can claim any name, these claims also would not be visible to anyone else!

Thus, to make the system more useful, there should be actors in GNS ecosystem playing a role similar to Certificate Authorities (CAs). However, the central role of CAs in the case of traditional X.509 PKI for DNS is problematic: if one of the accepted CA's delivers a certificate, it will be accepted by everybody. Like DNSSEC — and unlike X.509 CAs — a CA in GNS can only certify names directly under their name. Thus, users can always explicitly see which CAs they trust for each lookup.

A university BFH may have a policy to accept as subdomain only staff working for the university and whose identity has been verified by the university. So `hans_musterman.bfh` refers to a person whose identity is Hans Musterman and works for the `bfh`. For students, the `bfh` may have created a separate zone, `student.bfh`, allowing the university to certify Hans Musterman is a student by creating `hans_musterman.student.bfh`. Note that Hans Musterman may be a student and staff at the same time, in which case both names would refer to the same zone. Furthermore, BFH may itself be recognized as a university, for example by being referenced from the `edu` zone as `bfh.edu`.

In the GNS system, DNS-style delegation can be equivalent to certification. Unlike DNS, a user can select which actor is used to validate the identity of which person. For instance a colleague working for the same firm `thefirm` will be known as `hans_muster.thefirm`, but for someone outside the firm, which do not know the public key of the firm, the same individual might be `hans_muster.thefirm.fr`, where AFNIC verifies the public key of `thefirm`. For people knowing `hans_muster` as `hansi` from the sport club, they will use the identity `hansi.sportclub` or `hansi.sportclub.fr`. All those identities will be valid in a certain context. Hans Musterman may choose to have them point to the same zone (thereby making it clear that they all refer to him), or to separate zones which would allow him to separate his personal and his professional life. This way, each person may have different identities in different contexts. This solution is both better for security, since one do not need to trust all certificates issued by all CA's, and better for privacy, since one person will have the possibility to use different identities (pseudonyms) for different purposes.

GNS, inherited from its SDSI roots, also contains a natural mechanism for managing groups: a zone maybe be used to define records that can be interpreted as a group. Members are identified by following a namespace delegation chain until the end of the chain. If the delegation labels are easily guessed, group membership is public, and otherwise members can choose to prove membership by disclosing the label of the respective record. Consider a classical use case where a company wants to offer special prices on products to university and school students. The company may define a record label "coupon" and delegate it to all entities that are contained in the university and school student groups. Group membership is then managed by the respective authorities (the schools) and they care little about the offer by the company. However, through the use of attribute delegation through the namespace, creating such a group policy is trivial. We present the technical details of this approach in Section 2.3.

Chapter 2

Technical background

2.1 The GNU Name System

This chapter presents the GNU Name System (GNS) [43], a censorship-resistant, privacy-preserving and decentralized name system designed to provide a secure alternative to DNS. As GNS can bind names to any kind of cryptographically secured token, it can double in some respects as an alternative to some of today's public key infrastructures, in particular X.509 for the Web.

The foundation of the GNS system is a petname system [39], where each individual user may freely and securely map names to values. In a petname system, each user chooses a *nickname* as his preferred (but not necessarily globally unique) name. Upon introduction, users adopt the nickname by default as a *label* to refer to a new acquaintance; however, they are free to select and assign any *petname* of their choice in place of—or, in addition to—the nickname. Petnames thus reflect the personal choice of the individual using a name, while nicknames are the preferred name of the user that is being identified.

The second central idea is to provide users with the ability to securely delegate control over a subdomain to other users. This simple yet powerful mechanism is borrowed from the design of SDSI/SPKI [5]. With the combination of petname system and delegation, GNS does not require nor depend on a centralized or trusted authority, making the system provider-independent. Decentralization for the network layer is achieved by using a distributed hash table (DHT) to enable the distribution and resolution of key-value mappings. In theory, any DHT or even a blockchain can be used. However, depending on the properties of the DHT in question, varying degrees of resilience will be the result. As such, the choice of the DHT is crucial for the performance of the system. We evaluate the performance properties of our DHT implementation in Chapter 4.

Finally, GNS is privacy-preserving since both key-value mappings as well as queries and responses are encrypted such that an active and participating adversary can at best perform a confirmation attack, and can otherwise only learn the expiration time of a response. We revisit the privacy properties in depth in Chapter 6 after studying the formal security properties in Chapter 5.

2.1.1 Names, zones and delegations

GNS employs the same notion of names as SDSI/SPKI: principals are public keys, and names are only valid in the local namespace defined by that key. Namespaces constitute the *zones* in GNS: a zone is a public-private key pair and a set of records. GNS records consist of a label, type, value and expiration time. Labels have the same syntax as in DNS; they are equivalent to local identifiers in SDSI/SPKI. Types and values extend concepts from DNS. In particular, GNS uses a secure variant of "NS" records ("PKEY" records) to allow users to delegate control over a subdomain to another user. Record validity is established using signatures and controlled using expiration values. The plaintext records of a zone are managed in a database on a machine under the control of the zone owner.

Names in GNS consists of a sequence of labels, which identifies a *delegation path*. We realize a petname system by having each user manage his own zones. When used as a drop-in DNS resolver, the user's zones augment or override DNS TLDs. Publishing delegations in the DHT allows transitive resolution by simply following the delegation chains. Records can be *local* or *global*, and global records are made available to other users via a DHT. We note that even if a record is "global", its contents and existence are still only knowable to those that know both the label and the zone under which the record is published.

GNS is privacy-preserving since queries and responses are encrypted such that even an active and participating adversary can at best perform a confirmation attack, and otherwise only learn the expiration time of a response. Note that the queries and responses themselves are encrypted, not the connections between a resolver and some authority. As all replies are not just encrypted but also cryptographically signed, GNS provides integrity protection since peers in the DHT cannot tamper with the results without immediate detection and data origin authentication.

Due to the use of a DHT, GNS avoids DNS complications such as glue records and out-of-bailiwick lookups. In GNS, the labels of a name correspond precisely to the lookup sequence, making the complete trust path obvious to the user. Finally, the use of a DHT to distribute records also makes it possible for GNS authorities to operate zones without visible, attributable critical infrastructure that could be used for targeted attacks.

2.1.2 Records in GNS

As GNS is intended to coexist with DNS, most DNS resource records from [28, 41] (e. g., "A", "MX") are used with identical semantics and binary format in GNS. GNS defines various additional records to support GNS-specific operations. These records have record type numbers larger than 2^{16} to avoid conflicts with DNS record types that might be introduced in the future. GNS also introduces several new records:

PKEY for delegation: "PKEY" records securely delegate control over a subdomain to another zone. Repeated delegation allows GNS to achieve transitivity of names. Secure delegation using "PKEY" records is central to GNS; it replaces the tree structure of DNS with a directed graph.

NICK for nicknames: This record type is used to specify the desired *nickname* for a zone. The value of the record consists of a label with the 63-character limit from DNS. If a nickname is desired for a zone, the same "NICK" record is added under *each* label of the respective zone; this ensures that the nickname is part of every response and thus no additional lookup is required to obtain the nickname.

GNS2DNS: "GNS2DNS" records delegate resolution for a subdomain from GNS to DNS.

Similar to "NS" records in DNS, the value in the "GNS2DNS" record is the name of the subdomain in DNS. In addition to the "GNS2DNS" record, the GNS zone must specify "A" or "AAAA" records under the same GNS label which specifies the IP address of the DNS resolver to contact for resolution (this is equivalent to the so-called glue records in DNS). For example:

	Name	RR Type	Value
Q:	www.example.gnu	A	
A:	example.gnu	GNS2DNS	example.com
A:	example.gnu	A	192.0.2.1
Q:	www.example.com *(DNS)*	A	
A:	www.example.com *(DNS)*	A	192.0.2.2

Given the first response, the GNS system will synthesize the DNS name "www.example.com" from the "GNS2DNS" record and the "www" remaining from the GNS name and send a DNS query to the DNS server at 192.0.2.1 based on the glue information from the "A" record. The resolution then continues using DNS. Note that this record type enables delegation to *DNS* from within GNS. Naturally, GNS cannot secure the DNS part of the resolution process.

These are all the special record types that GNS needs. GNS maximizes compatibility with DNS by using the same length limits for labels and names, and the same encoding rules for internationalized names as DNS.

2.1.3 Query privacy

To enable other users to look up records of a zone, all public records for a given label are stored in a cryptographically signed block in the DHT. To maximize user privacy when using the DHT to look up records, both queries and replies are encrypted. Let $x \in \mathbb{Z}_n$ be the ECDSA private key for a given zone and $P = xG$ the respective public key where G is the generator of the elliptic curve. Let $n := |G|$ and $l \in \mathbb{Z}_n$ be a numeric representation of the label of a set of records $R_{l,P}$. Using

$$h := x \cdot l \mod n \tag{2.1}$$

$$Q_{l,P} := H(hG) \tag{2.2}$$

$$B_{l,P} := S_h(E_{\texttt{HKDF}(l,P)} R_{l,P}), hG \tag{2.3}$$

GNS publishes $B_{l,P}$ under $Q_{l,P}$ in the DHT, where S_h represents signing with the private key h, HKDF is a hash-based key derivation function [19] and E represents symmetric encryption based on the derived key. Any peer can validate the signature (using the public key hG) but not decrypt $B_{l,P}$ without knowledge of both l and P. Peers knowing l and P can calculate the query

$$Q_{l,P} = H(lP) = H(lxG) = H(hG) \tag{2.4}$$

to retrieve $B_{l,P}$ and then decrypt $R_{l,P}$.

2.1.4 Zone revocation

In case a zone's private key gets lost or compromised, it is important that the key can be revoked. Whenever a user decides to revoke a zone key, other users must be notified about the revocation. However, we cannot expect users to explicitly query to check if a key has been revoked, as this increases their latency (especially as reliably locating revocations may require a large timeout) and bandwidth consumption for every zone access — just to guard against the relatively rare event of a revoked key. Furthermore, issuing a query for zone revocations would create the privacy issue of revealing that a user is interested in a particular zone. Existing methods for revocation checks using certificate revocation lists in X.509 have similar disadvantages in terms of bandwidth, latency increase and privacy.

Instead of these traditional methods, GNS takes advantage of the P2P overlay below the DHT to distribute revocation information by flooding the network. When a peer wants to publish a revocation notice, it simply forwards it to all neighbors; all peers do the same when they receive previously unknown valid revocation notices. However, this simple Byzantine fault-tolerant algorithm for flooding in the P2P overlay could be used for denial of service attacks. Thus, to ensure that peers cannot abuse this mechanism, GNS requires that revocations include a revocation-specific proof of work. As revocations are expected to be rare special events, it is acceptable to require an expensive computation by the initiator. After that, all peers in the network will remember the revocation forever (revocations are a few bytes, thus there should not be an issue with storage).

In the case of peers joining the network or a fragmented overlay reconnecting, revocations need to be exchanged between the previously separated parts of

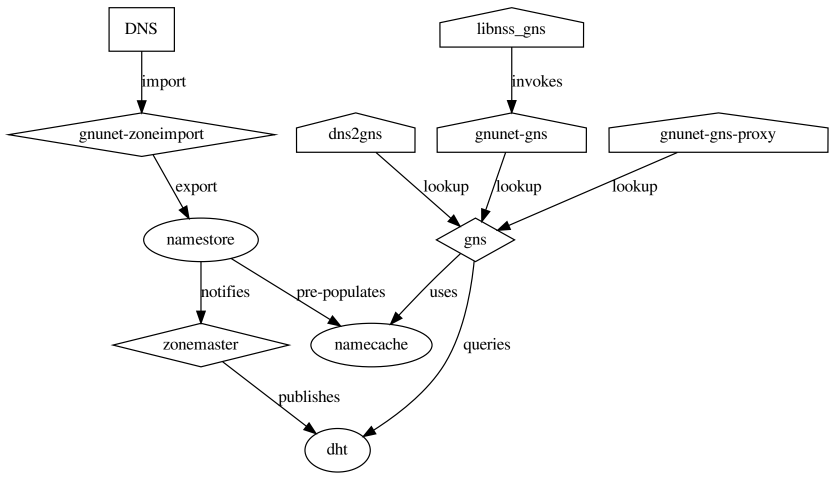


Figure 2.1: Data flow when importing legacy DNS records into GNS. Diamonds are used as the shape for stateless processes, subsystems with storage use circles, and application-facing APIs are shaped as houses. The DHT is the only component that typically involves communication over the network, all other interactions in this figure are local.

the network to ensure that all peers have the complete revocation list. This can be done using bandwidth proportional to the difference in the revocation sets known to the respective peers using Eppstein's efficient set reconciliation method. [9] In effect, the bandwidth consumption for healing network partitions or joining peers will then be almost the same as if the peers had always been part of the network.

2.1.5 Interaction with legacy DNS

It is possible to import existing DNS zones into the GNU Name System. For this, GNUnet includes the `gnunet-zoneimport` tool to monitor a DNS zone and automatically import records into GNS.

As many DNS zone operators refuse to allow AFXR requests, the tool can be given a list of DNS names to query, and then scans the entire zone, imports the resulting records (in cleartext) into the GNS database (the "namestore"), and then re-issues the queries to DNS whenever the original records are set to expire. Figure 2.1.5 illustrates the resulting data flow.

Whenever records are stored in the namestore, the "zonemaster" service is notified and performs the necessary cryptographic operations to publish the encrypted and signed records in the distributed hash table (DHT). This ultimately makes the result available to all GNS resolvers.

Note that the namestore by default also populates the namecache. This pre-population is cryptographically as expensive as the operation of the zonemaster. Thus, on systems that only serve to import a large (millions of records) DNS zone and that do not have a local GNS resolver in use, it is advisable to disable the namecache.

2.2 The re:claim identity management system

To manage digital identity attributes, we propose the decentralized, OpenID Connect compatible identity provider service re:claim [38][1]. In re:claim, attributes are stored as resource records in GNS and encrypted using ciphertext-policy attribute-based encryption (CP-ABE) and stored in a namespace.

In the following, we discuss the technical details on how ABE is used in combination with GNS and how it is leveraged to manage authorizations of requesting parties to access attributes via the OpenID Connect protocol.

2.2.1 Preliminaries

Attribute-based encryption schemes come in the two main flavours of ciphertext-policy ABE (CP-ABE) and key-policy ABE (KP-ABE). re:claim is built using CP-ABE, but both variants can be considered equally suitable.

The basic idea behind re:claim is to combine GNS and CP-ABE to realize a decentralized identity provider service. To understand the technical details of re:claim, we define the high-level functions and procedures and objects for the relevant components. Let an ABE scheme consist of the following functions:

$$\begin{aligned} \mathbf{Setup}_{\mathsf{ABE}}() &\rightarrow (msk_{\mathsf{ABE}}, pk_{\mathsf{ABE}}) \\ \mathbf{Keygen}_{\mathsf{ABE}}(msk_{\mathsf{ABE}}, A) &\rightarrow sk_{\mathsf{ABE}} \\ \mathbf{Enc}_{\mathsf{ABE}}(pk_{\mathsf{ABE}}, pt, policy) &\rightarrow ct \\ \mathbf{Dec}_{\mathsf{ABE}}(sk_{\mathsf{ABE}}, ct) &\rightarrow pt, \end{aligned} \tag{2.5}$$

where msk_{ABE} is the master secret key, pk_{ABE} the public parameters key and sk_{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_{ABE} using the function $\mathbf{Keygen}_{\mathsf{ABE}}()$. *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 $\mathbf{Enc}_{\mathsf{ABE}}()$ and $\mathbf{Dec}_{\mathsf{ABE}}()$, respectively.

We define an identity *attribute* as follows:

$$attribute = (name, value, version) \tag{2.6}$$

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

[1] `https://reclaim-identity.io`

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.

Further, let an identity provider (IdP) consist of the procedures:

$$\begin{aligned}\mathbf{Store}(ID_{\mathsf{user}}, attribute)\\ \mathbf{Delete}(ID_{\mathsf{user}}, attribute)\\ \mathbf{Authorize}(ID_{\mathsf{user}}, ID_{\mathsf{rp}}, attributes) &\rightarrow ticket\\ \mathbf{Revoke}(ticket)\\ \mathbf{Retrieve}(ID_{\mathsf{rp}}, ticket) &\rightarrow attributes\end{aligned} \tag{2.7}$$

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

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:

$$ticket = (ID_{\mathsf{user}}, ID_{\mathsf{rp}}, names, rnd) \tag{2.8}$$

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 their namespaces by executing $\mathbf{Setup}_{\mathsf{ABE}}()$.

2.2.2 Attribute storage

In re:claim, 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 their identity.

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 $\mathbf{Enc}_{\mathsf{ABE}}()$. The encrypted attribute value is published as a record under the attribute name using the name system function $\mathbf{Publish}()$. Figure 2.2 illustrates this process.

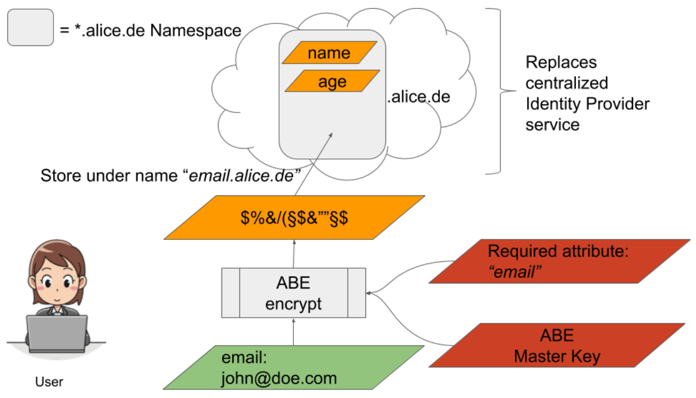

= *.alice.de Namespace
name
age
.alice.de
Replaces centralized Identity Provider service
Store under name "*email.alice.de*"
$%&/(§$&""§$
ABE encrypt
Required attribute: *"email"*
ABE Master Key
email: john@doe.com
User

Figure 2.2: In re:claim, the identity attribute is encrypted using ABE before it is stored in the users name space.

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 re:claim records from, e.g. IP addresses. In our design, all attribute resource records must have this type set.

In our re:claim implementation, to store an attribute, the following command is executed:

```
$ gnunet-reclaim -e johndoe -a email -V john@doe.com
```

2.2.3 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 $\mathbf{Keygen}_{\mathsf{ABE}}()$. 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.

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 as illustrated in figure 2.3.

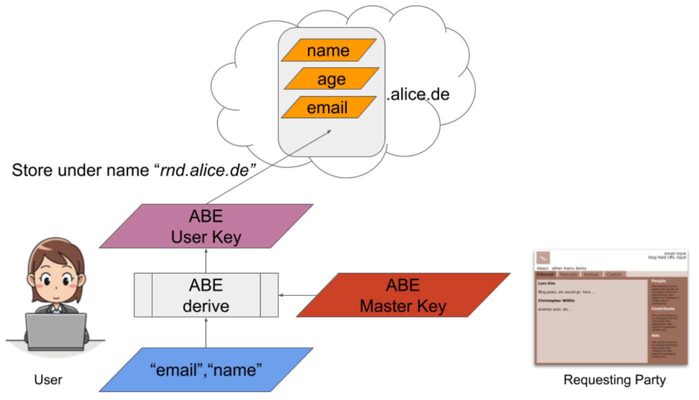


Figure 2.3: To authorize a requesting party the user creates a new ABE user key and stores it in the user name space.

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. In Figure 2.4, we illustrate how this ticket is transferred to the requesting party out of band using suitable protocols such as OpenID Connect.

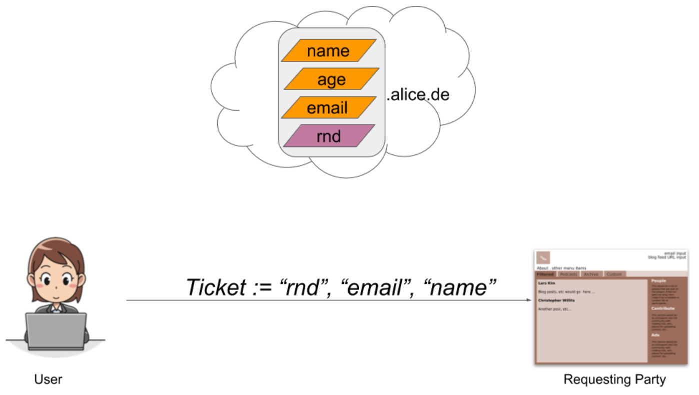


Figure 2.4: The label under which the ABE user key is stored is transferred out of band to the requesting party.

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".

In our re:claim implementation, to authorize a requesting party ID_{rp} to access the attributes a_0,a_1,...,a_n in a namespace of ID_{user}, the following command is executed:

```
$ gnunet-reclaim -e IDuser -i a1,a2,...,aN -rp IDrp
```

This command returns a string representing the ticket t that contains the label *rnd* and other metadata.

2.2.4 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 re:claim implemenation 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.

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-authorised to access all attributes they had access to before *except* for the deleted attribute.

2.2.5 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.

2.2.6 Retrieval

To retrieve an attribute a of identity ID_{user} an authorized requesting party ID_{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_{ABE} contains a set of tags that allows it to be used in the decryption of all attribute records that ID_{rp} is authorized to access. To retrieve sk_{ABE}, ID_{rp} must first resolve the key record under the name rnd in the identity namespace of ID_{user} (Figure 2.5).

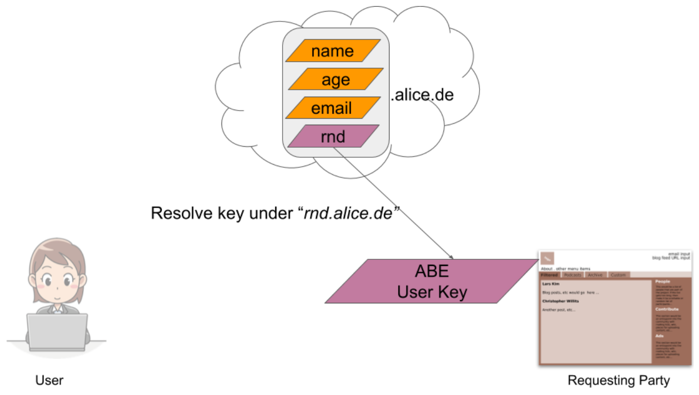


Figure 2.5: The requesting party retrieves the ABE user key from the name system.

To do so, ID_{rp} must have received the label rnd out-of-band in a ticket as discussed in the previous section. Given sk_{ABE}, the requesting party can decrypt the attribute value using the CP-ABE decryption function $\mathbf{Dec}_{\mathsf{ABE}}()$.

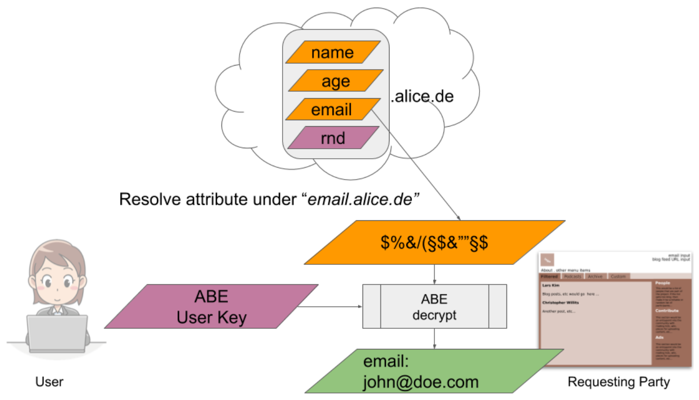


Figure 2.6: The requesting party retrieves the requested identity attributes from the name system and decrypts them using the ABE user key.

2.2.7 Revocation

We define revocation – as opposed to deletion – as the process to revoke access of a specific requesting party to user attributes in re:claim. 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 re:claim is used to prevent the decryption of an attribute record using a specific user key sk_{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[2]. 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.

2.2.8 OpenID Connect

Instead of inventing yet another authorization protocol for the Web, we built a standard compliant OpenID Connect layer on top of our re:claim implementation. We use the objects and communication patterns defined in the standard to piggyback the re:claim "Ticket" from the authorizing user to the requesting party. Further, OpenID Connect already defines how and when user consent must be provided and how identity attributes are encoded in objects and APIs.

[2]As opposed to re-bootstrapping the whole ABE scheme and issuing/publishing new keys for all RPs

In Figure 2.7, we illustrate how re:claim integrates into a standard OpenID Connect Authorization Code Flow.

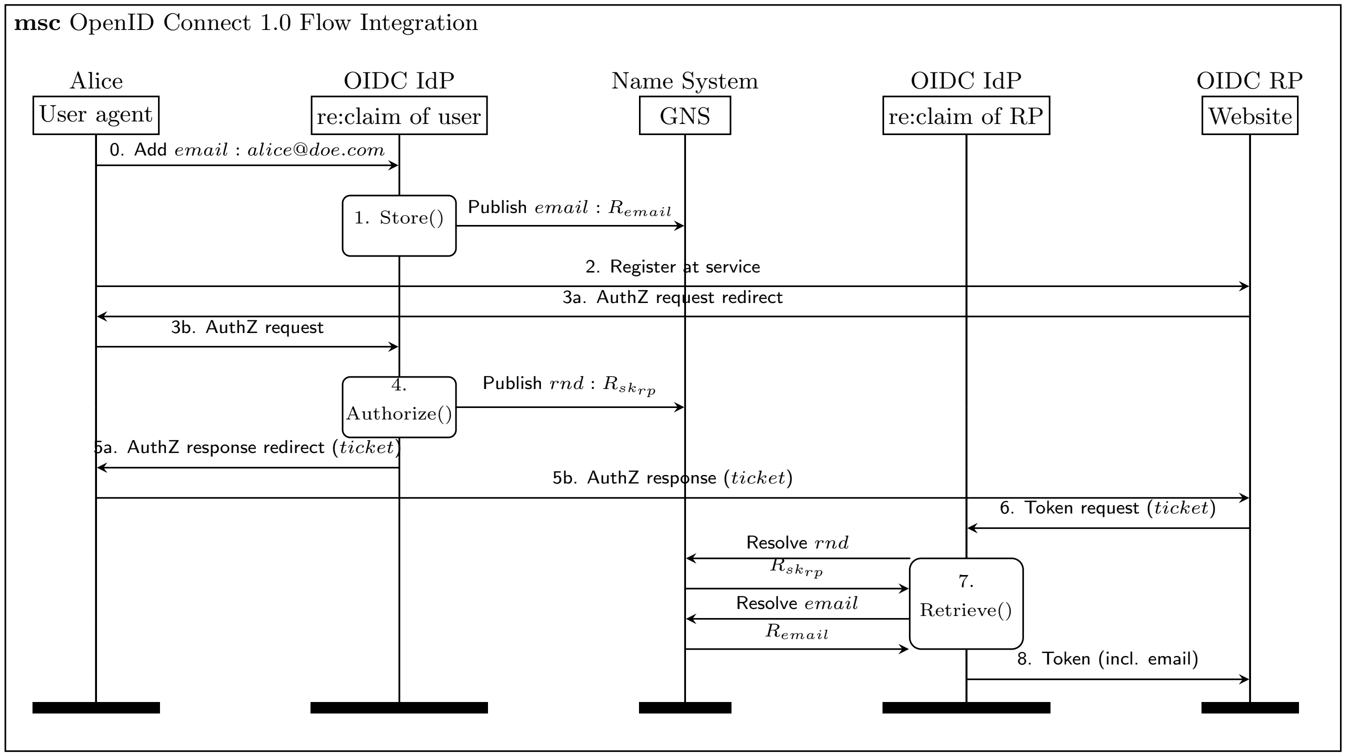


Figure 2.7: An example authorization and attribute retrieval performed through OpenID Connect (OIDC). Protocol steps 1,2,4,5 and 8 encompass the standard OIDC authorization code flow. Steps 3, 6 and 7 are interactions between the respective local re:claim and GNS components.

2.2.9 Sensor data access authorization

Instead of user identity attributes, re:claim is also suitable for storing attributes related to other, non-user entities such as IoT devices. Consider a sensor that is associated with a namespace ID_{dev}. The sensor can periodically publish sensor data (e.g. temperature) to the namespace as re:claim attributes. A webservice interested in sensor data, for example to engage in data analytics, can use re:claim to request access to the data. However, there is no standardized protocol for sharing sensor data from an IoT device like there is with OpenID Connect and user data. For this reasonn, we have designed an authorization process through the use of near field communication (NFC) that can be used by a resource owner (the owner of a device or a technician) to authorize requesting parties to access sensor data that is stored by the sensor in GNS through the use of re:claim:

Initially, a requesting party such as a website requests permission from the resource owner to access sensor data. This request is encoded in a QR code that is displayed to the owner/technician. The resource owner uses a smartphone running a dedicated re:claim app that is used to scan the QR code. The app displays the requested attributes, the requesting party public key ID_{RP} and a ticket endpoint E_{ticket} to the owner. At this point, the resource owner can see in the app what data (attributes) the requesting party is requesting. If the resource owner decides to consent to the authorization request, he initiates a NFC protocol with the IoT device. The device receives the NFC authorization request and generates a re:claim "ticket" as described in Section 2.2.3. The sensor sends the re:claim"ticket" to the ticket endpoint E_{ticket}, allowing the requesting party to retrieve the sensor data.

2.3 Group policies through the use of ABD

When managing attributes and identities in a decentralized manner, the question of attribute assertions and access control through attribute-based policies needs to be addressed. Based on work by Li et al. [24, 25] on the topic of attribute-based delegation (ABD), we created a system that allows to manage and evaluate attributes through decentralized trust and assertion chains. In [37], we show how the delegation paths of name systems can be used to represent the various attribute-based delegation types as defined by Li et al. [26, 22]:

$$A.a \leftarrow B \qquad (type1)$$

$$A.a \leftarrow B.b \qquad (type2)$$

$$A.a \leftarrow B.b.a \qquad (type3)$$

$$A.a \leftarrow \bigcap_{i=1}^{n} f_i \qquad (type4)$$

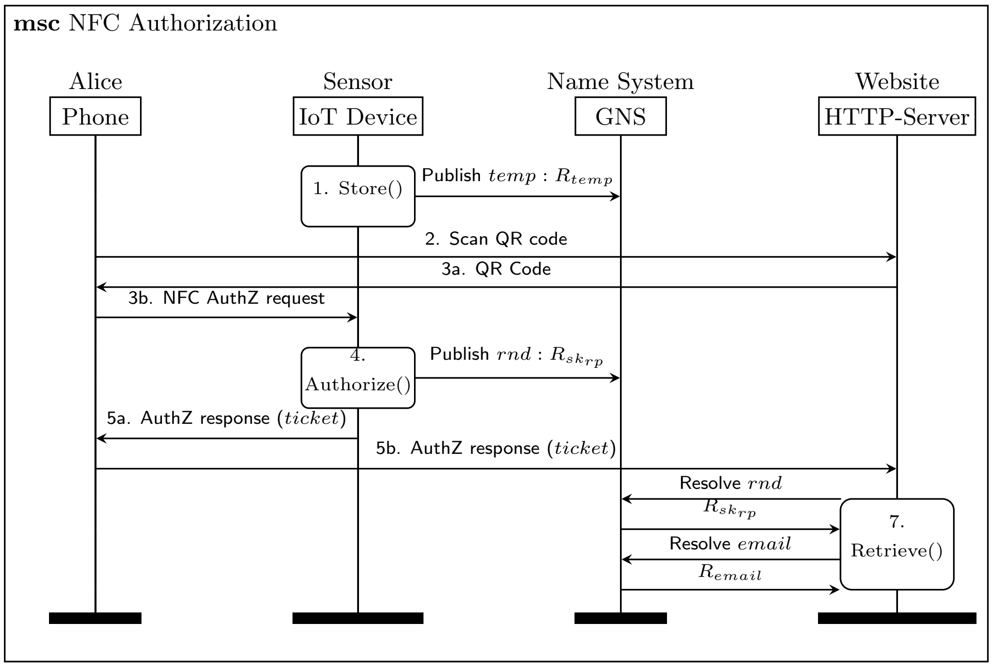


Figure 2.8: An example authorization and attribute retrieval performed through NFC. ss the standard OIDC authorization code flow.

2.3.1 Approach

We introduce a special resource record type "ATTR" for attribute delegations and modify the name system resolver logic to perform delegation chain discovery for such records. Attribute delegations such as $A.a \leftarrow e$ as introduced above are mapped into a namespace as follows: A is a namespace owned by an entity and a is the name of a record in A. The value of the record contains e, the delegation expression that defines the namespaces that a is delegated to. To support all four kinds of attribute delegations, our record contains an appropriate data structure to hold any attribute expression e in a delegation $A.a \leftarrow e$.

Specifically, we define the value of an "ATTR" record to contain one or more entries in a *delegation set*. A delegation set entry consists of a subject namespace B as well as a set of attributes and is used to represent delegation types 1-3. To model delegation types 1-3, a resource record contains a single entry in the delegation set. A type 4 delegation record contains a delegation set with n entries, each specifying the respective required delegation expression $A.a \leftarrow \bigcap_{i=1}^{n} f_i$.

While the type 4 delegation constitutes a logical "AND", a logical "OR" is not explicitly defined. However, the existence of multiple delegation records in the same namespace under the same attribute a implicitly defines this case.

Delegation and chain discovery

To confirm that an issuer delegated an attribute a to an entity B, a delegation chain must be discovered. A valid chain can be found if B holds a set of credentials $C_{A.a} \in C_B$ that allows one to build a delegation chain $D_{A.a,B}$ from an issuer namespace A and attribute a to $C_{A.a}$.

Finding a delegation chain can only be guaranteed if all attribute delegations $d \in D_{A.a,B}$ are resolvable *and* B is in possession of an appropriate set of credentials $C_{A.a}$. We define the resolver function $resolve(l, N, t)$ that is used to resolve resource records of type t under the name l in the namespace N. A call to $resolve(a, A, \text{"}ATTR\text{"})$ will return the resource records representing all issued attribute delegations $A.a \leftarrow e$ as delegation sets. Each expression e in the delegation sets is checked against the issued attributes in the set of subject credentials C_B. If we have found a valid delegation chain from the original attribute to a credential subset $C_{A.a}$, we have verified that the attribute $A.a$ is delegated to B. Our algorithm is a combination between SDSI-style rewriting and the backward resolution of a delegation graph in Li's approach for RT_0. However, as we enforce issuer-side storage by defining delegations in the issuer namespace, we do not require the more complex unified approach by Li that uses backward *and* forward search of the delegation graph.

To resolve a delegation $D_{A.a,B}$ using a name system, the namespace of the issuer and the attribute to look up must be known in advance. For an initial attribute $A.a$ the name to look up is a in the namespace A. We define $A.a$ as the root node and all resolved delegation expressions e found under $A.a$ as children of $A.a$ in the delegation graph. From then on, we follow a rewrite-resolve-check pattern until we can match a credential against a delegation subject.

If a resource record containing a delegation set with a single entry is resolved, the expression e is of type 1-3. Otherwise, it is of type 4. In both cases, we use SDSI-style rewriting rules [5]:

For a type 1-3 delegation $e := B.b_1.b_2.b_3...b_n$ we perform a lookup query using only the leftmost attribute b_1 and we *rewrite* the resulting expressions from a call to $resolve(b_1, B, \text{"}ATTR\text{"})$ by appending b_2 through b_n. This leads to a reduction of the original delegation expression if the query returns a type 1 delegation or an enlargement for a type 3 delegation. In the case of a type 2 delegation, the expression complexity does not change. For a type 4 delegation $e := \bigcap_{i=1}^{n} f_i$ we process each f_i like type 1-3. Rewriting a type 4 delegation set is simply a matter of rewriting every delegation set entry individually.

The rewritten delegations are added as children of e in the delegation graph and checked against the subject credentials. The process continues iteratively until a matching set of credentials is found that allows us to backtrack the delegation graph to $A.a$. When we backtrack the delegation graph and encounter a node that holds a type 1-3 delegation, it is verified that the delegated attribute has a path to a set of subject credentials and we can continue backtracking. If we encounter a node representing a type 4 delegation, we have to make sure that every f_i in the node is satisfied by a set of credentials before we can continue.

2.3.2 Authorization through policy evaluation

We define a resource r to be protected by a policy P that specifies a set of attributes. A verifier V can perform attribute-based authorization of a subject that requests access to r. To do so, the verifier initially retrieves P by querying a policy storage. We define the attribute issuer for all attributes $x \in P$ to be the verifier V. The verifier is initially unaware as to which credentials the subject must provide to satisfy P. At the same time, the subject is initially not aware of what attributes are required by P to access r. Our simple authorization protocol with delegation chain discovery is illustrated in Figure 2.9.

(1) The subject S tries to access the resource r.

(2) To retrieve the access policy P for the resource r, V uses a function *getPolicy*. Afterwards, V sends a response containing the policy P.

(3) S uses a function *collect* that finds subsets $C_{V.x}$ of the subjects credentials C_S that satisfy the attributes $x \in P$. S sends the set $C_P := \bigcup_{x \in P} C_{V.x}$ to the verifier.

(4) We define a function $verify$ that uses delegation chain discovery to verify that a delegation chain D exists for a set of credentials to an attribute. V uses this function to confirm that a delegation chain $D_{V.x}$ can be found for all $x \in P$ using C_P. Access is granted only if all delegation chains can be found.

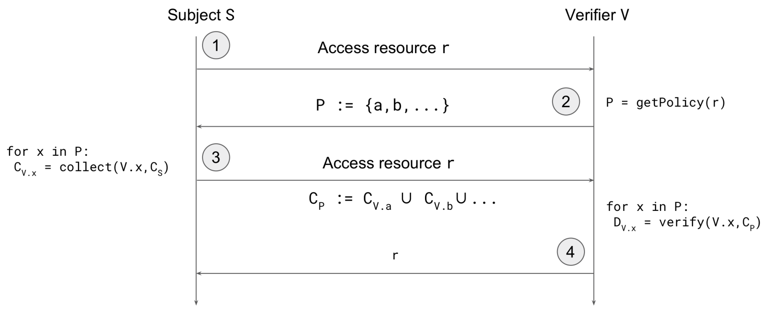

Subject S
Verifier V
1
Access resource r
P := {a,b,...}
2
P = getPolicy(r)
for x in P: C_{V.x} = collect(V.x,C_S)
3
Access resource r
C_P := C_{V.a} ∪ C_{V.b} ∪ ...
for x in P: D_{V.x} = verify(V.x,C_P)
4
r

Figure 2.9: Authorization with delegation chain discovery.

2.3.3 Revocation

Revocation of a delegation is achieved by having the respective issuer revoke the attribute name that points to it in the name system. A name in the name system can only be resolved if it exists and is not expired. Attribute delegations must be treated in the same fashion. Whenever an issuer wants to remove an attribute delegation, he must delete the respective records from his namespace. It is also important that attribute delegations must have a set expiration date. Distributed name systems tend to cache records in the network until they expire. Even if a record is deleted by the namespace owner, it might still linger until caches are purged or the record has expired.

Chapter 3

Usability

In this chapter we present the results of four usability studies involving the use of the GNU Name System (GNS) for various scenarios in different application domains.

3.1 Network service authentication

To demonstrate usability of GNS for network service authentication, we setup GNS as a replacement for DNS. As GNS provides authenticated replies, using GNS instead of DNS allows for the secure and authenticated distribution of IP addresses as well as associated X.509 public keys or certificates via TLSA records [17], thereby obsoleting the existing certificate authorities (CAs) and replacing them with GNS zones that are limited to certify services within their respective domain. In addition to addressing the lack of end-to-end security in DNSSEC, the use of GNS instead of DNS also provides significantly improved query privacy over DNSSEC or even DNS-over-TLS.

The ideal outcome of the experiment would be that users cannot really tell the two setups apart, as today DNS is definitively usable and thus an indistinguishable experience is the best we can hope for — while of course under the hood providing better security and privacy. *Objectively* the setup with the GNS resolution has to be slightly slower, not so much because of the name resolution itself, but because `gnunet-gns-proxy` implements an HTTPS man-in-the-middle proxy that verifies the TLS certificates against the information in GNS and then creates a valid certificate on the fly for the browser using its own CA. In an ideal deployment, GNS name resolution would be integrated dirctly with the browser and thus not incur this significant overhead.

The usability study used two notebooks with identical hardware both running a minimal installation of Ubuntu with a Firefox browser. System **A** is simply this trivial setup. On system **B** we additionally configured a GNUnet peer. First of all, the peer was configured to only connect to the testbed DHT we deployed in EC2 using the “xt” transport plugin and `http://h2020xt1.`

`gnunet.org:1080` and `http://h2020xt2.gnunet.org:1080` for the "hostlist" option to bootstrap. GNUnet was also allowed to use "unlimited" bandwidth. Finally, the AUTOSTART option for `gnunet-gns-proxy` must be set. GNS is by default already setup to resolve `.fr` via the DHT. In EC2, two "t2.medium" hosts were constantly iterating over the 5 million records of ".fr" to keep the DHT content current. Firefox was configured to connect to the network via the `gnunet-gns-proxy` and to resolve DNS queries over this SOCKS5 proxy. The proxy's certificate was imported into the browser's CA root set.

Users were then asked to use both machines and to access various sites including in particular domains in `.fr`.[1] Afterwards, they were given a survey asking them to compare their browsing experience as well as some background about themselves. Users were **not** informed which system is using DNS or GNS, and also not told that the difference between the system was in the way they did name resolution. Table 3.1 summarizes the results on the comparison.

Table 3.1: Survey responses comparing DNS (A) and GNS (B) when browsing the Internet. Scores ranking from "System (A)" via 3 ("neither") to 5 for "System (B)".

Question	1	2	3	4	5	AVG ± STDDEV
Which system was faster?	8	0	10	3	8	3.03 ± 1.52
Which system was more fun to use?	7	1	16	4	3	2.83 ± 1.21
Which system do you believe provided better security?	4	2	20	3	2	2.86 ± 0.98
Which system do you believe provided better privacy?	3	1	22	3	2	3 ± 0.89
Which system performed more reliably?	9	0	15	2	4	2.73 ± 1.34

We note that the participants had no information about the actual implications of using DNS vs. GNS on security and privacy based on the instructions. The goal of the usability study was after all to detect *subjective* differences. Table 3.1 summarizes the results from the survey. The results show that users have a hard time telling the difference between DNS and GNS. As DNS is used by billions of Internet users every day, this implies that GNS is clearly usable for name resolution without any further training of end-users.

3.2 IoT sensor data access authorization

We demonstrate the feasability of GNS as a suitable authentication and authorization system for the Internet of Things (IoT) through the use of re:claim. We

[1]Detailed instructions and demographic data on the participants is Appendix A.2.

have conducted a usability experiment that involves a user acting as a technician, a webservice that is capable of processing and analizing sensor data and a device that has various sensors attached to it.

The device is generating a constant stream of sensor data and is storing it in re:claim as attribute data. In addition to the sensors, the device also features an NFC interface that allows a technician to trigger a re:claim authorization procedure. The technician is equipped with a mobile phone which is running a re:claim mobile application.

In our study, the technician is tasked to authorize the webservice to access the sensor data that is generated by the device and stored in GNS.

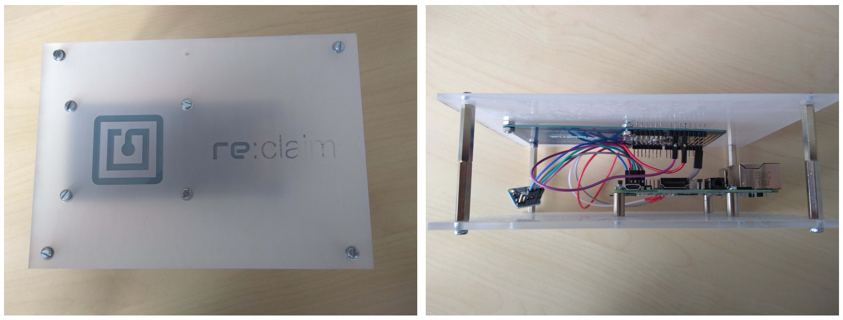
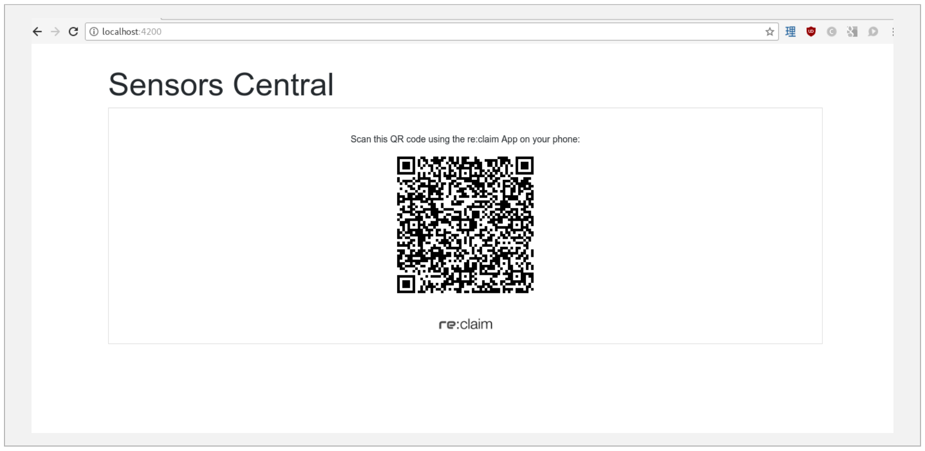


Figure 3.1: Our IoT prototype. The prototype is built using a Raspberry Pi 3B. Attached to the Pi are a NFC shield used for the NFC authorization flow and a BME280 Sensor used to collect data.

To do so, the technician first accesses the website of the webservice.



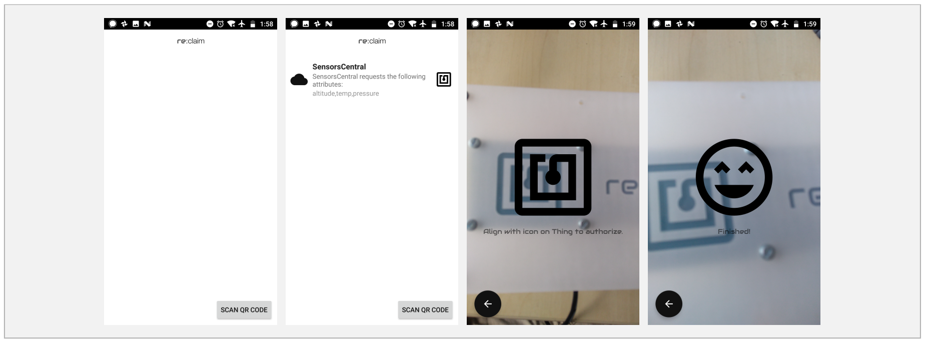
Sensors Central

Scan this QR code using the re:claim App on your phone:

Figure 3.2: The website QR code is scanned using the reclaim app.

The website will display a QR code and tell the technician to scan the

QR code using the re:claim app. Using the re:claim app, the technician must scan the QR code. After having scanned the QR code, the app displays which attributes the webservice is requesting. In our setup, the attributes are temperature, pressure as well as altitude. We assume that the technician consents to this authorization request by bringing the phone in close proximity to the device.



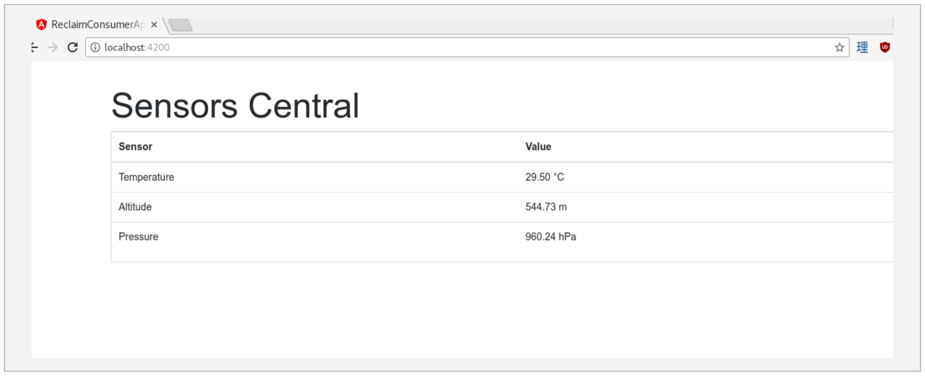
SensorsCentral requests the following attributes:
altitude,temp,pressure

Align with icon on Thing to authorize.

Finished!

Figure 3.3: The scanned entry is selected to initiate the NFC authorization flow with the device.

After a final confirmation dialog, the phone transfers the information required by the device to execute a re:claim authorization procedure. Finally, the app receives a re:claim ticket from the device and transfers it to the webservice. The webservice in turn processes the ticket and retrieves the attributes from re:claim and GNS. The sensor data is then displayed to the technician on the website without further interaction to indicate the successful authorization.



Sensors Central

Sensor	Value
Temperature	29.50 °C
Altitude	544.73 m
Pressure	960.24 hPa

Figure 3.4: After a successful authorization, the website is triggered to retrieve and display the sensor data.

In conclusion to the test our users/technicians were asked to evaluate their experience using a standardized questionnaire including a system usability scale

(SUS). The wide-spread nature of the SUS scope allows us to use the resulting SUS score to judge the usability compared to existing, similar systems. Figure 3.5 shows a box plot of our collected SUS scores. The median SUS score is above 80 points indicating that the usability of our system compared to other systems [1] is well above average. In fact, it is above 80,3% which groups it in the top 10% of all scores collected in [1].

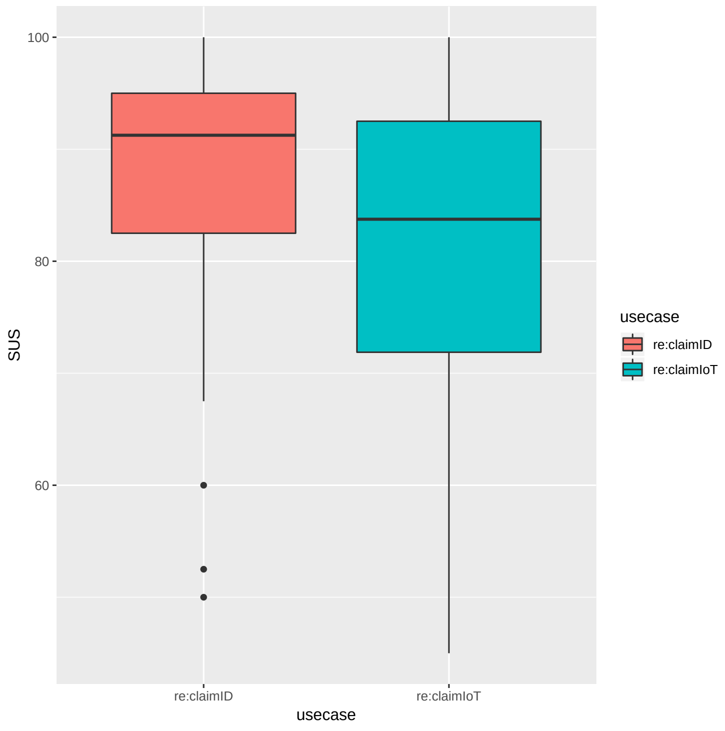


Figure 3.5: SUS scores for the re:claim use cases. The re:claimID bar reflect the results from the user authorization study. The re:claimIoT bar reflect the results from the sensor data access authorization study.

3.3 End-user identity access authorization

We propose to use or re:claim system as a drop-in replacement for current, so-called "social login" systems built using OpenID Connect. The most widely spread implementations and providers for social logins are Google and Facebook. Especially smaller websites and services rely on the identity provider services of those offerings to outsource identity management. This includes necessities such as authentication and personal data storage. Authentication is a critical security property difficult to get right and personal data storage can prove to be a liability as loss of this data can have severe legal consequences.

Our aim is to provide a social login component that behaves similarly to existing solutions. This is why we build an OpenID Connect layer on top of re:claim and we demonstrate that our implementation satisfies usability requirements. The respective components in re:claim are compliant with the OpenID Connect standard specification and can thus be integrated by websites and services following the respective integration guides for the protocol.

The user experience, however, is not defined in the specification and thus we designed a user experience that presents the user with a website that requests access to personal information from the user by initiating an OpenID Connect flow. We prepare a laptop for our user that is running the webpage as well as a local re:claim instance. The user is initially presented with a webpage in the browser that allows him to login using either Google, Facebook or re:claim. Using an guidance sheet, the user is instructed to login using the re:claim option:

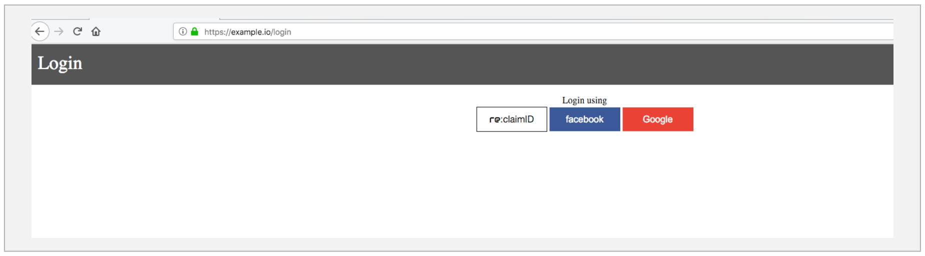
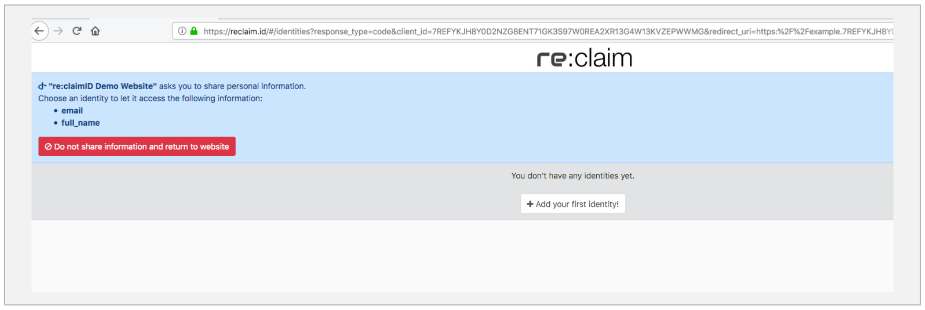


Figure 3.6: The demo website asking the user to login.

After the user clicks on the login button, the are redirected to the re:claim OpenID Connect identity provider. At this point, the personal data requested by the website is displayed. In this case, the website requests the user's full name and email address.

Initially, the user does not have any identities in their re:claim system so the first step is to create a new identity. The user interface guides the user through this process.


https://reclaim.id/#/identities?response_type=code&client_id=7REFYKJH8Y0D2NZG8ENT71GK3S97W0REA2XR13G4W13KVZEPWWMG&redirect_uri=https:%2F%2Fexample.7REFYKJH8Y

re:claim

"re:claimID Demo Website" asks you to share personal information.
Choose an identity to let it access the following information:

- email
- full_name

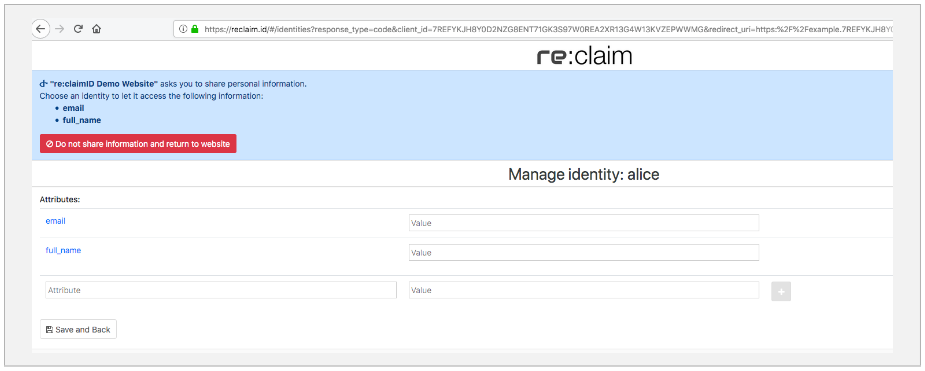
Do not share information and return to website

You don't have any identities yet.

Add your first identity!

Figure 3.7: Initially, the user is asked to add their first identity.

After the user creates an identity, they have the option to provision it with personal data. The attributes requested by the website are preallocated to save the user from manually entering them.



https://reclaim.id/#/identities?response_type=code&client_id=7REFYKJH8Y0D2NZG8ENT71GK3S97W0REA2XR13G4W13KVZEPWWMG&redirect_uri=https:%2F%2Fexample.7REFYKJH8Y

re:claim

"re:claimID Demo Website" asks you to share personal information.
Choose an identity to let it access the following information:

- **email**
- **full_name**

Do not share information and return to website

Manage identity: alice

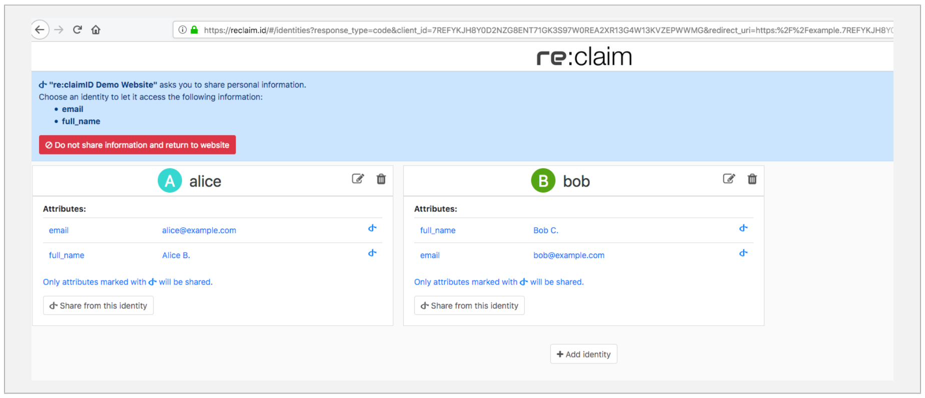
Attributes:

email	Value
full_name	Value
Attribute	Value

Save and Back

Figure 3.8: The user is asked to add the requested attributes after adding a new identity.

The user has the option to create any number of identities at this point and edit them to their needs. Once the user is finished, they can choose an identity they wish to share personal data from.



https://reclaim.id/#/identities?response_type=code&client_id=7REFYKJH8Y0D2NZG8ENT71GK3S97W0REA2XR13G4W13KVZEPWWMG&redirect_uri=https:%2F%2Fexample.7REFYKJH8Y

re:claim

"re:claimID Demo Website" asks you to share personal information.
Choose an identity to let it access the following information:

- **email**
- **full_name**

Do not share information and return to website

A alice

Attributes:

email	alice@example.com
full_name	Alice B.

Only attributes marked with will be shared.

Share from this identity

B bob

Attributes:

full_name	Bob C.
email	bob@example.com

Only attributes marked with will be shared.

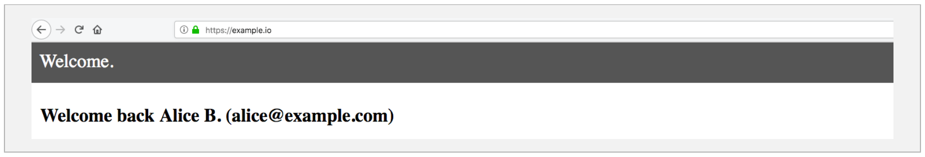
Share from this identity

Add identity

Figure 3.9: The main re:claim user interface listing all available user identities.

The user is the redirected to the requesting website according to the OpenID Connecto protocol.

The website is provided with the OpenID Connect specific “authorization code” that represents the user consent to access their personal data. The authorization is further used by the website to show that it is authorized to access this data to their local re:claim OpenID Connect instance. By exchanging the authorization code – which is used to a piggyback re:claim ticket – the website



Welcome.

Welcome back Alice B. (alice@example.com)

Figure 3.10: After selecting an identity to login, the user is redirected to the requesting party and is logged in after the website has verified the re:claim ticket.

can retrieve the personal data that the user has chosen to share. Finally, the user will be presented by a welcome webpage displaying the successful login and the personal data that the user now has shared with the website.

After the test, the users are asked to fill in a survey including a standardized system usability scale (SUS) and further questions to evaluate key usability indicators. The wide-spread nature of the SUS scope allows us to use the resulting SUS score to judge the usability compared to existing, similar systems. Figure 3.5 shows a box plot of our collected SUS scores. The median SUS score is above 90 points indicating that the usability of our system compared to other systems [1] is well above average. In fact, it is well above 80,3% which groups it in the top 10% of all scores collected in [1].

3.4 Accident insurance claims in Switzerland

In e-health, high-quality healthcare requires efficient and reliable access to personal data. The access of patients' data is essential to ensure the smooth functioning of many clinical and administrative processes. The data involved are to a large extent sensitive, since they contain personal data of patients including medical data. Key issues in e-health are the distributed ownership and location of the patients' data. The data are created and maintained by different actors and then distributed via the patient to other health professionals. Today's solutions include either a central storage of the patients' data, often named electronic patient record (EHR), or the involvement of patients, who must bridge the gap between independent systems and pass the data itself (by phone, email or letter). A new method of secure, but simple access to these sensible data for health professionals under the condition that they are involved in the treatment of a patient is needed.

Initial situation

The accident insurance claims are regulated by the Federal Law on Accident Insurance (Bundesgesetz über die Unfallversicherung, UVG)[8]. Accident insurance is compulsory for all employees and unemployed persons. All others, like self-employed persons, can have a private accident insurance.

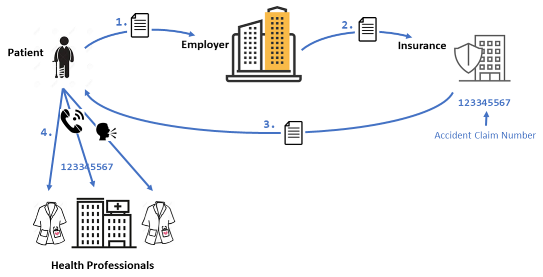


Figure 3.11: Initial situation: the patient transmits the accident claim number to the health professionals.

- *Case 1: Compulsory accident insurance*
  Following an accident an employee must notify their employer without delay. The employer must in turn notify the insurance company immediately. The employee subsequently receives a form that must be completed truthfully by the employee or by the treating physician, and then sent without delay to the insurance company.

- *Case 2: Private accident insurance*
  The insurance company is informed directly by the casualty.

In both cases the insurances company will issue a claim number for the casualty; this can take several days. The claim number is needed to file the costs for medical treatments with the insurance company. This a manual process driven by the casualty: they must transmit actively by phone or in person the received claim number to all medical institutions (doctors, hospitals, therapists, laboratories, etc.) and pharmacists, see Fig. 3.4. If the casualty fails in transmitting the claim numbers to all involved parties, the incidental costs will be claimed to their private health insurances.

All health professionals and medical institutions need the accident claim number for accounting. If the accident claim number of a patients is not correctly entered in the accounting systems in time, the invoices are sent to the health insurance and must be reposted afterwards.

Target situation

As above described, the process of transmitting the accident claim number is today a purely manual process. The GNS allows for the first time to design an electronic process, shown in Fig. 3.4. The patient, formerly main responsible in the process, has now only to initialize the process by inventing a *keyword* and giving it to their employer or private accident insurance and to all health professionals during the visit. This enables on one side the insurance to issue

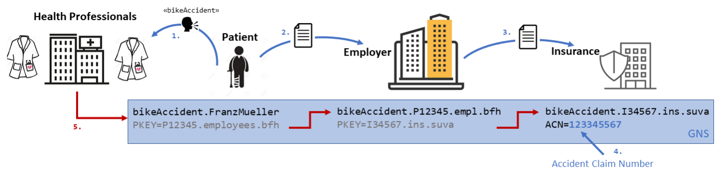


Figure 3.12: Target process: the health professionals get the accident claim number from GNS.

an accident claim number for the patient and on this other side it allows the automatic retrieval from the health professional side.

1. A person, later called patient, has an accident. This patient receives an emergency treatment. They authorize the emergency physician to access their EHR (in the EHR, the physician finds the public key, e.g. `P023245` for the patients' GNS zone). The patient has not yet contacted their employer or accident insurance, therefore the accident claim number is not yet available. The patients provides instead a keyword, e.g. "BikeAccident" to the physician, that allows the physician's system to query for the accident claim number later.

2. The patient informs their employer as requested by law. In the form they have to fill out they note the keyword they chose for the accident. The EHR zone of the patient (in our example `P023245`) includes a delegation to the zone of their employer (e.g. *PKEY=E123456* under `employer.P023245`)

3. The employer transmits the received information about the accident, including the keyword, to its accident insurance company. The employer also has an entry in its zone related to the its accident insurance (e.g. *PKEY=I34567* under `accident-insurance.E123456`).

4. The accident insurance company decides about the case and issues an accident claim number for the patient, when the case was accepted. The number is stored in the GNS zone of the accident insurance company under a label identifying the patient and the keyword (e.g. *BikeAccident-P023245.I34567*).

5. The emergency physician and all health professionals involved in treatments related to the accident can now automatically retrieve the accident claim number from the GNS (e.g. by resolving

   ```
   BikeAccident-P023245.accident-insurance-employer.P023245
   ```

) and claim their costs to the right insurance company. Note that the clinical staff does not need to be able to compose the above name, as with appropriate standardization the path can be automatically generated by the software from just the keyword and the EHR. In any case, the patient and employer are not involved anymore.

Usability test

As the patient has only to initialize the process and is later not any more involved, for the usability study, we focused on health professionals and the process of getting the accident claim number into the clinical information system (CIS). The prototype showed a rudimentary CIS allowing to create a new case for a patient that needs treatment after an accident.[2] This special set-up requires a certain process knowledge of the test participants, therefor we invited mainly persons with medical background or with knowledge about the Swiss health system.

The survey was filled out by 44 persons. According to the self reported professional context 25 persons belong to the main target group of health professionals and medical informatics specialists. Received SUS scores for all users and for the main target group are shown in Table 3.2.

The results are quite promising for further development of the new process. The majority of users felt confident and liked the new automatic process. The prototypical implementation of our CIS led to some cutbacks in the evaluation. For some it was difficult to separate the presented user interface from the process to test. Another point that influenced the evaluation was the point that not all invited users were previously familiar with the existing process for accident claims.

During the tests, we got valuable feedback how to enhance the process and how to improve usability further. The first point was the unnoticed appearance of the accident claim number issued by the insurance. Some users would like to have an explicit notification (pop-up, email, SMS) to be better informed. Other users wished to have more information from the insurance company, e.g. current address of the patient, more information about the accident. Pursuing such ideas will be limited by applicable data protection and privacy laws in the medical sector.

[2]Detailed instructions and demographic data on the participants are in Appendix A.5

Question	SUS Score	
	All users	Main target group
(As health professional) I think that I would like to use this system frequently.	4.16	4.00
I found the system (the process of getting the accident claim number) unnecessarily complex.	1.75	1.92
I thought (the part of) the system (where the accident claim number is retrieved) was easy to use.	4.32	4.4
I think that I would need the support of a technical person to be able to use this system.	1.68	1.76
I found the retrieving of the accident claim number in this system was well integrated.	4.41	4.28
I thought there was too much inconsistency in this system.	1.70	1.84
I would imagine that most people would learn to use the new process very quickly.	4.57	4.52
I found the new process for the accident claim number very cumbersome to use.	1.89	1.96
I felt very confident using the system.	4.14	4.04
I needed to learn a lot of things before I could get going with this system.	1.70	1.68
Sum	30.33	30.4
Total SUS	75.82	76.00
Rating	good	good

Table 3.2: SUS score for different user groups

Chapter 4

Reliability

Aside from a few cryptographic attack vectors, the reliability of the GNU Name System (GNS) is largely dependent on the availability and performance of the underlying distributed hash table (DHT). While the design of GNS does not mandate a particular DHT, our implementation uses the Byzantine fault-tolerant R^5N DHT [12] of GNUnet[1]. Thus, in this section we are focusing on reporting results from a set of controlled reliability experiments we performed on the R^5N DHT using the Amazone Elastic Cloud (EC2).

In terms of impact of DHT failures on the GNS authentication solution, if a DHT lookup fails to return a result in a timely fashion, GNS fails to obtain the required credentials which will result in a false rejection. As our measurements show, the exact rate will depend on the effect of caching and the characteristics of the DHT infrastructure. We note that it is never possible that lookup failures will grant users access that should not have access. Thus, the false acceptance rate (FAR) of GNS is always zero assuming users have sufficiently protected their key material.[2]

4.1 DHT fundamentals

As mentioned, all public GNS data encrypted and published via a DHT. A DHT is typically created using a network of peers cooperating to maintain a shared data store. Anyone can store a `PUT(key, value)` in the network and retrieve the `value` based on the key `GET(key)`. There are different possible implementations of a DHT for GNS. Our implementation uses the "Random Recursive Routing for Restricted-Route Networks" (R^5N) implementation [12].

In R^5N, the PUT operation stores data on a randomized subset of the participants to the network. PUTs are repeated at a certain frequency to refresh the data and keep it available despite churn. GET is also non-deterministic and

[1] `https://gnunet.org/`

[2] We are not aware of a study providing reliable data on key missmanagement by users that would be meaningful for the broad range of use-cases we consider.

searches the information in some nodes. If the information is not found, the node is expected to repeat the GET to access the information. Both GET and PUT operations are randomized and thus may take somewhat different paths each time. Consequently, adversaries have difficulties to block all the operations on a particular `(key, value)` pair.

GNS `value`s are accepted for storage by the network if they are **cryptographically signed** with the private key corresponding to the `key`. This prevents anyone other than the zone's owner from publishing `value`s under that `key`. We note that since the signing key is *derived* from the zone's key pair, it is not possible to deduce the zone from the `key` or `value`. In GNS, values cannot exceed 64kb in size, thereby limiting attacks involving large transmissions of invalid values.

4.2 DHT reliability measurements in EC2

As described in Section 9.1.4, we configured a DHT replication factor of 10 for our EC2 setup. Thus, while peers dropping out of the network will always impact overall performance, the content would only be expected to become unavailable if all 10 replicas become unreachalbe.

We experimentally assessed the DHT performance as follows. After importing the five million records of ".fr" into GNS, we setup three clients that would query the DHT provided by EC2. As GNS caches results, we partitioned the ".fr" names into five groups of one millon entries each: the first group is queried by each client, groups two to four only by one of the clients, and group five by none of the clients.

We then measured query performance for queries in each group under three scenarios: **(1)** *Normal business*, where the EC2 DHT is fully operational. **(2)** *Massive outage*, where we reduced the EC2 DHT by one third, removing all nodes in the nearest zone (Frankfurt), leaving only Ohio and Seoul. Measurements are performed from Biel (Switzerland), so turning off Frankfurt also disables all of the nearby caches and as a result significantly increases network latency. **(3)** *Churn*, where two of the DHT peers were turned off for two hours (and their caches purged), in a round-robin fashion iterating over all 24 peers in 24h. This results both in an average capacity loss of 10% as well as quite catastrophic data loss as the longest-running peers with the best-performing data sets of the DHT are lost. We also note that the two systems performing the data import each take 48h to import the entire data set from DNS.

We used a DHT with 24 "t2.micro" peers (5 replicas, 5 million records, 1 million records per peer). All three clients were configured to issue a fresh query every 250 ms, thereby simulating the load of 1.000 users.[3] When the DHT did not respond after 120 seconds, we considered a query as failed. For the successful queries, we report the median query latency.

Table 4.1 summarizes the results.

[3]The DHT could have handled a higher load, but then the client's (non-SSD) harddrives could have become the bottleneck as the clients check the local cache before asking the network.

	(1) normal	(2) Frankfurt down	(3) churn
Group 1 (shared)	605 ms / 97%	2730 ms / 84%	14 s / 73%
Group 2	772 ms / 86%	2946 ms / 84%	20 s / 71%
Group 3	771 ms / 86%	2942 ms / 84%	20 s / 71%
Group 4	760 ms / 86%	2920 ms / 84%	19 s / 71%

Table 4.1: Performance and reliability measurements; we provide the reliability as a percentage, and the median latency.

We note that for scenario (1) we are operating the DHT deliberately at capacity to arrive at low cost estimates. Higher reliability and lower latencies could be achieved by providing additional capacity and servers closer to the clients. Scenario (2) demonstrates what happens if a significant amount of capacity is lost. We expected slightly lower reliability numbers for Group 2–4, and cannot quite explain the lack of a performance drop there. Scenario (3) highlights the importance of populated caches. It demonstrates that long-running systems with caches are critical for latency and reliability; thus, stable peers with large caches can substantially improve performance, assuming records have commensurate lifetimes.

Lagging behind DNS

The import of DNS records into GNS in our implementation comes with some lag, as we do not have direct access to the raw data of ".fr" and thus must rely on periodically crawling the zone. However, we did collect more comprehensive data on ".se" (as this zone does support AFXR). [34] Out of approximately 10 million records, ".se" has 20,000 glue records, 4.5 million "NS" records, and more than 5.7 million records relating to DNSSEC (DS, RRSIC and NSEC). Looking at the churn rate, changes to RRSIG records completely dominate with 912k changes per day *on average* in the first 6 weeks of May and June 2018. In constrast, on a typical day, less than a dozen glue records and the combined number of changes to NS, DS or NSEC records is less than 10,000.

Overall, over 95% of all changes to the ".se" zone apply only to DNSSEC, which is an area of DNS that is completely replaced by conversion to GNS, eliminating the causes for the high churn rate by design. The remaining *relevant* churn rate is about 4000 records per day, or 0.04%. Thus, with a typical lag of about two days, using GNS instead of DNS today would result in a failure rate of about 0.1% from GNS being out of date. We note that this problem would no longer apply once DNS registrars directly export their data into GNS, instead of us crawling DNS to obtain it.

4.3 Revocation reliability

GNS uses a flooding mechanism to spread revocation notices. Basically, when a peer receives a fresh revocation, it immediately sends it to all of its neighbors.

When peers connect, they use difference digests [9]. to efficiently reconcile revocation lists.

As a result, revocations are communicated at basically lightspeed (plus $\approx$ 100k cycles for signature verification at each hop) on the shortest available network path from the source of the revocation to all systems.

Thus, revocation reliability is basically 100%. The system is furthermore highly efficient for checking if a key has been revoked, as this is a simple *local* lookup. The system is also extremely efficient in the absence of revocations, as in that case it imposes virtually zero overhead.

However, *actual* revocations are costly, as all peers have to be informed, requiring $O(|E|)$ bandwidth for $|E|$ edges in the network. GNS limits the potential for abuse of the revocation mechanism by requiring the revocation notice to include a proof-of-work token.

4.4 Completeness and robustness

DHTs by design deal with peers joining and leaving the network, including without warning. [31] Combined with data replication, DHTs can be very robust against partial failures. GNUnet's R^5N DHT [12] is furthermore designed to be tolerant against Byzantine failures. As a result, performance merely degrades if the resources available to malicious participants increases. [11]

Most importantly, the use of a DHT encourages the creation of a desirable game theoretic equilibrium as it provides a Commons. Nodes participating in the GNS DHT obliviously provide a service to *all users*. As a result, *all users* suffer if there is an outage. Thus, assuming GNS is widely used, it would only be a target for virtual terrorists, but not for larger rational state actors as those would also undermine name resolution of their allies and themselves.

We note that solutions that are not adequately decentralized and where infrastructure can be attributed to particular user groups does not share this game theoretic drive towards cyberpeace. As long as infrastructure can be attributed to serve a particular group, there will be incentives for other groups to attack it. What really minimizes the incentive to attack is our unique combination of a fully decentralized database with GNS, which keeps the infrastructure itself in the dark about which users it is serving.

4.5 Maintenance

The DHT implementation continuously exports performance metrics during operation, allowing administrators to monitor network performance. This can be used to provision additional network resources under conditions of high load. As the DHT operates as a peer-to-peer network, managing nodes that join or leave the network (also called churn) is part of the core protocol.

GNUnet uses a modern software development process including static analysis, regression testing, performance regression testing and continuous integration

Table 4.2: Reliability comparison.

Technology	Decentralized	Self-organizing	Byzantine fault-tolerant	Commons for cyberpeace
DNSSEC	✓	✗	✗	✗
X.509 CAs	✓	✗	✗	✗
verimi.de	✗	✗	✗	✗
id4me.org	✓	✗	✗	✗
keyp.io	✓	✓	✓	✓
miracl.com	✓	✗	✓	✗
DecentID[15]	✓	✓	✓	✓
fidoalliance.org	✓	✗	✗	✗
GNS+re:claim	✓	✓	✓	✓

to minimize the possibility that software updates break critical features.

A significant disadvantage of using a peer-to-peer network is that incompatible protocol changes theoretically require scheduled maintenance to avoid fragmenting the network by protocol version. If necessary, a possible work-around would be to operate two DHTs in parallel for a while. However, in general we expect protocol versioning to allow us to migrate to new protocol versions without a flag day (NCP/TCP) or parallel production networks (IPv4/IPv6).

4.6 Summary

Table 4.2 compares the reliability properties of GNS with those of major competitors. We consider whether the infrastructure is fully decentralized (no single points of failure), self-organizing (no social engineering attacks), Byzantine fault-tolerante (some operators may be malicious), creates a Commons for cyberpeace (does not allow attackers to exclude themselves from being harmed).

Chapter 5

Security

In this chapter, we discuss the security properties of GNS. We initially define the adversary model under which GNS is designed to be used and elaborate on the used cryptographic primitives in the GNS implementation. Based on the adversary model we then give an overview of possible attack vectors and how they are mitigated using the security properties of the design and implementation of GNS. Finally, we briefly compare how similar technologies relate to GNS with respect to their security properties.

5.1 Adversary model

The GNS adversary model includes attackers with the resources and powers of a nation state willing and trying to limit the access of users to information without causing excessive damage to its own economy. The goal of the adversary is to force name resolution to change in the respective name system, by either making the resolution fail or by changing the value to which an existing name [1] maps.

We allow the adversary to participate in any role in the name system. This implicitly excludes the existance of a global trusted third party. In addition, the adversary is allowed to assume multiple identities. We impose no bound on the fraction of collaborating malicious participants, and we assume that the adversary can take control of names using judicial and executive powers (for example by confiscating names or forcing third parties to misdirect users to adversary-controlled impostor sites). Computationally, the adversary is allowed to have more resources than all benign users combined.

The adversary may directly compromise the computers of individual users; for the security of the system as a whole, we only require that the impact of such attacks remains localized. The rationale for being able to make such an assumption is that the economic and political cost of such tailored methods is very high, even for a state actor. Similarly, the adversary cannot prevent the

[1] which is not originally under the control of the adversary itself

use of cryptography, free software, or encrypted network communication. The adversary is assumed to be unable to break cryptographic primitives. As far as network communication is concerned, we assume that communication between benign participants generally passes unhindered by the adversary.

We note that as more states and big operators use GNS for managing their identity systems, the fewer state actors there will be that might be willing to attack it: as participants in the DHT cannot know which requests serve which users, a denial of service (DoS) attack would *indiscriminately* destroy the service for everybody. Thus, the privacy features of GNS make a key contribution towards cyberpeace.

5.2 Security goals

In this section we discuss the security goals that GNS and re:claim aim to satisfy. In the following, we present the respective cryptographic algorithms and architectural design choices that allow GNS.

5.2.1 Availability

By not relying on a centralized service that serves user attributes and allows the user to manage authorizations, our system provides high availability guarantees especially in the face of powerful attackers such as nation states that may utilise "lawful" interception techniques. One major drawback of decentralized services with peer-to-peer connectivity is the possiblity 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 decentralizing the service that allows users to manage identities and selectively share attributes. This is achieved through the use of GNS where the user acts as the main authority over the data.

As GNS is built on top of the R^5N [12] DHT. R^5N is designed to perform well in restricted-route environments with malicious participants. re:claim 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.

5.2.2 Authenticity and integrity

Data stored in GNS is signned using the user's private key, ensurign data authenticity. 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.

For the asymmetric cryptography, GNS uses elliptic curve cryptography [6] for which efficient implementations exist even for embedded systems [16]. Specifically, we use the ECDSA signature scheme with Curve25519 [43] in our implementation. The public key is used to identify zones, where the private key is used to sign the published values including the delegation of subdomains. If desired, the private key can also be used to prove the ownership of one identity. The security proof thus directly follows from the security of the underlying signature scheme.[2]

The underlying security assumption for GNS is the Elliptic curve discrete logarithm problem (ECDLP). We settled on Curve25519 as it is designed to minimize security risks from the implementation while also achieving good performance properties. Curve25519 is based on the equation $y^2 = x^3+486662x^2+x$ using the modulus $p = 2^{255}-19$. According to SafeCurves[3] it is one of the curves having the properties required for Elliptic Curves Cryptography (ECC). Today, no effective attack is known on Curve25519 elliptic curve cryptography.

5.2.3 Confidentiality

Data stored in the DHT is encrypted using AES and Twofish using independent, HKDF-derived [19] IVs and symmetric encryption secrets. The use of independent IVs and encryption secrets ensures that the security is additive: the adversary has to break both AES and Twofish.

To generally ensure confidentiality of attributes in re:claim, we propose to encrypt them using ABE before they are stored in the name system. Through the ABE-layer of re:claim, requesting parties are issued user keys that only allow decryption of those attribute subsets that they are authorized to access. We propose to boostrap 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

[2]The additional multiplication of both the private and public keys with another scalar for privacy obviously does not affect the security properties.

[3]`https://safecurves.cr.yp.to/`

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).

5.2.4 Privacy

We argue that in centralized authentication and authorization 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. GNS mitigates this issue by protectinng the confidentiality of queries and responses as well as metadata.

GNS uses asymmetric cryptography for storing keys and signing the delegations. Each user generates one or many public private key pairs. The keys are used for signing all the values inserted inside the distributed hash table (DHT). Both queries and replies inside the DHT are encrypted, so as to ensure that they cannot be linked to the zone. We discuss details in Section 6.2. The user querying the DHT uses an encrypted query key in such a way that the nodes that store the respective value are able to verify and provide – but not decrypt (!) – the answer. Thus, GNS implements a form of private information retrieval, where even the nodes in the P2P network that observe the request it is not disclosed what the person is looking for. Only nodes that are able to guess the query key are able to confirm that a query is for a specific piece of data (this attack is discussed further in Section 5.5.1). Furthermore, since data is replicated in the network, nobody ought to be able to monitor and or control all the queries.

Due to the use of a DHT, all GNS queries go to the same fully decentralized and shared global infrastructure instead of operator-specific servers. This provides censorship-resistance and makes it impossible to target a zone-specific server because all machines in the DHT are jointly responsible for all zones — and censorship by the peers is also infeasible, as the key-value pairs do not reveal which zone they belong to.

For re:claim, 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 typically has limited access to the attribute data of a user, and furthemore cannot observe access patterns of other services.

5.3 Secure implementation

Our implementation was assessed using static analysis available from `gcc`, `clang`, CodeSonar and Coverity. Aside from issues identified as false-positives, the logic related to GNS is free of defects that could be found via these static analysis tools. We have also run the code using dynamic instrumentation, using both `valgrind` as well as address sanitizer and undefined behavior instrumentations available from `gcc/clang`. Again, to the best of our knowledge no defects remain. We also used `lcov` to assess our test case code coverage, and while some error paths are not covered, almost all of the interesting code paths are covered by our tests. Finally, our releases are cryptographically signed and developers must cryptographically sign every commit, and every change is automatically sent to a public mailinglist for public review.

The cryptographic primitives used are from libgcrypt, a widely audited free software cryptographic library which is also used by GnuPG. We use its pseudo-random number generator, Curve25519 (256 bits), AES-256, Twofish and SHA-512 implementations. Curve25519 has (roughly) a security level of 128 bits, thus our overall security level from the various primitives should also be 128 bit.

5.4 Data flow

Figure 5.1 illustrates the data flow in the system. Users maintain their zone data locally in their database. Records that other users may use are encrypted (not shown) and published in the DHT (step 0). Other users that know both the public key and the label (i.e. by starting with data in their own local database, steps 1–3) can query the DHT (step 4–5), verify the signature (not shown) and decrypt the record (not shown). Access to the DHT could be performed over an anonymous channel to ensure that the DHT is completely oblivious to who is using the infrastructure.

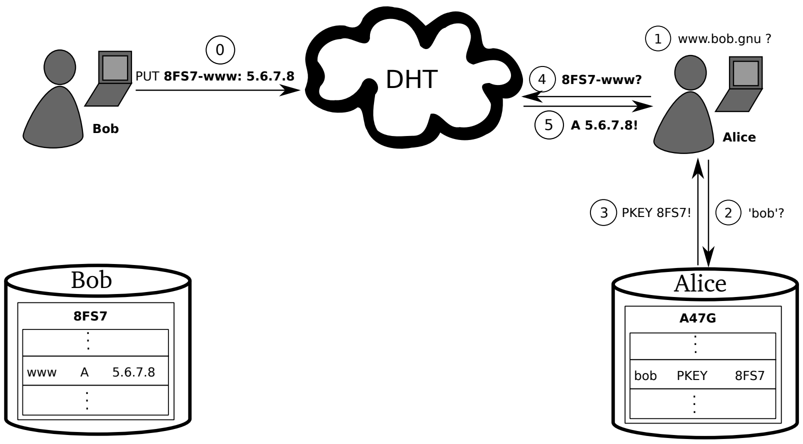


Figure 5.1: Data flow in the GNU Name System.

5.5 Attack vectors and mitigations

In terms of resistance to denial-of-service attacks, our reliability experiments have shown that even a deployment that is utilized at capacity does not suffer from catastrophic failure when exposed to churn or downtime affecting 30% of peers (Section 4.2). In this section we focus the discussion on further attack vectors.

5.5.1 Confirmation attack

Given GNS's cryptographic approach, an adversary can only perform a confirmation attack; if the adversary knows both the public key of the zone and the specific label, they can perform the same calculations as a peer performing a lookup and, in this specific case, gain full knowledge about the query and the response. As the DHT records are public, this attack cannot be prevented. To rule out this attack, private information retrieval would be necessary, but result in linear retrieval costs [32] and thus a system that would be unable to store the large number of records required of practical solutions.

However, we note that users can use passwords for labels or secret public keys[4] to restrict access to zone information to authorized parties. The presented scheme ensures that an adversary that is unable to guess both the zone's public key and the label cannot determine the label, zone or record data.

5.5.2 Revocation

Another attack vector might be to compromise a user's private key, for example via malware on the target system. While GNS cannot prevent this, it does ensure that the user can limit the damage as soon as the compromise has been detected by revoking the old key.

The GNS zone revocation mechanism is rather hard to disrupt for an adversary. The adversary would have to be able to block the flood traffic on all paths between the victim and the origin of the revocation. Thus, our revocation mechanism is not only decentralized and privacy-preserving, but also much more robust compared to standard practice in the X.509 PKI today, where blocking of access to certificate revocation lists is an easy way for an adversary to render revocations ineffective. [27]. This has forced vendors to include lists of revoked certificates with software updates. [20].

5.5.3 Deleting information

The system is also safe against deleting of information and therefore censorship. Each information being replicated at r semi-random locations, where r is typically in the range of dozens or possibly hundreds (depending on the size of the network) of peers. It is thus not possible for one actor (or even for a group of

[4]Public keys need not be public! This is a case where common nomenclature creates false associations that are hard to break.

actors or a state) to remove all the occurrences of a given key-value pair. Since the peers storing the information are randomly chosen, the probability is small ($\frac{1}{3^r}$) that even a state controlling $\frac{1}{3}$ of the nodes would have all the replicas of one key-value pair.

5.5.4 DHT value manipulation

The system is resistant against value spoofing, as it is not possible to change the information in a value in the DHT without siging it. There can not be done unless the attacker has access to the private key of the victim. As long as private keys remain secret, it is not possible to propagate any information in the DHT that is not correctly signed and comes with a valid expiration date. The expiration date ensures that peers do not propagate expired information, and thus also prevents an adversary from replaying stale data.

5.5.5 Identity theft

One of the major problems for authentication systems is identity theft. This occurs when someone can impersonate someone else. For instance if Alice is normally controlling a device Dave, and Bob wants to impersonate Alice to control Dave. Dave will use a challenge response to test if Alice has the secret key corresponding to her identity. Therefore, as long as the key of Alice remains secret, Bob has no possibility to solve the challenge and can not take the place of Alice.

Real-life identity theft typically involves the adversary obtaining sensitive attribute data of the victim. With GNS and re:claim, users can share attributes with services in a controlled way: the service only receives access to the attributes it requires, as approved by the user. Furthermore, unlike with other solutions, there is no central identity provider, and thus no place from where "all" attributes of many users are stored. As a result, GNS minimizes the chances of sensitive attributes being disclosed to an adversary.

5.5.6 Traffic amplification

GNS records are about as compact as those of DNSCurve. However, there is no real potential for traffic amplification as the DHT relies on secure connections with a proper handshake that prevents spoofing attacks. The DHT connections are long-lived, limiting the performance impact of the cryptographic handshake compared to DNS-over-TLS.

5.5.7 Quantum computing

GNS is based on Curve25519. If quantum computers of sufficient size are ever created, GNS and many other deployed public key systems would be broken as well [4, Table 1]. Today, it is unclear whether a post-quantum cryptosystem with adequate semantics and performance exists. [4]

5.6 Secure key management

User will need to generate and manage their keys using a software or hardware vault. GNS is an open system, so many different vaults can be implemented, providing different graphic user interfaces (GUIs), for example those tailored to address specific disabilities.

Existing GNS vault implementations can use Postgres, MySQL, Sqlite or flat-file databases to store zone data, and zones can be manipulated via command-line, browser or Gtk+-based user interfaces.

5.7 Communication protocol security

GNS itself does not mandate any particular communication protocol, as it focuses on making key material and credentials available to the applications. We expect GNS to be frequently used in combination with TLS which has been implemented by supporting TLSA records in GNS. TLS v1.3 has undergone extensive security evaluations from the community [35].

More importantly, GNS can be used with a wide range of "legacy" authentication protocols, such as HTTP basic authentication (basically username and password). While such solutions may be vulnerable to well-known attacks, this is simply a fundamental trade-off between backwards-compatibility and security: GNS can help secure addressing and provide public keys or pre-shared keys, but the communication protocol using these keys may ultimately limit the security of the overall solution.

5.8 Comparison

Table 5.1 compares the security properties of GNS with those of major competitors. We provide the minimum number of bits of security guaranteed by the protocol design for end-to-end security; if various standards-compatible settings exist, we report the weakest setting. We also reviewed whether the design or protocol has been published in appropriate academic security conferences and whether it uses common state-of-the-art cryptography, that is neither exotic primitives nor "broken" primitives are acceptable. Finally, we review whether the system architecture distributes risks, that is whether security failures of individual providers or users have a localized security impact.

Table 5.1: Security comparison. TLS is used to indicate that theoretically weak TLS cipher suites (including the NULL cipher according to the TLS standard) could be used. Also, we consider TLS 1.2 to be borderline "state-of-the-art" as it will be obsolete soon with TLS 1.3.

Technology	Guaranteed minimum end-to-end security level	Assessment in published academic research	Uses state-of-the-art cryptography	Distributed risk from failures
DNSSEC	0 bits	✓	✗	✓
X.509 CAs	0 bits	✓	✓	✗
verimi.de	TLS	✓[2]	TLS 1.2	✗
id4me.org	TLS	✓[2]	TLS 1.2	✓
keyp.io	unspec.	✗	unspec.	✓
miracl.com	unspec.	✓	✓	✓
DecentID[15]	unspec.	✓	unspec.	✓
fidoalliance.org	TLS	✓[2]	TLS 1.?	✓
GNS+re:claim	128 bits	✓	✓	✓

Chapter 6

Privacy and data protection

The right to privacy is one of the Human Rights protected by the European Convention on Human Rights [29, Article 8]. It is also protected by various national legislation and at the level of the European Union by the General Data Protection Regulation (GDPR) [10].

The main privacy objectives required by the European GDPR are the following [10]: Data must be collected for a specific purpose, collection must be transparent, and the collected data must contain only the minimal data necessary for the purpose. The data must be stored for no longer time than necessary. The persons collecting data must provide technical and organisational safeguards for security and the data must be accurate and allow users to correct inaccuracies.

While one cannot technically ensure that data requested is only used for the specified purpose or that companies only request the minimal data necessary, GNS can help address the other concerns: GNS and re:claim empower users to choose when disclose what private information, and if they choose to do so, make it convenient to keep the shared information current, and also to later revoke access.

GNS has been designed to minimize information leakage and preserve privacy for users publishing as well as for users retrieving information. Most importantly, whereas most identity systems will require some kind of trusted identification service, in GNS, we have a totally decentralized infrastructure that is oblivious to the operations performed by its users.

The system design empowers the user: Each user is at the center of the decision making about who is granted access to their private data. Most importantly, users have tools for creating as many pseudonyms as needed, and can thus limit how collected data can be linked. Whereas centralized system based on a public key infrastructure (PKI) will require everybody to trust the same set of certificate authorities (CAs), in GNS, every actor can select which third parties are trusted.

6.1 Pseudonyms and the right to be forgotten

A key approach users have to protect privacy is to create separate identities to separate the different traces of their activities on the Internet. For example, recent studies have shown that many teenagers maintain multiple accounts on certain online platforms [30]. The use of pseudonyms is also the only practical approach to realize the right to be forgotten, as rarely users will want to delete *all* traces of their electronic lives.

GNS offers the possibility for users to separate their traces by allowing users to generate as many pseudonyms as they want. This includes the possibility to use each pseudonym only once. This makes it more difficult to link the information associated with the same person across activities. This principle is quite similar to the system used for transactions on Bitcoin where fresh wallet addresses can be used to make it harder to link wallets to users. The same is possible with GNS: one may not know which pseudonym belongs to which person, and moreover, even if one pseudonym is linked to a particular identity, GNS allows users to keep their different pseudonyms unlinkable.

Naturally, in practice GNS cannot fully solve the problem of partial identities [40] where people use some kind of unique identifier (first and last name for instance) across their activities. In this case, it may still be possible to link the partial identities. However, GNS identifies pseudonyms by a public key and not by a username. This helps eliminate at least one commonly used unique identifier.

6.1.1 Link to legal identities

For many actions, pseudonymity is useful. In most of the use cases, the online service does not need to know what is the real identity of the person using this service. This can be the case for ordering music on a streaming service, ordering goods from a ecommerce web site or visiting an email service. But in some cases, the possibility to link a pseudonym with the legal identity is necessary for accountability issues. If a problem occurs, in some cases it should be possible to access the legal identity of one client.

For example, we could imagine that states may want to have a means to access the identity of persons receiving parcels in packstations or pickup stations (as discussed in Section 7.1). They may fear that people will order drugs on the Darknet and receive their illicit goods this way. This use-case can be addressed with the creation of some agencies that are charged with certifying pseudonyms. Those agencies would know the link between the pseudonym and the legal identity of the person. They maintain accounts for people they identified and offer them the opportunity to create any number of certified pseudonyms.

For instance, Mr John Smith is identified by the agency “myagency”. He receives an account. When he wants a parcel to be delivered he generates a new zone entry `johndoe1234.myagency`, that is linked with the ego’s public key. The ego can then be used to receive the packet. The item is sent to a nearby packstation, to the ego certified by the record `johndoe1234.myagency`. Neither the e-commerce site, nor the delivering firm have an idea of the identity of their client. The client can access his packet without any other information using his private key. But in case of problems, the police may ask “myagency” to disclose the legal identity corresponding to the ego `johndoe1234.myagency`. Since the users may use many agencies, nobody can easily link all the egos of one person. The agencies do not have any information on what the person did with the given ego, since the record is in the DHT and the agency is not informed about services using it.

The system offers law enforcement access to the identity of an ego in case of illegal behaviour, while people can remain anonymous as long as they behave legally. The users retain the opportunity to use as many identities as desired. They can also use different agencies to get more independent identities.

6.1.2 Certified attributes

One can also imagine that some of the agencies decide to certify that a person is over 18 and to provide a special type of pseudonyms to the user with this age. For example, persons above the age of 18 may have the possibility to create identities like `asdf2343eef.over18.myagency`.

Instead of using GNS zones, certification providers could also provide users with signatures using attribute-based credentials [36] which can then be shared as GNS records. Our re:claim uses this style of sharing attributes as GNS records, which is appropriate in cases where attributes express more than a simple set membership.

IT services requiring that users are over 18 will decide which agencies they trust for the verification of the age, then they will accept all users whose identity is provided by those agencies. The services will still not have any possibilty to learn the legal identity of the users visiting their sites. Neither will have the certification agency the possibility to check the use of the certificate it provided.

6.2 Privacy of record data

In addition to users being able to create any number of GNS zones to use as unlinkable pseudonyms, GNS also protects the actual data stored within each zone. We will now focus on these privacy properties offered by GNS.

6.2.1 Existence of zones

To begin with, the mere existence of a zone is never implicitly disclosed. In GNS, records under label l in zone $Z = zG$ are stored under the DHT key

$$Q_{l,Z} := H(lZ) \tag{6.1}$$

and signed by the public key lZ (see Section 2.1.3). Thus, peers in the DHT would have to solve the Elliptic curve discrete logarithm problem to derive the zone Z from lZ. For appropriate curves (like GNS's choice of Curve25519) this problem is untractably hard today.

Naturally, the public key of a zone will have to be explicitly disclosed to other parties that should have (read-only) access to the zone. But this disclosure itself can usually happen via GNS records, which themselves are again confidential.

6.2.2 Response privacy

GNS responses $B_{l,Z}$ from zone Z published under label l containing GNS records $R_{l,Z}$ are symmetrically encrypted:

$$B_{l,P} := E_{\texttt{HKDF}(l,Z)}R_{l,Z}), lZ \tag{6.2}$$

Assuming the cryptographic hash function (for GNS, SHA-512) used by $HKDF$ is sufficiently strong, the shared secret $HKDF(l, Z)$ can be assumed to be random in the random oracle model as long as either l or Z are secret. As discussed in Section 6.2.1, Z is never implicitly disclosed by GNS and thus *may* act as a shared secret between the zone's owner and services trying to resolve records in the zone. Similarly, the label l *may* be chosen to be a high-entropy password. Assuming the hash function can be modeled as a random oracle, decrypting $B_{l,P}$ requires access to both Z and l, allowing users to use either or both for access control. Naturally, it is also possible to use well-known values for both to publish records that should simply be public.

6.2.3 Query privacy

For records that are not published under such well-known $\langle l, Z\rangle$ names, GNS also achieves query privacy: given $H(lZ)$ (the query) or even lZ from the answer, it is not possible to compute l or Z: Given just l or just Z, computing the other would require solving the discrete logarithm problem. Given neither l or Z, the problem is even unsolvable as for any Z' in the group there exists is a solution l' such that $lZ = l'Z'$.

6.2.4 Zone enumeration privacy

As without knowing the label l and the zone Z it is not possible to derive the query $H(lZ)$, it is also not possible to enumerate all records in a zone without brute-force guessing of all labels. Like with the original DNS (and unlike DNSSEC with NSEC3 records [21]), brute-force zone enumeration would require an infeasibly large number of queries to the DHT.

6.2.5 Revocation check privacy

Revoking a zone's public key in GNS discloses the (former!) zone's existence to the world, but only at the time where it is explicitly no longer in use. Publishing the revocation information globally via the flooding mechanism used by GNS has a significant advantage to privacy as well. Protocols like the online certificate status protocol (OCSP) [7] disclose to OCSP servers whenever a user wants to check whether a zone key has been revoked. As checking for revocations is a common step just before using information from a zone, this is a privacy nightmare. With GNS-style revocation, checking whether a key has been revoked is a cheap and *local* operation which thus does not leak any information to other parties.

6.3 Data protection by default

Finally, we should point out that our reference implementation does not collect any data. In particular, our software does not report personal information nor telemetry data back to us. The software is available from a global pool of FTP mirrors, thus we do not even obtain download statistics. However, our system uses a privacy-preserving and Byzantine fault-tolerant method to obtain a global estimate on the total number of peers that are online [13].

Privacy-by-default is also implemented in the user interface. In particular, even for records the user creates locally, the user must explicitly consent to publish them to the world (and then share the zone's public key and label to make them effectively available). If the user enters record information but does not check the "public" box, even the encrypted record data is not shared in the DHT.

Our main project Website does not use any tracking, does not include any third party resources, requires TLS with a strong ciphersuite and is pinned to use TLS-only by major browsers. It also works well even if the user disables JavaScript. Software releases are signed using GnuPG keys verified by the well-known GNU project. System documentation is provided as a manual that is part of the main download. Thus, users can access the documentation for the system locally without leaving further traces of what manual pages they are reading on the Internet.

6.4 Summary

Table 6.1 compares the privacy properties of GNS with those of major competitors. Zone privacy is about the system not disclosing to unauthorized parties the existence of accounts, records or attributes in the absence of requests. A solution achieves private information retrieval if even during an access to information systems facilitating the access cannot identify which information unit was accessed. Selective attribute sharing implies that the user can precisely control which attributes (or records) are shared with which parties. Revocation means that the current implementation allows the user to securely revoke access to attributes or accounts / pseudonyms. Revocation does does not require retroactively removing data that was already shared with others, only preventing future access to the attribute or account.

Table 6.1: Privacy comparison.

Technology	Zone privacy (existence of records)	Private information retrieval	Selective attribute sharing	Revocation
DNS	✓	✗	✗	✓
DNSSEC	✗	✗	✗	✓
X.509 CAs	✗[18]	✗	✗	✓
openid.net	✓	✗	✓	✓
verimi.de	?	✗	✓	✓
id4me.org	✗	✗	✓	✗
keyp.io	✗	✗	✗	?
miracl.com	✗	✗	✗	✗
DecentID[15]	✗	✗	✓	✗
fidoalliance.org	✓	✗	✗	✓
GNS+re:claim	✓	✓	✓	✓

Chapter 7

Applicability

The usability studies from Chapter 3 demonstrate the applicability of our authentication solution for use-cases across a wide range of sectors including health, telecommunications (DNS), government (Reclaim identity management), and the Internet of Things (IoT).

Existing business models can typically continue in the context of GNS deployments. For example, just like we have DNS registries today, in the future we can have GNS registries. However, a crucial difference is that with GNS security and privacy of the customers would be significantly improved.

Beyond the use-cases where we have performed usability studies, we are working on using GNS in various other applications. This section introduces some of them.

7.1 Certified delivery

Delivery of goods generally requires disclosing one's postal address. The address may later be missused for sending advertisement or to attack privacy of the persons. The delivery at home is also problematic if nobody is at home when delivery person comes to bring the parcel. That is the reason why some firms have developed new system for delivering parcels out of home: *Pickup-station* from the French *"La Poste"*[1].

We can use GNS to secure delivery of goods to the right person. A user would provide a Web site with access to selected attributes of their GNS zone. For shipping, they might share the location of a pickup station close to their home. In order to collect the parcel, the client goes to the station, where the package is. The user enters the GNS identity used for ordering and then the station starts a challenge response with the smart phone of the user containing the chosen key.

As a result, the delivery happens without disclosing private data to the store, preventing the firms from sending unsolicited mail while also assuring the firm

[1]See `https://www.pickup.fr/relais/pickup-station.sls`

that parcel was delivered correctly.

If law enforcement agencies need to control the delivery of goods to prevent people to receive illegal products using this system, this can also be solved using CAs that certify GNS identities. Suppose such an agency is named `agency`. Then a user Anna Muster, could create an account at the agency. To get an account she needs to show a real ID and to prove her legal identity. Then she can create a master account `anna_muster.master.agency`, which will only be used to identify her to the agency system. This login will then be used to certify a fresh identity each time a private login is needed. For instance, if Anna Muster buys a sex toy and wants them to be delivered, she could create ask `agency` to certify a fresh identity: `1238dijnciuwejniu9.agency`. This identity is only linkable to the legal identity of Anna Muster by `agency`. In case of prosecution, law enforcement can thus identify and convinct. However, outside of this legitimate use-case, the one-time pseudonym can not be linked to her.

7.2 Telephony

GNUnet uses the GNU Name System as a PKI for P2P voice conversations. Existing P2P voice applications, such as Skype, typically use a centralized service for user authentication. This is highly problematic as this is one place where attacks can be mounted against the system, from denying access to interception and impersonation. One alternative is the use of X.509 client certificates for users, which is, for example, supported by Mumble[2]. However, the use of certificate authorities (CAs) in X.509 allows a large number of CAs to act as trusted third parties, with the weakest CA determining the security of the system. Furthermore, the cost of certification or the desire to use pseudonyms drive users to use self-signed certificates, which provide no more than TOFU security.

GNUnet includes a simple conversation service which uses GNS and GNUnet's implementation of the Signal protocol [33] to establish a secure connection between the participants. GNS "PHONE" records contain the public key of a peer and an integer specifying the *line* under which a user application realizes a phone service for incoming calls. A *call* can then be made by specifying the GNS name that resolves to the "PHONE" record. The connection to the target peer is then secured using ECDHE and AES using the public key of the target peer. The caller signs the call request with his zone key. The callee performs a reverse lookup against the caller's public key to determine the *caller id*. If the caller's public key is not in the callee's zone, a name is generated from the public key instead.

[2]`http://mumble.sourceforge.net/`

7.3 Online social networking

The SecuShare project[3] is building a decentralized P2P social networking application [42] by combining the PSYC multicast protocol with the GNU Name System. GNS zones are used to identify participants. Name resolution is used to address objects published by participants, such as their profiles, communication channels and rooms enabling group communication.

The GNS zone of a user also corresponds to their address book, associating names of other users with their zones via PKEY records. SecuShare's implementation is still experimental, but the applicability of GNS to represent the social graph for online social networking applications is clearly demonstrated by the SecuShare design.

7.4 Tor hidden services

GNS can also be used to assign more memorable names for privacy-sensitive applications. Specifically, the Tor project is discussing the possibility of using GNS to provide memorable names for its ".onion" services.[4]

7.5 Medical applications

Finally, BFH is working on a project with Ypsomed AG, a major industry player in the health sector that is considering the use of the GNU Name System for management of private health data and optimize treatments. The Ypsomed Group is a leading developer and manufacturer of injection and infusion systems for self-medication. Patient privacy and data integrity are critical for Ypsomed, and the company is interested in the ability of GNS to provide a mechanism for secure and private asynchronous data exchange between the various actors in the Ypsomed value chain.

7.6 Summary

Table 7.1 compares the application domains of GNS with those of major competitors. We say a technology enables Internet service addressing if it can be used to obtain network routing information for Internet services. We say it provides service authentication if it can be used establish a secure channel where the service is authenticated. We say it can be used for Internet-of-things mutual authentication if it is suitable for mutually authenticated access to personal devices of end-users. We say it supports single sign-on (SSO) with attribute sharing if end-users can use the technolology to (easily) authenticate themselves and provide attribute data to network services.

[3]`https://secushare.org/`

[4]`https://trac.torproject.org/projects/tor/wiki/doc/OnionServiceNamingSystems` and `https://blog.torproject.org/cooking-onions-names-your-onions`

Table 7.1: Applicability comparison.

Technology	Internet service addressing	Internet service authentication	IoT mutual authentication	Single sign-on attribute sharing
DNS	✓	✗	✗	✗
DNSSEC	✓	✓	✗	✓
X.509 CAs	✗	✓	✗	✗
openid.net	✗	✗	✗	✓
verimi.de	✗	✗	✗	✓
id4me.org	✗	✗	✗	✓
keyp.io	✗	✗	✗	✓
miracl.com	✗	✗	✓	✓
DecentID[15]	✗	✗	✗	✓
fidoalliance.org	✗	✗	✓	✗
GNS+re:claim	✓	✓	✓	✓

Chapter 8

Compatibility

In this section we will explain what protocols, hardware and software our solution is compatible with, as well as how new systems can be integrated with our solution.

To begin with, the GNS Socks5 proxy is compatible to RFC 1928 [23], the DNS-to-GNS proxy is (largely) binary compatibile with RFC 1035 [28] and applicable subsequent DNS extensions, and re:claim is compatible with the OpenID Connect standard.

GNS assumes that the user is in control of their own hardware and the private keys entrusted to it. How the user secures their hardware or their local login process is thus orthogonal to GNS. Users are free to use any conceivable multi-factor authentication (HSM, biometrics, passwords) for this process.

8.1 Hardware requirements

Hardware requirements for using the GNU Name System are pretty low. GNUnet, including the GNU Name System, has been packaged for LEDE (previously known as OpenWRT)[1] which is a distribution focusing on low-end hardware such as SoHo routers and NAT boxes.

On LEDE, GNUnet with GNS is known to run fine on systems with a total of 64 MB of RAM. The GNUnet packages require a total of 8 MB ROM. GNUnet requires a 32-bit CPU and has been running on Sparc, Intel/AMD, ARM and MIPS processors in the past.[2] Even a low-end Cortex-M4 is adequate to handle the most expensive cryptographic operations of the GNU Name System (EdDSA verification) in about 1.2 million cycles [16]

[1]`https://github.com/openwrt/packages/tree/master/net/gnunet/files`
[2]Our Sparc is currently not operational.

8.2 Operating systems

GNUnet includes an operating system abstraction layer with provisions for running on Windows, Solaris, OS X and GNU/Linux.

For tight integration with the operating system's name resolution, plugins to integrate with the Windows and GNU libc name resolution processes exist.

The community is working on a port to Android/iOS.

8.3 Network technologies

GNUnet typically operates on top of TCP/IP. However, GNUnet can also directly run over WLAN. The community is working on having peers communicate directly over BLE without TCP/IP.

8.4 Application programming interfaces (APIs)

Applications can use the GNU Name System for name resolution in various ways. This section presents the existing set of mechanisms.

8.4.1 Interoperability with DNS resolution

To integrate with DNS, the GNS implementation needs to intercept all DNS queries for zones configured by the user to use GNS, and to inject appropriate responses. Furthermore, other TLDs should then be forwarded to the traditional DNS system. Our current implementation provides three alternative methods to do so:

- On GNU systems, a plugin for the name services switch (NSS) [14] in GNU libc can be used to answer GNS queries before a DNS request is ever created. Mechanisms similar to NSS exist for other platforms; we also have an equivalent plugin working on Microsoft Windows.

- The resolver configuration (usually `/etc/resolv.conf`) can be changed to point to an IP address (i.e. 127.0.0.1) with a modified DNS resolver. We have implemented a DNS-to-GNS gateway which resolves names in GNS zones internally, and acts as a simple proxy for all other TLDs, passing those requests to an actual DNS server.

- Browsers can be configured to use an HTTP SOCKS Proxy and delegate name resolution to the proxy. In this case, our `gnunet-gns-proxy` performs GNS (or DNS) name resolution, and generates X.509 certificates on the fly if TLSA-based validation of the TLS connection succeeded.

The NSS-based approach has the key advantage that it allows GNS to learn the identity of the user that issued the query. As a result, it can fully personalize the GNS lookup on a per-user basis for multi-user systems. A potential disadvantage is that some applications may bypass the operating system and directly contact a DNS resolver. Here, the network-level approaches can provide an alternative.

The DNS-to-GNS proxy is useful to allow legacy systems to access the GNS distributed database without installing GNS or changing their system configuration.

8.4.2 APIs for new applications

`libgnunetgns` provides a C API for name resolution using the GNU Name System. Clients first connect to the GNS service using `GNUNET_GNS_connect()`. They can then perform lookups using `GNUNET_GNS_lookup()` or cancel pending lookups using `GNUNET_GNS_lookup_cancel()`. Once finished, clients disconnect using `GNUNET_GNS_disconnect()`.

Instead of using `libgnunetgns`, it is also possible to connect to the GNU Name System service using TCP (on loopback) or using a UNIX domain socket (UDS). The TCP/UDS protocol consists of only two messages, the `LOOKUP` message and the `LOOKUP_RESULT`. Each `LOOKUP` message contains a unique 32-bit identifier, which will be included in the corresponding response. Thus, clients can send many lookup requests in parallel and receive responses out-of-order. A `LOOKUP` request also includes the public key of the GNS zone, the desired record type and a few flags. Finally, the `LOOKUP` message includes the name to be resolved. The response includes the number of records and the records themselves in a well-defined serialization format.

Finally, GNUnet includes a command-line tool `gnunet-gns` which allows users and applications to resolve names in a way that is equivalent to what `nslookup` or `dig` offer for DNS.

8.4.3 Defining new record types

The `libgnunetgnsrecord` library is used to manipulate GNS records (in plaintext or in their encrypted format). Applications mostly interact with `libgnunetgnsrecord` by using the functions to convert GNS record values to strings or vice-versa, or to lookup a GNS record type number by name (or vice-versa). The library also provides various other functions that are mostly used internally within GNS, such as converting keys to names, checking for expiration, encrypting GNS records to GNS blocks, verifying GNS block signatures and decrypting GNS records from GNS blocks.

To define a new GNS record type, one needs to write (or extend) a plugin for `libgnunetgnsrecord`. The plugin needs to implement the `gnunet_gnsrecord_plugin.h` API which stipendulates the existence of four basic functions that are needed by GNSRECORD to convert typenames and values of the respective record type to strings (and back).

The `libgnunetgnsrecord` library will then locate, load and query the appropriate plugin. Which plugin is appropriate is determined by the record type.

Table 8.1: Compatibility comparison.

Technology	DNS	LDAP	X.509	OpenID
DNSSEC	✓	✗	✓[3]	✗
X.509 CAs	✗	✗	✓	✗
verimi.de	✗	✗	✗	✓
id4me.org	✓	✗	✓	✓
keyp.io	✗	✓	✗	✓
miracl.com	✗	✗	?	✓
DecentID[15]	✗	✗	✗	✗
fidoalliance.org	✗	✓	✓	✓
GNS+re:claim	✓	✗	✓	✓

The `libgnunetgnsrecord` library loads all block plugins that are installed at the local peer and forwards the application request to the plugins.

Record types should be below 2^{16} if the respective record type is binary-compatible to a DNS record type managed by the respective Internet Assigned Numbers Authority (IANA) registry. Record types above 2^{16} should be used for GNU Name System-specific records. The range above 2^{16} is managed by the GNUnet Assigned Numbers Authority (GANA).

8.5 Summary

Table 8.1 compares the compatibility properties of GNS with those of major competitors. We say a technology is compatible with DNS if it can be used to resolve domain names like DNS does. We say a technology is compatible with LDAP if it can use LDAP directories. We say a technology is compatible with X.509 if it can be used to securely obtain X.509 certificates. Solutions that we consider compatible with OpenID Connect must implement (at least) the OpenID Connect Core specification [3].

[3]`https://openid.net/specs/openid-connect-core-1_0.html`

Chapter 9

Affordability

To estimate the cost of the solution, we consider the three main cost categories: operating expenses for the infrastructure, licensing costs and one-time integration costs. To estimate operating expenses, we deployed the system for a few months in Amazons EC2 cloud. Amazon offers detailed billing, allowing a comprehensive assessment of commercial deployment costs. We note that our deployment used expensive "on-demand" resources, a permanent deployment on dedicated systems should thus be even cheaper in the long run.

9.1 Deployment cost in EC2

We investigated the cost of deploying the service in the Amazon EC2 cloud. For most of our use cases, the number of records stored as well as the number of queries made by the applications in our limited deployments is very small, making useful measurements difficult. However, the use case involving the replacement of DNS is different, as large data sets exist in the form of more-or-less public DNS zone files. Thus, the DNS use case makes for a good case study for what it would cost to deploy GNS at scale.

To evaluate the cost of large-scale GNS deployments, we we imported the entire ".fr" TLD from AFNIC into the DHT.

9.1.1 Memory consumption for zone import

Our importer requires a bit above 1 GB RAM to keep a list of the 5 million domain names, their respective expiration times and associated data structures in a heap (sorted by expiration time). While memory consumption could be reduced by using the database for this, this would double the number of queries to the database which is not a reasonable trade-off in this case.

9.1.2 Compute time for zone import

The GNU Name System cryptography consumed most of the CPU load. It takes about 576 CPU credits to derive the key material and to re-sign all 5 million records.[1] Given a typical expiration time of 48h for records in the ".fr" zone, approximately 12 CPU credits/hour would thus be (barely) sufficient to maintain the signed zone.

Thus, at a **minimum**, maintaining the ".fr" zone in the GNU Name System requires a dedicated `t2.small` instance (which also includes a sufficient amount of RAM). The annual cost of a `t2.small` instance at Amazon is (at the time of writing) ≈ €100.

In practice, this is a bit low as the CPU would be at 100% load and may not fully keep up with the zone's changes. Furthermore, some redundancy would likely be good to cover outages. Thus, we ran our actual experiments on two `t2.medium` instances, which cost €200 annually.

9.1.3 Bandwidth consumption for DNS queries

We note that the bandwidth consumption of the import process is minimal, as 24-48 DNS queries and responses per second (with typically a few hundred bytes) barely registers on modern networks.

Furthermore, assuming a DNS registry were to publish their data directly into GNS – which would mirror the other use cases – this limited amount of bandwidth would not even be needed.

9.1.4 DHT replication

As the DNS data is being imported, the GNU Name System's `zonemaster` process begins to publish the encrypted and signed blocks with GNS record sets in the DHT. This is a continuous process, with the zonemaster re-publishing records periodically as well as whenever they change. The various peers providing the DHT service cache the blocks that fall into their range, and route queries and responses for DHT clients.

All DHT peers have the same configuration (modulo private keys and IP addresses that identify the machine), thus if some peers become unavailable, others take over without the service being interrupted. However, if too many peers leave the network too quickly, some cached records may be lost and become unavailable until the `zonemaster` republishes the records.

We have configured the DHT template to fit the requirements of a "t2.micro" instance (1 GB RAM) so that users can run it for "free" for the first year as part of in Amazon's "Free tier". Of the 1 GB RAM, we reserved 500 MB for the DHT to cache results in RAM, thereby eliminating the need for most disk IO. At 500 MB most peers can cache about a million GNU Name System record

[1] We estimated the total time for all 5 million records by extrapolating the velocity observed for a smaller record set on a `t2.micro` instance.

sets, as the cryptographic overheads add about 200 bytes to the few hundred bytes compared to the original DNS data.

Running a “t2.micro” instance for one year costs about EUR 50. Each instance holds roughly one million records. Realistically, we would want 10 replicas per record across the world for increased reliability, thus the computational cost would be EUR 500 per million records annually.

9.1.5 Total cost

For our actual experiments, we used 24 “t2.micro” instances (in three regions) for the DHT and two “t2.medium” instances with one “t2.micro” RDS database each for the DNS import logic. This setup would cost approximately USD 3600 annually, for approximately 5 million records at the reported performance and reliability figures.

Suppose the average user will publish fewer than 50 records in their zones. Even if records are published for self-hosted network services, medical data, references to other entity’s zones, and credentials for IoT devices, 50 should still be sufficiently high on average for a while. Then, the annual publishing cost per user is roughly $\frac{e\,3600\cdot 50}{5000000}$ — or 3 cents — based on the data collected in Section 9.1.2.

Next, suppose each user performs 1000 lookups each day. While this number may seem high for typical authentication problems, it becomes more realistic if we include assume GNS is queried instead of DNS. 1000 lookups multiplied by 50000 bits per lookup (5000 bits payload, multiplied by 10 for DHT internal hops allowing for $\approx 2^{10}$ peers in the EC2 DHT) is 18 GBit of traffic per year, costing an additional 18 cents per user annually.

Consequently, to operate the DHT infrastructure to support the retrieval operations would cost approximately 21 cents per user annually. Appendix A.6 includes the EC2 billing we received from Amazon.

For comparisson, `easydns.com` offers DNS “pro” services at USD 55/year for 5 million queries per month (this is for service providers, not end-users).[2] This is 156 times the number of queries we estimated above that a normal user would make, and our cost calculation scaled to the query rate “pro” users of EasyDNS are allowed would thus be €31. Thus, operating costs for the GNS infrastructure are comparable to those of DNS.

9.2 Licensing costs

The entire software is available for free as free and open source software from `https://gnunet.org/`. Users are not expected to pay any royalties. Furthermore, the community provides support via an IRC channel (`freenode.org#gnunet`) and a public mailinglist (`gnunet-developers@gnu.org`).

[2]`https://easydns.com/pricing`

9.3 One-time costs

The primary one-time costs for the system are the initial integration costs for applications, as well as configuration and installation. We cannot precisely estimate these costs, as they differ widely between applications as well as individuals or organizations. However, will try to point out key aspects of the existing implementation that should make this process feasible.

9.3.1 Cheap integration

The GNU Name System includes a SOCKS proxy which enables HTTP(S)-based applications to use GNS for lookups and TLSA-based X.509-validation without modifications. The GNU Name System also includes plugins into Windows and GNU/Linux name resolution processes (such as GNU libc's NSS) to enable migration from DNS to GNS for applications using the operating systems' stub resolver. Finally, GNS includes a DNS server which answers queries via GNS for GNS-enabled zones, and otherwise forwards to DNS.

9.3.2 Comprehensive integration

Applications that want to make use of advanced GNS features, including new record types, attribute-based encryption or simply the assurance that GNS was used instead of DNS, need to use GNS via the respective API. GNS provides a simple C API, a stream protocol (TCP or UNIX domain socket), a REST API and a command-line tool for resolution. Bindings for some popular languages (such as Python) have also been written (by third parties). For new record types, the main work is to implement functions for the conversion from the binary value to human-readable text and vice-versa.

Further, GNS and re:claim expose REST APIs which allows for a technology-independent integration.

9.4 Summary

Annual operating costs for the system are estimated to cost at best a few cents per user, comparable to those of today's DNS infrastructure cost. Migration costs will depend on the application, ranging from relatively simple installation for systems where compatibility to existing protocols is achieved (DNS, OpenID) to more comprehensive re-development. There is cost for licensing.

Chapter 10

Openness

The GNU Name System and *all* applications and components included in this reported are publicly available as free and open source software under the GNU Affero General Public License (v3+).

We are not aware of any patent claims against the design or implementation, and the consortium also does not hold any patents on the system.

The protocols are documented in a manual that is available under the GNU Free Documentation license at `https://docs.gnunet.org/`. Academic publications on the GNU Name System, the R^5N DHT and re:claim are available from the bibliography hosted at `https://gnunet.org/`.

The reference implementation is publicly available in Git repositories hosted at `https://git.gnunet.org/`. The cryptographic protocols and primitives were developed outside the US and thus do not fall under any export restrictions that we are aware of. GNUnet binary packages exist for a range of free software distributions.

In conclusion, we believe the project has done everything possible to provide the most open, freely available authentication system.

Summary

Table 10.1 compares the openness properties of GNS with those of major competitors. We consider a technology open if the protocol specification is freely available and unencumbered by restrictive copyrights or patents. If the specifications for accessing a service based on the technology are freely available, the technology has an open API. If implementations of all critical components are available as open source software we say it has an open reference implementation (even if such an implementation is not provided by the standardization body itself, if one exists). Finally, the design is gatekeeper-less if there is no entity that has a priviledged position (such as ICANN with DNS) to set rules for using the system (this excludes entities that define the protocol as required for interoperability).

Table 10.1: Openness comparison.

Technology	Open standard	Open API	Open reference implementation	Gatekeeper-less design
DNSSEC	✓	✓	✓	✗
X.509 CAs	✓	✓	✓	✗
verimi.de	✓	✗	✗	✗
id4me.org	✓	✓	✓	✓
keyp.io	✗	✗	✗	✗
miracl.com	✗	✗	✗	✗
DecentID[15]	✗	✓	?	✓
fidoalliance.org	✓	✓	✓	✓
GNS+re:claim	✓	✓	✓	✓

Appendix A

Appendix

This appendix provides auxiliary information about the experiments.

A.1 Reproducing our EC2 setup

To ensure reproducability, we performed various experiments for the reliability (Chapter 4) and affordability studies (Chapter 9) using Amazon Web Services, specifically EC2 (computation) and RDS (database). We have published the respective virtual machine images, thus with the following instructions anybody should be able to reproduce our results.

A.1.1 Create a virtual private Cloud

First, go to the “VPC Dashboard”, select “VPC Dashboard” and then “Start VPC Wizard”. Select “VPC with a Single Public Subnet”. Use “dht” for the “VPC name” and enable an “Amazon provided IPv6 CIDR block” and “Specify a custom IPv6 CIDR”. Leave the other settings unchanged.

Next, select “Subnets” and select “Public subnet” using the “Subnet actions”, enable auto-assignment of public IPv4 and IPv6 addresses. Then create another subnet, “intranet-1”, using “10.0.1.0/24” for the IPv4 CIDR block and “01” for the custom IPv6 CIDR. Create a second subnet “intranet-2” using “10.0.2.0/24” and “02” respectively. Make sure you specify different availability zones for both.

A.1.2 Launching the RDS database

Go via “Services” to the “RDS” dashboard. Click on “Instances” and then “Launch DB instance”. Select “PostgreSQL”. For “DB engine version”, use “PostgreSQL 10.3-R1”. Use a “db.t2.micro” instance. Leave the storage at the 20 GiB default. Set the DB instance identifier to “namestore”, the “Master username” to “gnunet1” and the “Master password” to “password1”. On the next screen, select the “dht” VPC. Use “gnunet” for the database name. Set the backup to 0 days (no backup). Leave the other settings unchanged.

A.1.3 Preparing EC2

Now go via “Services” to the “EC2” dashboard. Click on “Security Groups”. You should see two groups, the “default” group for your VPC and the “rds-launch-wizard”. Set the name of the “default” group to “dht” and the “rds-launch-wizard” to “db”. Select the “dht” security group, and edit the “inbound” rules to allow TCP port 1080 and 2087 from anywhere, and SSH from your own system, and DNS (UDP) from anywhere. Select the “db” security group, and edit the “inbound” rules to allow “Postgres” from the “dht” security group.

Next, to to “Key Pairs” and “Create Key Pair”. Follow the instructions and download your SSH private key. Store it in a safe location. You will need to pass the respective filename to `ssh` commands using the `-i` option when logging into your instances.

A.1.4 Launching DHT nodes

In the “EC2” dashboard, select “Instances” and then “Launch Instance”. Select the “GNUnet DHT” community AMI[1] and use a “t2.micro” instance. Select “Next: Configure Instance Details”. You can now specify the number of DHT nodes to launch. For “Network”, use the “dht” VPC. Selec the “public” subnet. Leave the other settings unchanged.

Do not change anything on the “Add Storage” or “Add Tags” screens.

On the “Configure Security Group” screen, select “Select and existing security group” and then select the “default” group of your VPC.

After selecting “launch”, pick the key pair you created earlier from the list.

Now go to “Instances” and select the instance you just created. Change the name to “DHT-X” (vary X based on the number of instances you created). Lookup the “IPv4 public IP” in the table and use the value in `$IP` below. Then type:

```
$ ssh -i my-key ubuntu@$IP
```

to login to the system. There, use `sudo bash` to become “root”. Then enter

```
# ip addr add $IP dev eth0
```

to add the public IP address to the network interface of your system. Finally, use

```
# su - gnunet
$ gnunet-arm -s
```

to launch the peer.

[1]Owner should be 756606553745, available in Ohio, Frankfurt and Seoul

A.1.5 Launching the DNS importer

In the “EC2” dashboard, select “Instances” and then “Launch Instance”. Select the “GNUnet-FR-importer” community AMI[2] and use a “t2.medium” instance. Select “Next: Configure Instance Details”. using the “dht” VPC. You should probably only launch a single instance. For “Network”, use the “dht” VPC and for “Subnet” the “public subnet”. Enable auto-assignment of public IPv4 and IPv6 addresses. Leave the other settings unchanged.

Do not change anything on the “Add Storage” or “Add Tags” screens.

On the “Configure Security Group” screen, select “Select and existing security group” and then select both the “default” and “rds-launch-wizard” groups.

After selecting “launch”, pick the key pair you created earlier from the list.

Now go to “Instances” and select the instance you just created. Change the name to “DNS-importer”. Lookup the “IPv4 public IP” in the table and use

```
$ ssh -i my-key ubuntu@$IP
```

to login to the system. There, use `sudo bash` to become “root”. Then enter

```
# ip addr add $IP dev eth0
```

to add the public IP address to the network interface of your system.

Next, use `psql` to verify that you can connect to the RDS. For this, look in the RDS console for the DNS name under “endpoint”. It should be something like “namestore.XXXXX.YYYY.rds.amazonaws.com”. Using this value for “$HOST”, type

```
# screen
# su - gnunet
$ psql postgres://gnunet1:password1@$HOST/gnunet
```

If this succeeds, configure GNUnet to use this database and launch the peer using:

```
$ gnunet-config -s namestore-postgres -o config \
  -V "postgres://gnunet1:password1@$HOST/gnunet"
$ gnunet-arm -s
```

Use “ctrl-a c” to create another root shell. In it, download the DNS zone from AFNIC and prepare the GNS zones:

```
# su - gnunet
$ ./convert.sh YYYYMM # replace by year and month after 15th
$ gnunet-identity C fr      # create TLDs we wish to support
$ gnunet-identity C asso.fr # the FR zone includes these also
$ gnunet-identity C com.fr
$ gnunet-identity C gouv.fr
$ gnunet-identity C nom.fr
```

[2]Only available in Ohio.

```
$ gnunet-identity C presse.fr
$ gnunet-identity C tm.fr
$ gnunet-identity -d # display result, note the public keys
```

The value of the public key listed under ".fr" must be added in the GNUnet configuration file in section "[gns]" under the key ".fr". Now we need to link the ".fr" GNS zone to the other GNS zones we created, and then import the DNS zone into GNS:[3]

```
$ gnunet-namestore -a -z fr -t PKEY -n asso -e never -p \
  -V NB8B975VVYRMA9E4TR7KYQRAG09HTAP9X06M0BQ2TKP1V7WBTC80
$ gnunet-namestore -a -z fr -t PKEY -n presse -e never -p \
  -V FNGGR7EWTV6DQ7EK5JP6RJZT1TCYRSFV3WWHBDF6K7365GYJ1REG
$ gnunet-namestore -a -z fr -t PKEY -n gouv -e never -p \
  -V JXY38SC0C5M4XF1ZX9S5N2X4KKRS6RPT35M5YG8AGGWDCY4B3ZC0
$ gnunet-namestore -a -z fr -t PKEY -n com -e never -p \
  -V ABY9KZZRF42M1SCHP4SKZVCN4M4BYNZTEY198RZ5T31WYVX431PG
$ gnunet-namestore -a -z fr -t PKEY -n nom -e never -p \
  -V Q57SMV4ES4JAXPHDE6T9JABSA7W5S841EA30DJ6NG2RW3QDBWH7G
$ gnunet-namestore -a -z fr -t PKEY -n tm -e never -p \
  -V FKWHH6AFRXQ7S0DNBWG0XFDG2S2YVR66AS38R6S3BTCYGQXD0T10
$ gnunet-zoneimport -m "14 days" \
  -s 12000000 194.0.9.1 < fr-names-YYYYMM.txt
```

Finally, you can create a third shell with "CTRL-a c" and watch the progress using:

```
# su - gnunet
$ gnunet-statistics -s zoneimport
$ gnunet-statistics -s namestore
$ gnunet-statistics -s zonemaster
$ gnunet-statistics -s zonemaster-mon
```

Logout using "ctrl-a d" followed by pressing "ctrl d" twice.

A.1.6 End-user setup

The public keys shown by "gnunet-identity -d" must be added to the configuration files of users that are to use this peer's version of ".fr" using commands like:

```
$ gnunet-config -s gns -o .fr -V $ZONEKEY # do not use this here
```

For our usability experiments, we provided users where the key material for ".fr" was already present on their system, as we will generally ship such key material with the distribution, similar to how modern operating systems include a default set of certificate authorities.

[3]Note that below we provide the public keys we used in our experiments.

A.2 Usability Study: DNS vs. GNS

A.2.1 Instructions to participants

The instructions given to the participants were the following:

> *Welcome to our private browsing study!*
>
> *In this study, you will be given access to two computers running a browser. We ask you to use and* **compare** *both systems for 4-5 minutes each. Note that the systems are labeled with* **PC A** *and* **PC B**, *you do not have to use them in this order.*
>
> *You should use the systems as you would usually use a browser to surf the Internet. However, please browse* some *French Web sites, such as `lemonde.fr`, `liberation.fr` or `lefigaro.fr`.*
>
> *After browsing using* **both** *computers, please go to `https://surveys.bfh.ch/index.php/851916?lang=en` to complete the survey.*
>
> *Thank you for your participation.*

A.2.2 Self-reported participant demographics

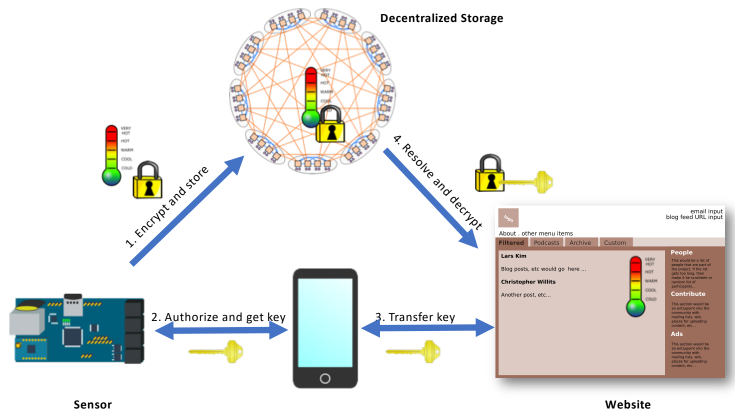
Out of 33 participants, we had 25 men, 3 women and 5 who refused to answer. The self-reported age groups of the participants were 11 in 18–24, 14 in 25–34, 33 in 35–44, 1 in 45–54, 2 in 55–64 and 1 older than 65. In terms of education, 1 reported no degree, 10 reported a high school graduate degree, 6 professional training, 1 secondary school, 3 a Bachelor, 8 a Master, and 2 a Doctorate.

A.3 Usability study: IoT

A.3.1 Instructions to participants

The instructions given to the participants were the following:

> Welcome to our Internet-of-Things (IoT) Authentication Study!
>
> In this study you will use a new innovative decentralized service "re:claim" IoT. It allows you to securely share access to your Things sensor data. You can authorize a Web site that requests sensor data by using the re:claim App.
>
> The sensor data of your Thing is encrypted and stored in a decentralized network. When you choose to authorize a webpage to access the data, the service will be given the key to decrypt:



The advantage of using re:claim IoT is that you do not have to provide direct connectivity to your Thing. Furthermore, if the sensor data ever changes, the updated data will then be automatically shared with all of the services that you previously authorized.

In this study we want to evaluate the usability of such an authorization process. You are asked to perform an authorization for our demo webpage. The webpage will ask you to authorize it to access a specific set of sensor data: Temperature, altitude and atmospheric pressure. To do so, please follow the steps on the next page.

> 1. Access `http://localhost:4200` in your browser
> 2. **Open the App "re:claim"** on your phone.
> 3. **Tap on the "Scan QR Code"** to start the authorization process.
> 4. **Scan the QR code** displayed on the webpage.
> 5. **Tap on the "re:claim" icon** at the right side of the "Sensors Central" item in the App.
> 6. **Align the displayed icon on the phone with the icon on the sensor** AND when prompted **tap the screen**.
>
> The webpage should now display receive the authorization and display the sensor data.
>
> After completing the experiment, please complete a short survey:
>
> `https://surveys.bfh.ch/index.php/898722?lang=en`
>
> Thank you for your participation!

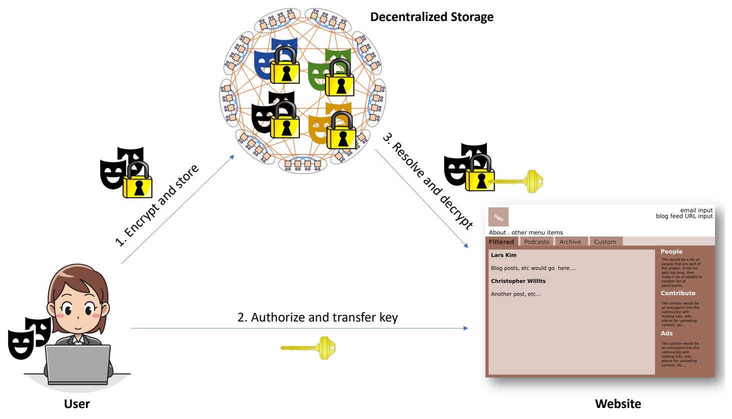
A.3.2 Self-reported participant demographics

A.4 Usability study: re:claim

A.4.1 Instructions to participants

The instructions given to the participants were the following:

> Welcome to our User Authentication Study!
>
> In this study you will use a new innovative decentralized authentication service "re:claim". The service functions just like other popular so-called "Social Logins" such as "Login with Google" or "Login with Facebook". It allows you to create profiles using nicknames (pseudonyms). You can log in and log out and the information of your profile (attributes) will stay and be provided to the Web site.
>
> However, the underpinnings of the re:claim service are different from login services like Google or Facebook in that it is completely under the users – your – control. Your pseudonymous identities and associated attributes of your profiles are only stored on your computer. When you choose to authorize a webpage to access your attributes, it will be encrypted and stored in a decentralized network so the service can access it even when you are offline:



The advantage of using this service is that you do not have to re-enter attributes for every service. Furthermore, if your attributes ever change, you can change them on your computer. The updated attributes will then be automatically shared with all of the services that you previously authorized just like with conventional profiles. Finally, you can choose to maintain many pseudonyms instead of being tied to one identity.

In this study you must perform such an authorization for our demo webpage at `https://example.io`. The webpage will ask you to login and authorize it to access a specific set of identity attributes: Your full name and email address. To do so, please follow the steps on the next page.

1. Access `https://example.io` in your browser
2. **Click on "re:claim"** to login. At this point you will be redirected to your local identity service
3. **Click on "Add identity"** to add a pseudonym.
4. **Enter a username** and click "Save".
5. Before you can use the pseudonym, you need to fill in the attributes requested by the webpage. The attributes used in this example are "email" and "full_name".
6. **Add an email address** for "email", e.g. "john@doe.com".
7. **Add a name** for "full_name", e.g. "John Doe".
8. **Click on "Save"**.
9. You may now create additional pseudonyms or attributes.

10. Finally, **click "Authorize"** to select which pseudonym to use and finish the login.

Your browser should now be redirected back to the webpage `https://example.io` and you should be greeted with your entered name and email address.

After completing the experiment, please complete a short survey:

`https://surveys.bfh.ch/index.php/617286?lang=en`

Thank you for your participation!

A.4.2 Self-reported participant demographics

A.5 Usability study: Accident insurance claims in Switzerland

A.5.1 Instructions to participants

Usability Test

The goal is to test the new **process for retrieving the accident claim number** (Schadensfallnummer) needed by the accounting process of health professionals. This number is issued by the accident insurance of the patients' employer some days after an accident has occured. Normally, the number is transmitted by phone to the health professionals after the visit.

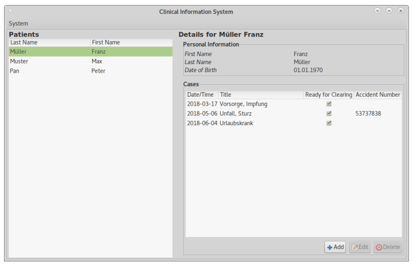
Initial situation

You are a health professional and entering patient's data in your clinical information system (CIS). The next patient is an **emergency** treatment after a bicycle accident.
You know that for patients with an accident a special accounting process requiring an **accident claim number** is in place.

Test scenario

Step 1
In your CIS, you are looking for the patient's information to open a new case:

- Name of Patient: Franz Müller



Patients

Last Name	First Name
Müller	Franz
Muster	Max
Pan	Peter

Details for Müller Franz

Personal Information

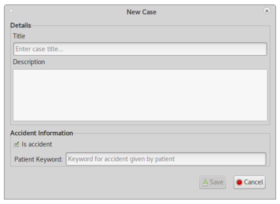
First Name	Franz
Last Name	Müller
Date of Birth	01.01.1970

Cases

Date/Time	Title	Ready for Clearing	Accident Number
2018-03-17	Vorsorge, Impfung	✓	
2018-05-06	Unfall, Sturz	✓	53737838
2018-06-04	Urlaubskrank	✓	

Step 2
Open a new case (using the "Add"–Button) and enter required information:

- Freely chose a title and description
- Mark the case as accident: Check the checkbox "Is Accident"
- Enter the keyword for the accident you have obtained from the patient: "bikeAccident"



Details
Title: Enter case title...
Description:

Accident Information
✓ Is accident
Patient Keyword: Keyword for accident given by patient

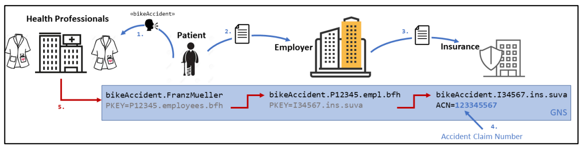
Step 3
Verify if the accident claim number can be retrieved by your CIS.

If not, the accident is not yet handled by the insurance company and you have normally have to wait a couple of days.

Step 4
Imagine that now the treatment of patient takes place….

Step 5
The test responsible will now play the role of the accident insurance and register the new case on its side. As result, the accident claim number for Franz Müller will be issued.

What has happened? The accident insurance stores the accident claim number for Franz Müller in its GNS zone[1]. By the linking of the different entries, now all actors knowing the keyword are able to retrieve this number.



.

Step 6
Imagine, you open your CIS again a couple of days later…

Try to find out, if the accident claim number for Franz Müller could have been retrieved by your CIS. That means the accident insurance company has accepted the patient's case and you have all the information you need for the accounting process.
Note, that without the new process, you would have to call the patient – maybe several times – to get this number.

Step 7

Fill out the questionnaire. Please go to: https://surveys.bfh.ch/index.php/995581?lang=en.

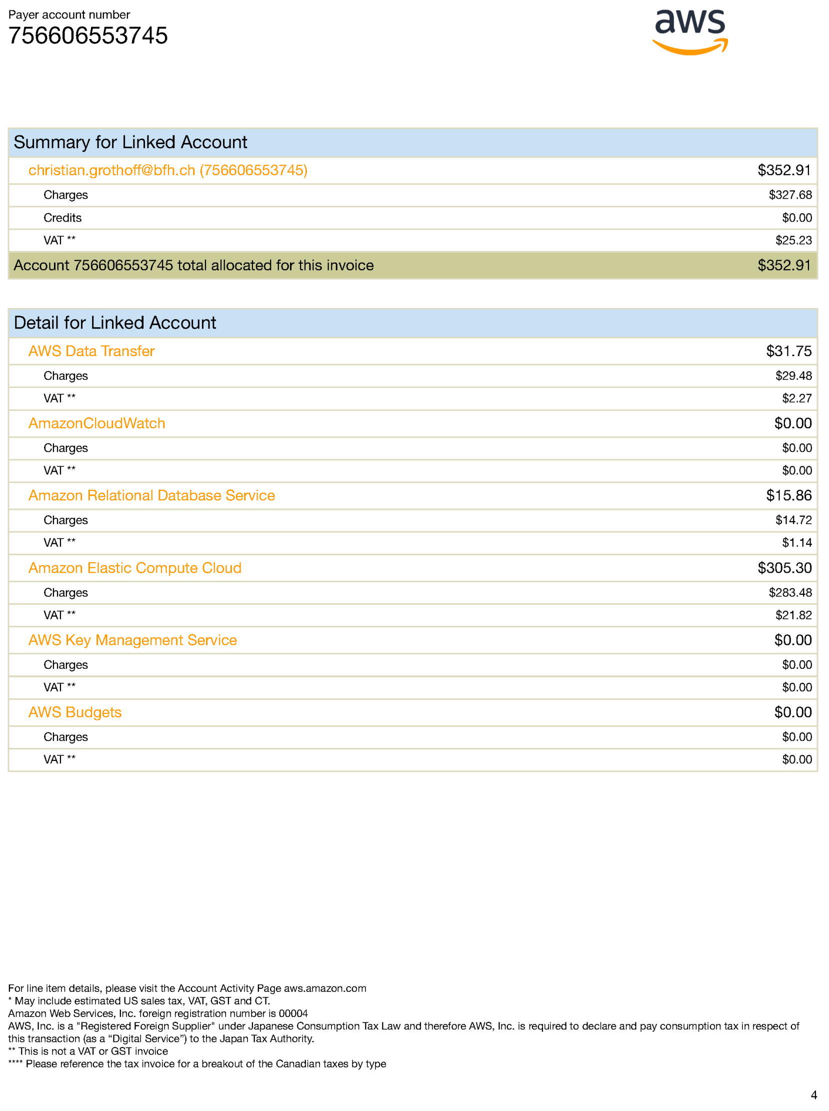
Thank you very much!!

[1] The Gnu Name System (GNS) is a privacy-preserving security protocol for authentication. It provides a 2-factor authentication based on a password and a secret key.

A.5.2 Self-reported participant demographics

Out of 44 participants, we had 24 women, 19 men and 1 who refused to answer. The self-reported age groups of the participants were 19 in 18–24, 16 in 25–34, 4 in 35–44, 3 in 45–54, 2 in 55–64. In terms of education, 1 reported a secondary school degree, 11 reported a high school graduate degree, 8 professional training, 14 a Bachelor, 6 a Master, and 4 a Doctorate. The professional background was divided in 4 groups: Computer Science 15 persons, Health informatics 5 persons, Health professionals 20 persons, Others 4 persons.

A.6 Cost study: EC2 billing



Payer account number
756606553745

Summary for Linked Account	
christian.grothoff@bfh.ch (756606553745)	$352.91
Charges	$327.68
Credits	$0.00
VAT **	$25.23
Account 756606553745 total allocated for this invoice	$352.91

Detail for Linked Account	
AWS Data Transfer	$31.75
Charges	$29.48
VAT **	$2.27
AmazonCloudWatch	$0.00
Charges	$0.00
VAT **	$0.00
Amazon Relational Database Service	$15.86
Charges	$14.72
VAT **	$1.14
Amazon Elastic Compute Cloud	$305.30
Charges	$283.48
VAT **	$21.82
AWS Key Management Service	$0.00
Charges	$0.00
VAT **	$0.00
AWS Budgets	$0.00
Charges	$0.00
VAT **	$0.00

For line item details, please visit the Account Activity Page aws.amazon.com
* May include estimated US sales tax, VAT, GST and CT.
Amazon Web Services, Inc. foreign registration number is 00004
AWS, Inc. is a "Registered Foreign Supplier" under Japanese Consumption Tax Law and therefore AWS, Inc. is required to declare and pay consumption tax in respect of this transaction (as a "Digital Service") to the Japan Tax Authority.
** This is not a VAT or GST invoice
**** Please reference the tax invoice for a breakout of the Canadian taxes by type

4

Figure A.1: Amazon invoice for a month of operating the infrastructure for publishing 5 million DNS records in 24 DHT peers (from two import systems iterating over the zone every 48h).

Bibliography

[1] https://measuringu.com/sus/, August 2018.

[2] Furkan Alaca and Paul C. van Oorscht. Comparative analysis and framework evaluating web single sign-on systems. https://arxiv.org/pdf/1805.00094.pdf, May 2018.

[3] R. Barnes. Use Cases and Requirements for DNS-Based Authentication of Named Entities (DANE). RFC 6394 (Informational), October 2011.

[4] Daniel J Bernstein and Tanja Lange. Post-quantum cryptography-dealing with the fallout of physics success. *IACR Cryptology ePrint Archive*, 2017:314, 2017.

[5] Dwaine Clarke, Jean-Emile Elien, Carl Ellison, Matt Fredette, Alexander Morcos, and Ronald L Rivest. Certificate chain discovery in spki/sdsi. *Journal of Computer security*, 9(4):285–322, 2001.

[6] Henri Cohen, Gerhard Frey, Roberto Avanzi, Christophe Doche, Tanja Lange, Kim Nguyen, and Frederik Vercauteren. *Handbook of elliptic and hyperelliptic curve cryptography.* CRC press, 2005.

[7] A. Deacon and R. Hurst. The Lightweight Online Certificate Status Protocol (OCSP) Profile for High-Volume Environments. RFC 5019 (Proposed Standard), September 2007.

[8] Bundesversammlung der Schweizerischen Eidgenossenschaft. Bundesgesetz über die unfallversicherung (uvg). https://www.admin.ch/opc/de/classified-compilation/19810038/index.html, September 2017.

[9] David Eppstein, Michael T. Goodrich, Frank Uyeda, and George Varghese. What's the difference?: efficient set reconciliation without prior context. In *Proceedings of the ACM SIGCOMM 2011 conference*, SIGCOMM '11, page 218–229, New York, NY, USA, 2011. ACM, ACM.

[10] Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation), May 2016.

[11] Nathan Evans. *Methods for Secure Decentralized Routing in Open Networks*. PhD thesis, Technische Universität München, 2011.

[12] Nathan Evans and Christian Grothoff. R5n: Randomized recursive routing for restricted-route networks. In *5th International Conference on Network and System Security*, pages 316–321, Milan, Italy, 2011. IEEE.

[13] Nathan S. Evans, Bart Polot, and Christian Grothoff. Efficient and secure decentralized network size estimation. In *11th International IFIP TC 6 Networking Conference*, volume 7289 of *LNCS*, pages 304–317. IFIP, Springer Verlag, 2012.

[14] Free Software Foundation. The GNU C Library - System Databases and Name Service Switch. `http://goo.gl/gQYOw`.

[15] Sebastian Friebe, Ingo Sobik, and Martina Zitterbart. Decentid: Decentralized and privacy-preserving identity storage system using smart contracts. In *17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pages 37–42, 2018.

[16] Hayato Fujii and Diego F. Aranha. Curve25519 for the cortex-m4 and beyond. In *LatinCrypt*, 2017.

[17] P. Hoffman and J. Schlyter. The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA. RFC 6698 (Proposed Standard), August 2012. Updated by RFCs 7218, 7671.

[18] Google Inc. Certificate transparency. `https://www.certificate-transparency.org/`, 2018.

[19] H. Krawczyk and P. Eronen. HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869 (Informational), May 2010.

[20] Adam Langley. Revocation checking and chrome's crl. http://www.imperialviolet.org/2012/02/05/crlsets.html, February 2012.

[21] B. Laurie, G. Sisson, R. Arends, and D. Blacka. DNS Security (DNSSEC) Hashed Authenticated Denial of Existence. RFC 5155 (Proposed Standard), March 2008. Updated by RFCs 6840, 6944.

[22] Adam J Lee. *Towards practical and secure decentralized attribute-based authorization systems*. ProQuest, 2008.

[23] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, and L. Jones. SOCKS Protocol Version 5. RFC 1928 (Proposed Standard), March 1996.

[24] Ninghui Li, Benjamin N Grosof, and Joan Feigenbaum. Delegation logic: A logic-based approach to distributed authorization. *ACM Transactions on Information and System Security (TISSEC)*, 6(1):128–171, 2003.

[25] Ninghui Li and John C Mitchell. Rt: A role-based trust-management framework. In *DARPA Information Survivability Conference and Exposition, 2003. Proceedings*, volume 1, pages 201–212. IEEE, 2003.

[26] Ninghui Li, John C Mitchell, and William H Winsborough. Design of a role-based trust-management framework. In *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on*, pages 114–130. IEEE, 2002.

[27] Netcraft Ltd. How certificate revocation (doesn't) work in practice. Blog entry at Netcraft blog: `http://news.netcraft.com/archives/2013/05/13/how-certificate-revocation-doesnt-work-in-practice.html`, May 2013.

[28] P.V. Mockapetris. Domain names - implementation and specification. RFC 1035 (Internet Standard), November 1987. Updated by RFCs 1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2181, 2137, 2308, 2535, 2673, 2845, 3425, 3658, 4033, 4034, 4035, 4343, 5936, 5966, 6604, 7766.

[29] Council of Europe. Convention for the protection of human rights and fundamental freedoms (european convention on human rights, as amended) (echr). `https://www.echr.coe.int/Documents/Convention_ENG.pdf`, 1950.

[30] Joanne Orlando. How teens use fake instagram accounts to relieve the pressure of perfection. `http://theconversation.com/how-teens-use-fake-instagram-accounts-to-relieve-the-pressure-of-perfection-92105`, March 2018.

[31] Zhonghong Ou, Erkki Harjula, Otso Kassinen, and Mika Ylianttila. Performance evaluation of a kademlia-based communication-oriented p2p system under churn. *Comput. Netw.*, 54:689–705, April 2010.

[32] Ginger Perng, Michael K. Reiter, and Chenxi Wang. Censorship resistance revisited. In *Information Hiding, 7th International Workshop, IH 2005, Barcelona, Spain, June 6-8, 2005, Revised Selected Papers*, pages 62–76, 2005.

[33] Bart Polot and Christian Grothoff. Cadet: Confidential ad-hoc decentralized end-to-end transport. In *IEEE/IFIP Annual Mediterranean Ad Hoc Networking Workshop (MedHocNet)*, 2014.

[34] Samuel Pulfer. Tld analyse – analyse über die zeitliche veränderung der records einer tld am beispiel schweden. Available upon request, June 2018.

[35] E. Rescorla. The transport layer security (tls) protocol version 1.3. draft-ietf-tls-tls13-latest, March 2018.

[36] Ahmad Sabouri, Ioannis Krontiris, and Kai Rannenberg. Attribute-based credentials for trust (abc4trust). In Simone Fischer-Hübner, Sokratis Katsikas, and Gerald Quirchmayr, editors, *Trust, Privacy and Security in Digital Business*, pages 218–219, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[37] M. Schanzenbach, C. Banse, and J. Schütte. Practical decentralized attribute-based delegation using secure name systems. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pages 244–251, Aug 2018.

[38] M. Schanzenbach, G. Bramm, and J. Schütte. reclaimid: Secure, self-sovereign identities using name systems and attribute-based encryption. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pages 946–957, Aug 2018.

[39] Marc Stiegler. An introduction to petname systems. `http://www.skyhunter.com/marcs/petnames/IntroPetNames.html`, February 2005.

[40] Jose M. Such, Agustin Espinosa, Ana Garcia-Fornes, and Vicent Botti. Partial identities as a foundation for trust and reputation. *Engineering Applications of Artificial Intelligence*, 24(7):1128–1136, 2011. Infrastructures and Tools for Multiagent Systems.

[41] S. Thomson, C. Huitema, V. Ksinant, and M. Souissi. DNS Extensions to Support IP Version 6. RFC 3596 (Draft Standard), October 2003.

[42] Gabor X Toth. Design of a social messaging system using stateful multicast. Master's, University of Amsterdam, Amsterdam, 2013.

[43] Matthias Wachs, Martin Schanzenbach, and Christian Grothoff. A censorship-resistant, privacy-enhancing and fully decentralized name system. In *13th International Conference on Cryptology and Network Security (CANS 2014)*, pages 127–142, 2014.