TECHNISCHE UNIVERSITÄT MÜNCHEN

LEHRSTUHL FÜR SICHERHEIT IN DER INFORMATIK

Fakultät für Informatik

Towards Self-sovereign, Decentralized Personal Data Sharing and Identity Management

Martin Schanzenbach

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:		Prof. Jens Grossklags, Ph.D.
Prüfende/-r der Dissertation:	1.	Prof. Dr. Claudia Eckert
	2.	Prof. Dr.-Ing. Georg Carle

Die Dissertation wurde am 18.05.2020 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 23.11.2020 angenommen.

Acknowledgments

I would like to thank my supervisor Prof. Dr. Claudia Eckert for giving me the opportunity to conduct research in this highly relevant and interesting area. She provided me with valuable insights, guidance and room for exploration throughout this project. I would also like to thank my second examiner Prof. Dr.-Ing. Georg Carle.

Further, I wish to thank my colleagues at Fraunhofer AISEC who provided me with an inspiring, productive and enjoyable work atmosphere in the past years.

I am especially grateful to thank Dr. Julian Schütte and Christian Banse for their patient guidance and persistent encouragement giving me the perseverance to finish this dissertation. This extends to Prof. Dr. Christian Grothoff for his insightful comments and advice.

I thank my co-authors of the publications referenced in this thesis:
Ava Ahadipour for her work on surveying authorization in distributed systems.
Georg Bramm for his support and expertise in attribute-based encryption.
Bernd Fix for his ideas on improving proof-of-work calculations.
Thomas Kilian for his work on zero-knowledge credentials as part of his Bachelor thesis.

Finally, I would like to thank my parents and wife for their constant support and love.

Abstract

Today, identity management is a key element for commercial and private services on the Internet. Over the past decade, digital identities evolved away from decentralized, pseudonymous, user-controlled personas towards centralized, unabiguous identities managed at and provided through service providers. This development was sparked by the requirement of real identities in the context of electronic commerce. However, it was particularly fuelled later by the emergence of social media and the possibilities it provides to people in order to establish social connections. The following centralization of identities at a handful of service providers significantly improved usability and reliability of identity services. Those benefits come at the expense of other, arguably equally important areas. For users, it is privacy and the permanent threat of being tracked and analyzed. For service providers, it is liability and the risk of facing significant punishment caused by strict privacy regulations which try to counteract the former.

In this thesis, we investigate state-of-the-art approaches to modern identity management. We take a look at existing standards and recent research in order to understand the status quo and how it can be improved. As a result from our research, we present the following contributions: In order to allow users to reclaim control over their identities and personal data, we propose a design for a decentralized, self-sovereign directory service. This service allows users to share personal data with services without the need of a trusted third party. Unlike existing research in this area, we propose mechanisms which allow users to efficiently enforce access control on their data. Further, we investigate how trust can be established in user-managed, self-sovereign identities. We propose a trust establishment mechanism through the use of secure name systems. It allows users and organizations to establish trust relationships and identity assertions without the need of centralized public key infrastructures (PKIs). Additionally, we show how recent advancements in the area of non-interactive zero-knowledge (NIZK) protocols can be leveraged in order to create privacy-preserving attribute-based credentials (PP-ABCs) suitable for use in self-sovereign identity systems including our proposed directory service. We provide proof of concept implementations of our designs and evaluate them to show that they are suitable for practical applications.

Zusammenfassung

Identitätsmanagement ist ein Schlüsselelement von heutigen, Internet-basierten Diensten. Über die letzte Dekade entwickelten sich digitale Identitäten weg von dezentralisierten, nutzerkontrollierten und pseudonymen Personas hin zu eindeutigen Identitäten welche zentral bei Dienstleistern verwaltet und gespeichert werden. Diese Entwicklung wurde zunächst angestoßen durch Anforderungen bezüglich Identitätsfeststellungen im Rahmen des elektronischen Handels. Nochmals befeuert wurde dies durch das Aufkommen der sozialen Medien und den damit verbundenen Möglichkeiten für Menschen soziale Verbindungen aufzubauen. Die darauf folgende Zentralisierung von Nutzeridentitäten in einer Handvoll von kommerziellen Dienstleistern verbesserte die Nutzbarkeit und Zuverlässigkeit dieser Dienste enorm. Diese Vorteile entstanden jedoch auf Kosten anderer, aber gleichermaßen wichtigen Aspekten. Für Benutzer betrifft dies vor allem die Privatssphäre und die Gefahr einem permanenten Tracking und Analyse ausgesetzt zu sein. Für Diensteanbieter ist es primär das durch strengere Datenschutzgesetze gestiegene Haftungsrisiko.

In dieser Dissertation untersuchen wir den Stand der Technik des mordernen Identitätsmanagements. Wir werfen einen Blick auf existierende Standards und aktuelle Ergebnisse aus der Forschung um so den Status Quo zu verstehen und Verbesserungen zu erarbeiten. Im Rahmen der Arbeit präsentieren wir folgende Forschungsergebnisse: Um es Nutzern zu ermöglichen die Kontrolle über ihre Identitäten und persönlichen Daten zurückzuerlangen schlagen wir den Einsatz eines dezentralen Verzeichnisdienstes vor. Ein solcher Dienst erlaubt es Nutzern ihre persönlichen Daten zu teilen ohne dafür einen zentralen Identitätsdienst zu benötigen. Hierbei schlagen wir Mechanismen vor, welche es Nutzern erlauben effizient und selbstbestimmt den Zugriff auf ihre Daten zu Verwalten. Weiterhin untersuchen wir wie Vertrauen in diese durch die Nutzer selbst verwalteten Identitäten hergestellt werden kann. Dazu präsentieren wir einen Ansatz zur Vertauensbildung durch den Einsatz von sicheren Namenssystemen. Dies erlaubt es Nutzern und Organisationen Vertrauensbeziehungen aufzubauen ohne hierbei auf zentralisierte Dienste Dritter oder einer zentralen Public-Key-Infrastruktur zurückgreifen zu müssen. Wir untersuchen, wie neue Erkenntnisse im dem Bereich der Zero-Knowledge-Verfahren genutzt werden können um datenschutzfreundliche Identitätsattribute auszustellen welche das vorgestellte System ergänzen können. Schließlich zeigen wir konzeptuelle Implementierungen unserer Konzepte und zugehörige Evaluierungen um die Praxistauglichkeit unseres Systems testen.

Contents

CHAPTER 1

Introduction

In this thesis, we present methods which enable and support the use of *decentralized, self-sovereign identity systems*. Our core contribution is a decentralized attribute sharing service which is enhanced using practical and privacy-preserving trust establishment techniques.

We argue that moving away from commercially driven, centralized identity services is the only way for users to reclaim authority and control over their digital personas. We further argue that in order to achieve this goal, the development of user-empowering, decentralized technologies for identity management is essential. This encompasses both control over the personalized embodiment of the persona and the power to control access to it.

In the following, we give a comprehensive motivation for our research and establish the context of this thesis. Further, we formalize our research questions and give an overview over our core contributions.

1.1. Motivation

Today, digital data is called the "new oil" [140]. This not only includes but arguably specifically denotes personal and private information of humans. Personal information, and whatever can be extrapolated from it, is a most profitable data category. Comparisons between data and oil fall into place if we look at how data fuels the business models of some of the most successful technology companies today, with Google and Facebook as prime examples. But there is another aspect to this comparison which is particularly valid with respect to personal and private data: Oil spills. Like oil spills, personal information is notoriously difficult – if not nearly impossible – to clean up. Once exposed to the environment, it becomes inseparable and untraceable in the vast lake of data which is the Internet. Consequently, the management of our digital identities is an important aspect in modern life. Interactions between users are increasingly occurring via digital means and through social media as opposed to real life [82]. This is also true for interactions between users and organizations such as companies or the state.

Recently, identity service providers which emerged from popular social networking services claim sovereignty over the users' identities. Often, the respective service providers even require the use of a "real" name. We often forget that in the beginning of the Internet, anonymity and pseudonymity were considered core values and enticements by its users [90]. Internet identities were considered disposable and exchangeable. With the emergence of social media, however, the revelation and nurturing of ones online identity became the norm. Conversely, such behavior is unnatural and inconsistent with psychological and philosophical observations. Current identity management does not take into account that in the real world identities are used like masks. Human identities commonly vary with respect to traits or discriminability depending on the social situation and context [31, 35, 126]. This has always been a key in social interactions to allow a person integrate into its social peer group to, for example, avoid conflict. Maybe this is even one of the reasons why social platforms today are a cesspool of heated arguments and radicalization of opinions.

We argue that modern identity management must not only provide reliable means for verifying the authenticity of identities and enable users to share their personal information with other parties. Identity management must take into account that our digital identities and the personal information they are comprised of are not only shaped directly by ourselves, but also indirectly by the third parties we decide to share identity data with. It has been argued that third parties establish behavioral profiles which are just as much part of our digital identities as the data we choose to share [136]. They allow to track users and learn what websites they visit and what content they prefer. This concept is pushed further through the use of algorithms and even artificial intelligence which allow data brokers to generate additional information based on the shared identity data in combination with user behavior. Through this, personal circumstances such as medical conditions or financial situations can easily be extrapolated. The user is limited to a single lever in this context: What he decides to share with the outside world and what not. The rest is not under his control.

Looking at it from another perspective, the state of modern identity management does not look any better: Initially, any service provider from instant messaging to online shopping implemented its own user and identity management system. This resulted in a situation where each user created a multitude of online accounts. While this allowed users to create alter egos, the number of duplicated user accounts and stale data that had to be managed by service providers exploded. At the same time, usability suffered for users when faced with managing a high number accounts. This prompted users to reuse the same credentials across services and accounts – a cardinal security sin. Besides the technical and organizational challenges which arose from this situation, legislation such as the General Data Protection Regulation (GDPR) [101] decreed by the European Union recently added another dimension to this development: Liability. Albeit only enforced hesitantly until now, service providers are in theory now faced with a sword of Damocles in the form of multiple millions of Euros or a percentage of annual revenue in possible fines. Compromised or stolen user data – whether through accidental leaks or malicious hacking attacks – may very well spell the financial doom of an enterprise. As a result, services are more than happy to outsource the management of identities to specialized service providers. At the same time, centralized identity provider services emerged from large social networks that claim authority over their users' digital personas. The most prominent examples are the identity services provided by Facebook and Google and the "Social Login" buttons which can be found on a significant number of websites. Due to the sheer number of user accounts registered at their respective services, the two are the de facto standard when it comes to outsourcing identities. Especially for smaller websites, such a service seems like a boon since it alleviates some of the liability concerns surrounding the legal situation. It also redeems the services from the correct implementation of security critical authentication mechanisms for its users, including password storage which is a common and fatal pitfall. However, we note that the dominant identity providers today do not actually take resposibility of the above in a legal sense whatsoever. This arguably puts a service provider in an even tighter spot: Less control over user's data while at the same time shouldering all of the legal resposibility.

1.2. Research questions

In the following, we state our research questions which are derived from and aim to address the above situation. We identify and single out four observations which we want to address in the context of this thesis:

Data as a liability As discussed above, most commercial services require user data for daily business operations, but this data is inherently a liability [75]. This is especially true in the current wake of strict data protection legislation and compliance laws such as the EU GDPR [101]. Hence, both technical and legal risks must be considered which likely requires a considerable mount of resources. In this situation, services turn to specialized Identity Provider (IdP) services instead of managing user data and identities themselves. Of course, those face the same types of liability risks which then become even more severe due to the increased number of users to manage. This development tends to induce consolidation of the respective IdP service offerings which directly leads to the next challenge.

Closed shop The technical and legal complexity, financial costs as well as the resulting risks are causes for a consolidation of the IdP market. Only large enterprises with the respective resources to handle the above remain and consequently the entry barrier for smaller competitors is high. The remaining providers manage and control over hundreds of millions of user accounts containing personal, potentially sensitive data. Studies such as Gigya's survey on the consumer identity provider landscape show that a service oligopoly of two service providers claim over 85% of the market [60]. Of course, this development is partly also caused by the popularity and general brand awareness that is associated with those two companies and their services. Another point is that those entities are not authorities for every aspect of users' identities. Information such as official name, postal address, citizenship or even email address are not necessarily attributes we would naturally expect a single identity provider to be the authoritative entity for. Nevertheless, for the two dominant service providers in the market inclusion of third party asserted attributes at the services is not possible at this time. We must conclude that the identity service provider landscape is currently not an even playing field with healthy competition or choice for end users.

If you build it, they will come Due to the consolidation tendency as discussed above large, centralized identity service providers have emerged. We argue that the high concentration of valuable, personal data attracts big data business interests [42, 9, 27], law enforcement covetousness and malicious actors alike. Further, as the services are normally free for the end user, the only business case of the mentioned IdPs often consists of analysis and marketing of personal data. This includes the user data itself as well as requests by third party services. The latter allows to track the user across Internet services. This conduct is combined with the questionable practice that users often have no choice but to agree that IdPs are allowed to analyze and market personal data as part of the terms of service. Hence, this setup of omniscient intermediaries is also a significant threat to the users' privacy and restricts users' ability to informational self-determination [101]. Users must completely trust the IdP with respect to protecting the integrity and confidentiality of their identity in their interest. Various breaches of large IdPs such as the ones at Yahoo that revealed up to 3 billion user records to the public [52, 145, 135] have shown that these expectations are hard to meet at times. Finally, IdPs such as Facebook – and for a long time also Google – enforce a "real-name policy" [29]. Denying pseudonymous identity can be considered to be in direct violation to the human right to be forgotten [101].

No separation of concerns IdPs are essentially sovereign authorities over certain data. They are capable of attesting the validity and correctness of specific statements regarding an identity. However, modern IdPs such as those discussed above conflate this function with the provisioning of this information to users and services. Implementations of the former requirement is realized through directory services such as X.500 or through cryptographic assertions such as X.509 certificates or a combination of both. The latter is traditionally covered by protocols such as OpenID Connect 1.0 (OIDC) or Security Assertion Markup Language (SAML).

The reasons for this conflation are deliberate: For one, identity information is usually most useful when asserted by a trusted authority. By having the attestation authority service serving user data directly, complex technical solutions including cryptographic signatures and digital certificates are not necessary. Hence, it is not only a low hanging fruit for an IdP to offer such services but also simplifies the verification of user data for third parties. However, even more significant is that it allows the identity service provider to cater to business models centered around data mining and targeted advertisements. The revenue opportunities in those areas are strong incentives that entice IdPs in this regard [42]. The consequence is that IdP services require end users to effectively yield their right to informational self-determination upon service registration.

In this thesis, we show how the challenges above can be addressed using technical means. We show that it is possible to design such a technical system in a way that is independent of centralized, third party infrastructure. At the same time, our approaches do not necessarily disrupt the traditional IdP business model which revolves around the validation and verification of identity information.

The following are our three research questions of this thesis:

- **Research Question 1**: How can we ensure the users right to informational self-determination regarding his digital identities?
- **Research Question 2**: How can we mitigate the liability concerns that arise with the management of identity data?
- **Research Question 3**: How can trust in identity attributes be established in self-sovereign identity systems?

The main body of this thesis contains a series of chapters in which we present our research methods and results. After each chapter, we summarize its content and how it relates to or answers one or more of the above research questions in the context of our initial motivation.

1.3. Organization

This thesis is organized in the following chapters: In Chapter 2, we give an overview over existing core concepts in the area of identity management which provide a baseline for us in order to tackle the above challenges. We present relevant scientific related work in Chapter 3. In Chapter 4, we present our main contribution: re:claimID, a decentralized directory service for decentralized identity management and personal data sharing. To complement our research in the area of decentralized directory services, we present approaches for decentralized trust establishment in Chapter 5. Finally, in Chapter 6, we conclude the thesis by giving a summary of our contributions and an outlook on future work. We present our scientific, peer-reviewed publications in Appendix A and discuss how they relate and contribute to our topic of research in the context of this thesis.

CHAPTER 2

Background

In this thesis, we present an approach to self-sovereign decentralized identity management. In order to understand our approach and the building blocks of identity management in general, we discuss relevant concepts that revolve around this topic. This includes a closer look at directory services, trust establishment as well as standardization efforts that currently exist. In particular standardization is an important factor in the design of decentralized systems related to personal data [96]. The revision and structuring of those concepts allows us to understand, build upon and extend them in order to address our identified research questions in the remainder of this thesis.

2.1. Directory services

Traditionally, identity information is stored in *directory services*. Essentially, directory services fill the role of yellow pages in information systems. In fact, the directory service Network Information Service (NIS) was originally called *Yellow Pages* (YP), in reference to the telephone directory. The umbrella term "directory service" includes widespread software such as the Lightweight Directory Access Protocol (LDAP) and Microsoft Active Directory (AD) which are both part of the X.500 protocol family. X.500 is the de facto standard when it comes to directory service protocols, but name systems such as the Domain Name System (DNS) or the NIS also belong in this same service category. In Figure 2.1, we present the various types of directory services, some of which are still in use today, others are considered legacy. We note that historically, name systems and the NIS precede X.500 directory services. The latter is considered a legacy directory service and is commonly replaced by LDAP or AD in modern IT infrastructures. The origin of one of our core contributions in this thesis comes from the idea to use name systems as identity directories in order to realize decentralized services which can be used for identity management and personal data sharing. In the following, we present past and present standards in directory services.

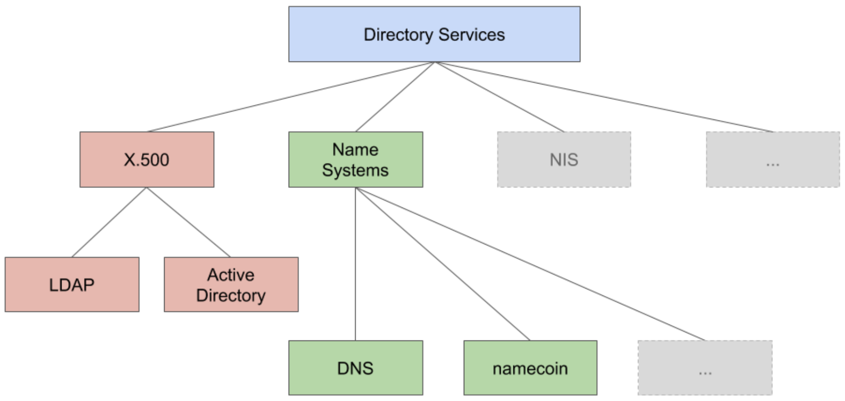


Figure 2.1.: Directory services.

2.1.1. Standards and specifications

X.500 is the umbrella term for a series of standards designed originally by the International Telecommunication Union (ITU). Of the many protocols defined in the specifications, the Directory Access Protocol (DAP) is the most relevant in the context of identity management. The DAP forms the basis for widespread implementations such as LDAP [148, 128]. X.500 directory services provide IT infrastructure administrators with the means to manage a Directory Information Tree (DIT). The DIT consists of entries identified by a unique *distinguished name* (DN) and a set of attributes.

In Figure 2.2, we modeled a DIT according to a fictional example scenario and user population. It contains both user and device entries which is indicated by the two distinct *organizational units* (OU) below the *domain components* (DCs) in the DIT hierarchy. The unique DN for the entry highlighted in gray is:

```
cn=john,ou=employees,ou=people,dc=mydomain,dc=org
```

Each entity managed in a DIT is identified by such a DN. The DN may be used to query attributes associated with a specific entity. Attributes may take on a variety of defined semantics, such as user attributes. Our example represents an entry commonly used for user attributes including a common name (“`cn`”) and a unique user ID (“`uid`”). Our example is by no means exhaustive or necessarily realistic but is only meant to illustrate the architecture and semantics of an X.500 DIT.

X.500 directories are usually served using central servers which are not accessible outside the local administrative domains. Consequently, X.500 directories are commonly used as services for identity management within organizational domains such as an internal company network. While public X.500 directory services are theoretically conceivable, their administration models are based on traditional role concepts: For example,

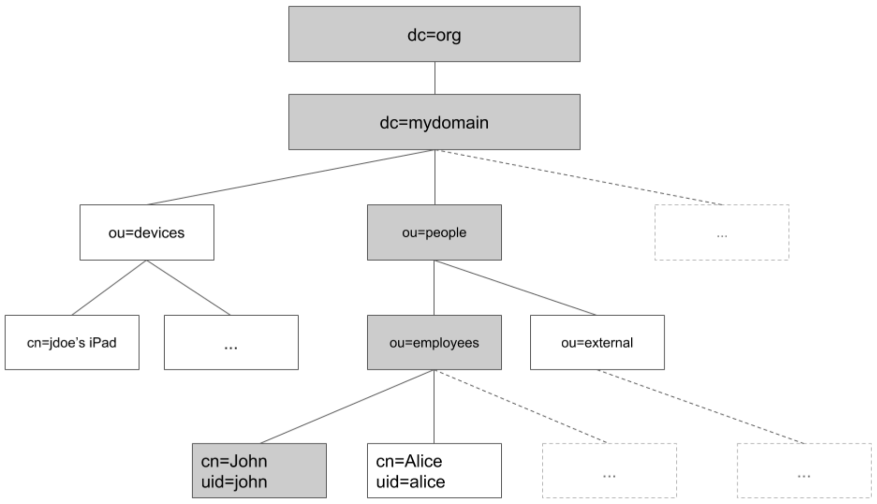


Figure 2.2.: Example of a DIT.

user self-service is rarely possible on the directory services themselves. Instead, the contents of directory services are provisioned through central IT departments of companies and organizations following well defined authorization hierarchies which are reflected in the X.500 directory itself. Consequently, deployment of X.500 is rarely seen on the Internet.

Security

Authenticity of data is guaranteed in X.500 directories as trust in the service is established either implicitly due to the locality of the service in the internal network or through traditional means such as TLS authentication. Further, the X.509 [28] standard defines how third party asserted attributes can be managed in X.500 directories. X.509 is more commonly known for being used in the TLS public key infrastructure (PKI) and defines a certificate format. Originally, it was designed and intended for authentication in the context of X.500 protocols, in particular as part of the DAPs. In such cases, the X.500 DIT is used to bind a DN of an entity to the public key information contained in the X.509 certificate. We dive a bit deeper into X.509 and its use in establishing trust as part of a PKI in Section 2.3. However, we can already see here a separation between assertions such as X.509 certificates and the delivery mechanism X.500. We criticize the lack of this separation in contemporary identity management in our introductory chapter.

Approaches to decentralization

In the wake of blockchain-based platforms and services, the need for directory services that can be integrated in highly decentralized architectures arises. Specifically, the hierarchical naming structure of traditional directory services based on X.500 or DNS require trusted third parties and thus stand in opposition to the idea of decentralized, "trustless"

```
{
  "@context": "https://w3id.org/did/v1",
  "id": "did:example:123456789abcdefghi",
  "publicKey": [{
    "id": "did:example:123456789abcdefghi#keys-1",
    "type": "RsaVerificationKey2018",
    "owner": "did:example:123456789abcdefghi",
    "publicKeyPem": "<PEMFILE>"
  }],
  "authentication": [{
    // this key can be used to authenticate as DID ...9938
    "type": "RsaSignatureAuthentication2018",
    "publicKey": "did:example:123456789abcdefghi#keys-1"
  }],
  "service": [{
    "type": "ExampleService",
    "serviceEndpoint": "https://example.com/endpoint/8377464"
  }]
}
```

Figure 2.3.: A minimal self-managed DID Document [105]

distributed ledgers [6]. From this requirement, a draft specification for Decentralized Identifier (DID) emerged [105].

A DID allows users to refer to identities in decentralized architectures by allowing entities to act as their own root of trust instead of relying on centralized trust anchors. This idea is reflected in the naming scheme for DIDs, which allows to define globally unique names based on UUIDs. As such, a DID stands in direct opposition to a DN commonly found in a DIT in X.500 DAPs. An example for a DID as given in [105] is `did:example:123456789abcdefghi`. Another concept called "DID Documents" are used to closer describe entities and attributes referenced by a DID. An example DID Document is illustrated in Figure 2.3.

As mentioned above, DID-based naming schemes are commonly found in blockchain-based designs of directory systems. One example would be uPort [137] which we dive into in more detail later as part of our discussions on related work in Chapter 3.

2.1.2. Name systems

On the surface, name systems appear to be quite primitive systems that at the same time are one of the major pillars of the Internet. In its simplest form, a name system is a system that allows registration and resolution of name-value mappings. For example, the DNS allows to translate domain names to IP addresses. Arguably, this is the most prominent and most important use case of the DNS and it is what enables users to browse the Web. However, the use cases for name systems are not limited to IP address resolution. As illustrated in Figure 2.1, a name system is just another type of directory service. We want to put particular emphasis on the striking resemblances between X.500 directories and name systems: Those include the hierarchical organization of DITs as well as the flexibility and use of its entries. Name systems such as DNS exhibit similar properties, most notably the hierarchical structure. However, name systems have one special feature that X.500-based services lack: Delegation of authority over parts of the directory structure to other entities.

In DNS, the namespace is divided into subnamespaces which are owned by so-called "*authoritative*"entities. Authoritative entities may further divide and delegate parts of their delegated namespace. The root namespace is reserved for a special entity and is identified by the label ".". This reserved namespace is called the "*root zone*". The authority over this namespace reserved to the Internet Assigned Numbers Authority (IANA) which is a function of the Internet Corporation for Assigned Names and Numbers (ICANN). Until recently, IANA was managed by ICANN under the contract of the U.S. Department of Commerce (DOC) [23]. At that time, the DOC verified additions and changes made to the DNS root zone. On in October 2016, the functions of IANA were fully transferred to ICANN[1].

At the second level of the hierarchy are top-level domains (TLDs) such as ".com" and ".org". A significant amount of TLDs are managed by states (country code TLDs). Figure 2.4 shows a fictional example of a DNS namespace hierarchy. In it, the top-level domain "de" is managed by the DENIC. Ergo, IANA *delegates* authority over the "de" subnamespace to DENIC. Names in DNS consist of ASCII labels which are optionally separated by a dot. DENIC itself further delegates subnamespaces of "de" using labels, such as "tum.de" to the Technical University of Munich. Each authority operates DNS servers which are capable of giving authoritative answers to name resolution requests.

Where X.500-based directory services require DAPs, in name systems name-value pairs are resolved top-down through the namespace hierarchy using *name resolvers*. Name servers that recursively query authoritative servers along the hierarchy until a result is found are called "recursive" resolvers. End users use so-called "stub" resolvers that are solely used to proxy resolution requests and responses to recursive resolvers. Stub resolver implementations are usually part of an operating system network stack.

Names in DNS are mapped to sets of resource records (RRs). Resource records serve the same purpose as entries in an X.500-based DIT. The content of a resource record is defined by its type. For example, the resource record type "A" denotes an IP address. Extensions to DNS define record types for X.509 certificates [63] or OpenPGP public keys [146]. Instead of DNs used in DAPs, in DNS we refer to records using *fully qualified domain names* (FQDNs). While the qualifier is different, the semantics and meaning of DNs and FQDNs are almost equivalent.

[1]https://www.icann.org/news/announcement-2016-10-01-en, accessed 2018/11/23

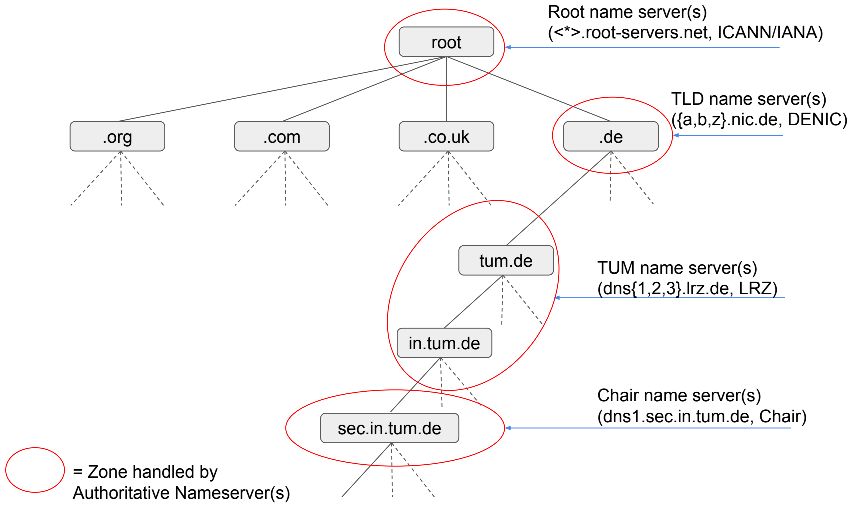


Figure 2.4.: An example DNS namespace hierarchy.

Security

Security of name systems is a research area which in the real world has found little application and even standardized security extensions are infamously unpopular. Paradoxically, this is presumably a result of the core role the name system plays on the Internet today. The DNS is critical infrastructure and as such disruptive changes to it are difficult to apply without risking the interruption of millions of services. Regardless, efforts to bring security properties to name systems have been made. This includes both approaches which aim to improve the existing DNS, but also blank slate approaches.

A variety of extensions and modifications exist which aim to add security properties to the DNS. Unfortunately, the DNS suffers from an inherent design flaw which makes it difficult to design a security model that is able to withstand strong attackers: As discussed above, domain names in the DNS are highly regulated and organized in a strict hierarchy. It is prone to gradually degenerate into a quasi centralized system from an organizational perspective. This property affects resilience as it attracts strong attackers including nation state actors and facilitates legal actions such as censorship.

Research done by Grothoff et al. [58, 24] includes surveys, analyses and categorizations of contemporary, state of the art name systems. The authors compiled a set of security goals which are found in name systems that claim to add security properties including integrity, authenticity, availability, confidentiality and censorship resistance to specific aspects of the name system and its records.

In Table 2.1, we can see the analysis results for DNS, various DNS security extensions, Namecoin, GNU Name System (GNS) and RAINS. It shows a comparison of the defenses offered by the various designs and their relative deployment complexity. According to the study, newer, disruptive approaches provide security properties absent in DNS-based name systems including its security extensions [65, 34, 19, 144].

This is particularly evident in the protection against network operator monitoring and censorship. The authors conclude that the choice of name system is depending on a variety of factors where security considerations only play a secondary role at best. For example, organizational aspects of the ecosystems where the name system is to be used and backwards compatibility including migration aspects are relevant. The authors state that the choice of name system is highly political. For free and democratic societies, it appears that peer-to-peer-based approaches such as Namecoin or GNS, should be preferred over semi-centralized, distributed systems such as DNS or RAINS. As our goal is the empowerment of users and returning the control over their identity back to them, this is a significant factor for us.

One of those disruptive approaches is Namecoin [94]. Namecoin is an alternative blockchain-based name system. The Namecoin blockchain is similar to the Bitcoin blockchain in that it makes use of a hash-based proof-of-work consensus mechanism. One property of blockchain-based name systems like Namecoin is that, except for stub resolvers, every client stores all records and zones in its local ledger database. For users, this means that record lookups are fast and cannot be intercepted or modified by an attacker in the network as only local data is queried. As such, it provides strong privacy and availability properties. The resulting drawback is the redundant replication of the complete ledger by every participant. For regular users that are not namespace administrators, Namecoin is essentially a local database with host names and other name-value mappings. It is similar to a local /etc/hosts file that contains a synchronized copy of all names in the domain name system.

The interesting property of Namecoin becomes evident at name registration. The blockchain proof-of-work decentralized consensus algorithm ensures that name-value mappings are difficult to delete or censor by third parties. This tamper resistance is an inherent feature of any blockchain and is ensured through the consensus mechanism. Integrity of the individual name value mappings is established through traditional public key cryptography signatures provided by the namespace owners.

Another interesting aspect of Namecoin is that there are already efforts that propose the use of this name system as identity directory [79]. We discuss this approach in detail as part of our presentation of related work in Chapter 3.

The other analyzed peer-to-peer-based name system is the GNS. GNS is a petname system [129]. A petname system implicitly mitigates name squatting by not having globally unique names but instead relying on query origin relative names. Unlike Namecoin, GNS does not use a replicated ledger as the storage mechanism of resource records. GNS is a name system built on top of a distributed hash table (DHT). In order to ensure availability of records, the GNS DHT also makes use of data replication across nodes in the network. But, the replication rate is nowhere near the 100 percent of a blockchain. Data replication is only as high as to avoid data loss in high node churn situations, making it more efficient. The DHT routing protocol ensures storage and retrieval of the mappings. Hence, lookups are not local but involve network queries and responses. GNS prevents namespace enumeration and eavesdropping by network

	Protection against						Ease of
	Manipulation	Zone	Client observation		Traffic	Censorship /	Migration /
	by MitM	walk	network	operator	Amplification	Legal attacks	Compatibility
DNS	✗	✓	✗	✗	✗	✗	+++
DNSSEC	✓	✗**	✗	✗	✗	✗	+*
DNSCurve	✓	✓	✓	✗	✗	✗	+*
DNS-over-TLS	✓	n/a	✓	✗	✓	✗	+
Confid. DNS	✗	n/a	✓	✗	✗	✗	++
Namecoin	✓	✗	✓	✓	✓	✓	-
GNS	✓	✓	✓	✓	✓	✓	- -
RAINS	✓	✗	✓	✗	✓	✗	- -

* EDNS0 is not perfectly compatible, ** with NSEC5: ✓

✗: not satisfied, ✓: satisfied, n/a: unchanged from DNS/DNSSEC, "+++/++/+" easy, "- -/-" hard.

Table 2.1.: Name systems security assessment results by Grothoff et al.[58]

operators using a query key blinding and encryption approach. In the context of our design of a decentralized directory service, we make use of this name system for our reference implementation. We provide more details and motivation on this choice in the context of Chapter 4.

2.2. Identity services

In order to utilize a service on the web, users are often required to share personal data like email addresses with service providers. As part of normal service operation, such as notifications or billing, services require access to uo to date user data. This data is required when a particular action is triggered.

Consider the use case of a user subscribing to a social networking service. After successful registration and providing the service provider with an email address, the service is able to send notifications such as status updates from friends to the user. At the time of notification delivery, the service needs access to the users' current email addresses that must be notified. However, services cannot interact with users that are offline in order to retrieve an up-to-date address. To mitigate this issue, services store user data in a database upon registration or retrieve it from a third party Identity Provider (IdP). If the data is persisted by the social networking service, it can become stale unless diligent users continuously update their data. Further, as elaborated in our motivation in Chapter 1, this data is a liability which poses potentially existential risks to service provider if accidentally disclosed or mishandled. This has lead to the emergence of decidated, third party identity services.

We distinguish between three types of IdP service architectures: centralized, federated and self-sovereign. The centralized approach is the current norm on the Internet and the so-called "Social Login" functionality provided by Google and Facebook IdPs services are archetypal representatives.

Federated identity services are traditionally found wherever disjoint trust domains must be interconnected. This use case is mostly found in the context of universities in order to facilitate common authentication for staff and students across research institutions. Another example are corporations that are faced with integrating possibly heterogeneous IT infrastructures as a result from mergers and acquisitions. In both cases, identity federation is a useful concept which allows to address such scenarios without the need of costly technology homogenization or inappropriate data sharing.

Finally, in the current wake of the blockchain hype, the addressing identity management in the context of decentralized architectures and applications gains traction. No central IdP can be integrated in such ecosystems without destroying the fundamental underlying premise of decentralization. Hence, decentralized ecosystems resort to so-called *self-sovereign* identity management. Self-sovereign identity management is the idea that users manage and share their digital identities without using central identity services. Instead, ad-hoc communication and peer-to-peer protocols are utilized in order to store and share identity data. The data itself is either self-asserted by the user or asserted by sovereign authorities such as governments (e.g. for postal addresses) or businesses (e.g. for email addresses) through out-of-band trust establishment. The user is enabled to share his data selectively with those who need it either directly or via a shared, decentralized storage medium.

In the following, we discuss and compare centralized, federated and self-sovereign identity systems while discussing their benefits and drawbacks.

2.2.1. Centralized

In centralized architectures, user identities and attributes are located, stored and managed in a directory service within the domain of the IdP service. Users are able to create identities by registering accounts and edit the respective identity attributes within the limitations imposed by the service provider. Control over attributes by the user is limited if the IdP validates and asserts attribute values such as account IDs or email addresses. Occasionally, non-critical information such as nicknames are left to the discretion of the user. Access control and data protection is ultimately delegated and enforced by the IdP.

Third party services, in this context referred to as relying parties (RPs), are able to request access to user attributes at the IdP. The details of such authorization procedures are defined in relevant specification such as OpenID Connect 1.0 (OIDC) or Security Assertion Markup Language (SAML). In order to interact with the IdP, RPs are usually also required to register at the service. Both user and service rely on the IdP to provide fresh, authentic attribute data. In this design, an IdP is in a position where usage patterns can be observed across RPs which can in turn be used for user profiling. Further, as RPs delegate user management and authentication to the IdP, insufficient or faulty user authentication as well as unavailable or incorrect user data may severely disrupt business processes. Consequently, this design requires users and RPs to place strong trust in the identity service provider to provide adequate availability, integrity and confidentiality guarantees. Figure 2.5 illustrates the architecture of centralized identity services.

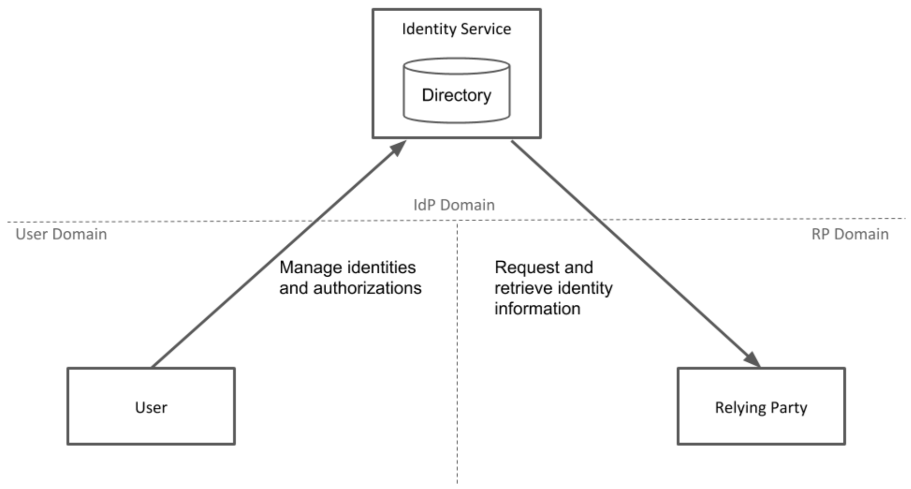
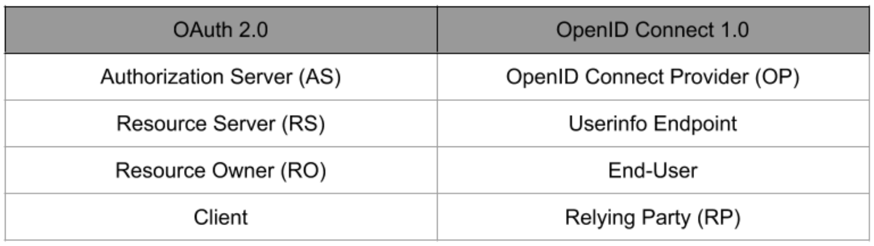

Identity Service
Directory
IdP Domain
User Domain
RP Domain
Manage identities and authorizations
Request and retrieve identity information
User
Relying Party

Figure 2.5.: A centralized IdP architecture.

Traditionally, central IdP services are built using the standardized protocol OIDC [112] or SAML. OIDC is a HTTP-based protocol for identity and access management. The official specification states that OIDC is an identity layer on top of the OAuth 2.0 [71] protocol. While OIDC is often used in a user authentication and login context, the protocol itself is not an authentication protocol or mechanism. OIDC is an authorization protocol that revolves around an end user giving authorization to access identity information to a RP. In OAuth 2.0, a *client* requests authorization to access a protected resource from the *resource owner* (RO). In the case of OIDC, the client is a RP, the RO is the user and the resource is the user's data. If the user consents, the RP receives an OAuth 2.0 *authorization grant*. Form factor, content and semantics of this grant depends on the chosen grant types as defined in the specification [71]. The RP uses the grant to request an access token from the OAuth 2.0 *authorization server* (AS). In OIDC the AS is in almost all cases the same entity as the IdP service which also serves the user's data. When presented with a valid grant, the IdP issues an *access token* to the RP. From then on, the RP is authorized to access the user information at the IdP by presenting the token.

In summary, the basic concepts of OAuth 2.0 apply directly to OIDC. Figure 2.6 illustrates the relation between OAuth 2.0 and OIDC entities. It shows a list of relevant entities in the OAuth 2.0 protocol on the right, juxtaposed with the corresponding entities in OpenID Connect 1.0.

Advantages The use of central service providers allows users to efficiently manage identity information and control access. Entities providing sovereign assertions regarding identities such as universities (enrollment status) or email providers (email address) often offer a whole range of IdP services. IdPs are able to realize this this one-stop shop approach with little to no additional effort using today's cloud-based infrastructure offerings. Further, users and RPs are primarily interested in the service functionality



OAuth 2.0	OpenID Connect 1.0
Authorization Server (AS)	OpenID Connect Provider (OP)
Resource Server (RS)	Userinfo Endpoint
Resource Owner (RO)	End-User
Client	Relying Party (RP)

Figure 2.6.: A juxtaposition of related entities in the OAuth 2.0 and OIDC specifications.

and less with the actual attribute assertion and management behind it. Availability is usually guaranteed through today's cloud infrastructures. Solely legal interventions, for example to impose censorship, are notable threats to the availability of centralized services.

Disadvantages The main disadvantage of centralization is that the IdP has full access and control over the managed user data. Abuse of this power is theoretically limited by applicable laws and regulations [53]. But, such regulations can be ignored or challenged [87] and service providers are major targets for targeted advertisement businesses as well as hackers, including government actors [30, 49, 66].

Further, centralized identity services inherently degenerate into identity service monopolies: From the perspective of a RP, the number of identity providers should be as low as possible because each may require additional integration effort. This issue is aggravated if identity providers use diverging formats for attributes and protocol implementations. As a consequence, RPs have an incentive to offer the bare minimum of IdP support to their users in order to guarantee a comfortable experience for most of them. This hinders small and new IdPs to enter the market. This effect becomes evident when looking at recent efforts to establish national alternatives [21] and their failure in gaining traction. As discussed above, centralized IdPs are a magnet for legal interventions as well as malicious actors that aim to inflict maximum impact. Consequently, large, centralized IdPs involuntarily act as prime targets for all kinds of attackers.

2.2.2. Federated

Identity federation is the approach to interconnect user populations across trust domains. In general, identity federation architectures build on the centralized IdP pattern discussed above and enhance it by allowing IdPs to exchange information. In a given trust domain, a RP is able to delegate authentication of a user to a different trust domain. The discovery of the respective user domain and associated IdP is done using dedicated protocol extensions such as the *SAML Identity Provider Discovery Profile* [97] or the *OpenID Connect Discovery* [111] specifications.

Historically, SAML was used for identity authentication and identity information sharing. Out of the many use cases for SAML, Single Sign-On (SSO) and identity federation are the most common. Use cases are modeled in so-called *profiles* in SAML. The most common profile is *Web Browser Single Sign-On*. Counterparts to relevant actors and entities in SAML can also be found in other, newer protocols such as OIDC. SAML makes heavy use of the Extensible Markup Language (XML)-based Simple Object Access Protocol (SOAP), which is characteristic for the time period it was designed in. Newer protocols like OIDC are build using HTTP and representational state transfer (REST)-based protocols. While OIDC has replaced SAML in most use cases including SSO, SAML remains the dominant protocol for identity federation.

Figure 2.7 illustrates how user information flows across different IdPs and trust domains from users to RPs in identity federation scenarios.

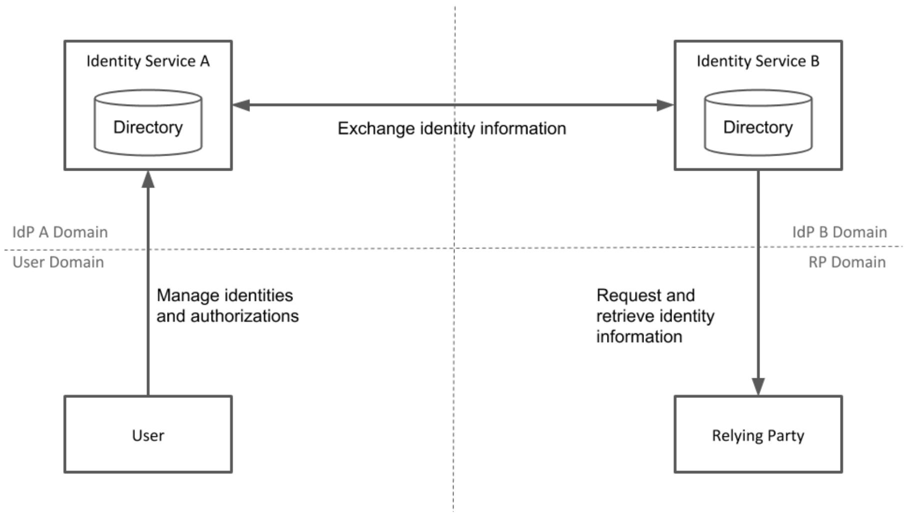


Figure 2.7.: A federated IdP architecture.

Advantages Identity federation theoretically makes it easier for a variety of identity providers to enter the market. Users are able to manage their identities in their trust domains and even at the IdP of their choice. Through this property, one could argue that market mechanisms force IdPs to follow consumer demand also with respect to privacy. Trust between domains is established individually between IdPs, bridging the possible gap between user and RP trust domains. Standardized protocols were specifically designed to support this use case and it is successfully applied widely in academia [37].

Disadvantages The biggest problem with identity federation is that large IdPs have no incentive to federate with other, smaller competitors. However, a well connected network of IdPs is central to the idea of large scale federated identity management. At the same time, RPs have strong incentives to integrate and trust large IdPs in order to cater to the larger user populations. It is obvious how this disparity leads to a situation where RPs tend to establish trust directly with large IdPs instead of via local competitors. Commercially, identity federation is not so much of an advantage which would explain why SAML identity federation is mostly popular inside academia and large corporations and much less in the consumer market.

2.2.3. Self-sovereign

The idea of identities without trusted authorities is not new. Previous research can be found from as early as 1996, when Carl Ellison proposed a scheme to "establish identity without certification authorities" [38]. In the wake of peer-to-peer and blockchain based services, decentralized self-sovereign identity systems re-emerged. The requirement for such systems arises from the fact that decentralized services and applications depend on a decentralized identity system that conforms with the respective threat models. Consequently, management and sharing of attributes and identities needs to be decentralized. In self-sovereign identity systems, users share personal information with RPs interactively upon request bypassing any third party IdP service. In its simplest form, the identity data is not asserted by any trusted third party. However, this does not mean that self-sovereign identity sharing does not support use cases where identity data is asserted by a trusted authority.

Effectively, there are two locations where users manage personal information in self-sovereign identity systems: locally or using a shared, distributed storage medium. In the first case, personal data is managed by the user in a local credential "wallet" on their devices such as personal computers or mobile phones. In the second case, users store data on decentralized storage mediums such as DHTs or blockchains. The latter case requires decentralized access control mechanisms in order to protect the user's data from unauthorized access.

One example for a system which follows the credential wallet approach is Idemix [22]. While the authors of Idemix do not categorize it a "self-sovereign" identity system, its architecture exhibits the same properties. However, this approach makes it impossible for the user to share his data in a way that allows RPs to access them even if the user's devices are offline. In fact, it forces RPs to persist the data on their end in order to ensure it continues to be accessible in the future. Centralized and federated IdPs achieve this by storing the data in their respective directory services. This implies the requirement of accessible, open and decentralized shared storage mediums. Figure 2.8 illustrates a high-level design of self-sovereign identities managed over a shared medium.

It is possible to combine the above approaches in order to allow interactive presentation of attributes as well as non-interactive lookups of attribute data by RPs. Access to the directory service does not necessarily need to include the IdPs and is instead only used to bridge the trust domains of the user and the RP. In Figure 2.9, we illustrate how a user

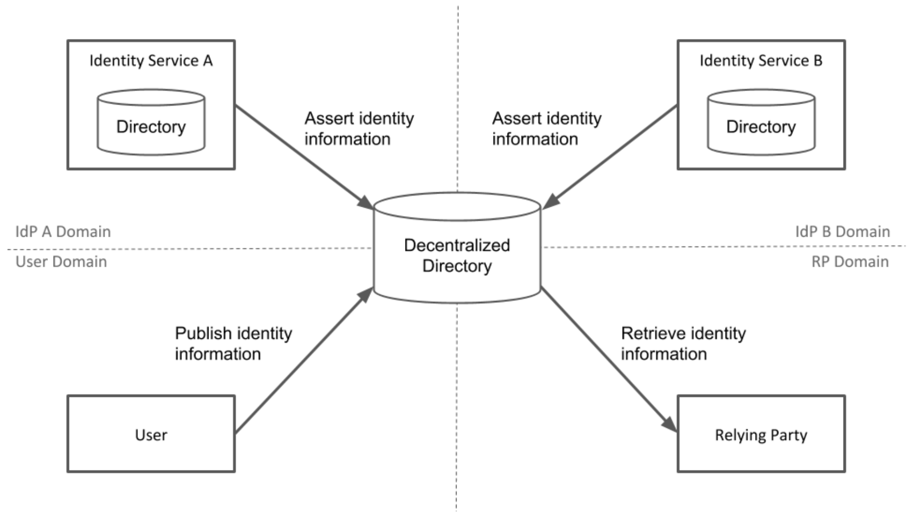


Figure 2.8.: A self-sovereign identity architecture with a decentralized directory service.

is able to share IdP asserted attributes with an RP using either an interactive protocol or the indirection of the shared storage medium.

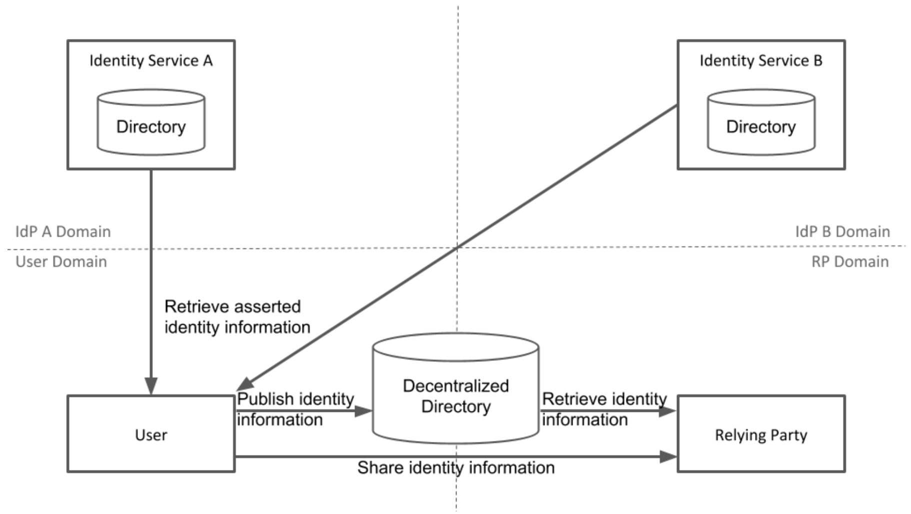


Figure 2.9.: An architecture combining a decentralized directory service with ad-hoc attribute sharing.

Advantages Self-sovereign identity systems put the user at the wheel when it comes to identity attribute sharing. No intermediate entity is required in order to share user data with RPs. At the same time, the possibility of attribute attestation through trusted third parties is preserved. Ideally, self-sovereign identity systems additionally include a directory service built on top of a shared storage medium. Such hybrid architectures support the separation of concerns we advocate for: The disentanglement of identity provisioning and trust establishment.

As with its centralized counterparts, IdPs still have a place in self-sovereign identity systems in order to provide data verification as well as attestation. However, IdPs are removed from the equation when it comes to access control and authorization of RPs through the user.

We even argue that this mitigation of central identity service providers enables email providers, governments and other authorities to reclaim their place as sovereign attestation entities themselves. At the same time, users are free to assemble one or more identities with asserted or self-asserted attributes and share this information on a need to know basis with RPs.

Disadvantages The classical self-sovereign identity approach comes with a few drawbacks: If data is only shared on demand between the user and a requesting party, propagation of updates to the data must be handled by the user as well. RPs might have processes that require access to user data but those processes are not triggered while direct interaction with the user is occurring. For example, a monthly notification or a billing process requires fresh addresses and banking information. To address this case, the RP has only one option: It must persist the data when it was requested from the user. If the data changes, for example when it is updated by the user, the RP is left with stale data. Another challenge in self-sovereign identity systems is the absence of implicit attestations of user attributes provided by an IdP in centralized architectures. Trust in attributes served over self-sovereign identity must be established out-of-band, for example through the use of PKIs or other trust establishment mechanisms.

There are three inherent challenges of directory services built on shared mediums: First, shared mediums such as DHTs or distributed ledgers are quite difficult to design and build. Further, enforcement of access control decisions by the user as data owner is difficult due to the absence of a trusted third party which usually takes care of this. The second challenge is ensuring confidentiality of attributes. This can be addressed through the use of an encryption scheme but this entails other challenges with respect to key management and revocation. Third, the implicit attestations which are usually in place when retrieving attributes via a centralized service are not available. Consequently, trust in attributes must be established out-of-band.

2.3. Trust management

In order to facilitate secure communication on the Internet, first and foremost trust between entities must be established. Respective mechanisms and protocols fall into the context of identity and access management.

Traditional communication paradigms based on centralized client-server architectures are shifting to decentralized communication between interconnected devices and services especially in the context of self-sovereign identities. The decentralization of

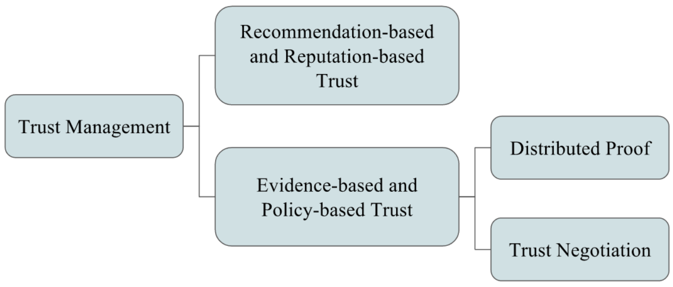

Trust Management
Recommendation-based and Reputation-based Trust
Evidence-based and Policy-based Trust
Distributed Proof
Trust Negotiation

Figure 2.10.: Trust management categories. [5]

communication and the diversification of trust domains entail non-trivial challenges on trust management.

Centralized architectures typically imply a single trust domain. In such cases, trust establishment is well understood and various solutions are available. It typically consists of a trusted third party that provides authentic descriptions of entities and asserts validity of properties. All entities use this information and interpret it as the trusted foundation for access control models. This model is founded on the assumption that assertions and attributes are managed by the central trusted third party. This assumption does not hold in decentralized architectures that consist of a high number of disjoint trust domains, as there is no central authority to manage assertions and to guarantee the authenticity of entities and attributes.

We investigated the landscape of trust establishment systems in distributed and decentralized systems in a survey [5]. In the survey, we categorize the employed trust models into whether they can be used in centralized, distributed and decentralized scenarios. Depending on the design of the various trust models, they exhibit properties that allow them to be used in self-sovereign identity systems. Figure 2.10 shows the different trust model categories which we discuss in the following.

2.3.1. Recommendation and reputation-based systems

According to Artz and Gil [7] there are two approaches to establishing trust in an entity. The first approach is a result of direct, personal interaction with the entity. It is the most common approach in trust establishment and the prime example is the X.509 PKIX which we discussed as part of directory services. The second approach is to establish a "reputation" of the entity using information from third parties, such as other peers in a network.

Recommendation and reputation-based trust models are commonly evaluated on the basis of trust metrics. Such metrics are computed using trust data which is collected and aggregated in the form of recommendations or reputations asserted by other entities. The used metrics and calculations vary across approaches which is its own field of research and we consider this out of scope in the context of this thesis. Reputation systems such as the one proposed by Abdul-Rahman and Hailes [1] establish an entities reputation by distinguishing between "direct" and "recommended" trust. Often, systems that model recommendation and reputation-based systems, combine both direct and recommended trust models [3, 2]. Notable approaches include the *EigentTrust* and *EigenRep* systems by Kamvar et al. [74, 73]. In EigenTrust, global trust values are calculated between two entities based on their previous local transaction history. In order to do this, transactions between entities are categorized as either "satisfactory" or "unsatisfactory". When calculating the global trust values, local trust values are weighted by the reputation of the entity which claims the local trust value over another entity. However, Selvaraj et al. [127] found that approaches such as EigenTrust and reputation-based systems in general require a central storage of reputation data. They argue that unless a trusted central storage is available, reputation systems are susceptible to a variety of security threats. The EigenRep approach [73] tries to mitigate those shortcomings through the use of DHTs.

From our survey data we conclude that recommendation and reputation-based systems generally suffer from the problem that a recommendation is based on a metric that is difficult to describe objectively. In the literature, this metric is commonly referred to as "transaction quality". However, unless a global consensus on the meaning of acceptable behavior for entities is found and interpreted identically by all entities, the meaning of a resulting metric is of little use. Further, the resulting metric allows to compare the two entities with respect to their reputation, but it is difficult to formulate sensible binary authorization decisions on a continuous value. As such, binary decisions with respect to trustworthiness are difficult.

In conclusion, recommendation and reputation-based fit into decentralized systems and consequently self-sovereign identity management. However, their shortcomings with respect to fuzzy metrics and the lack of practical, mature implementations are disadvantageous.

2.3.2. Evidence and policy-based systems

Evidence and policy-based trust models mostly revolve around credentials and policies. Unlike recommendation and reputation-based models, they rely on strong security mechanisms such as cryptography in order to establish absolute trust. Examples range from standardized PKIs such as X.509 to distributed policy languages.

X.509 historically is a type of entry in an X.500 directory service. However, the IETF in its PKIX working group [67] built upon the original definition in order to appropriate it for use as a PKI on the Internet. A X.509 certificate represents the binding of a public key to a distinguished name (DNs, see Section 2.1.1). However, a certificate may contain further assertions defined through extensions. The PKIX specifications define how trust in an entity can be established by following a trust chain. A trust chain must start at one or more certification authoritys (CAs) as trust anchors. Trust domains can be connected in X.509 PKIs through cross certification: In order to enable trust establishment from an externally certified entity, the local PKI creates a certificate for the external CA. This essentially integrates the external PKI into the local PKI. However, if a trust path is created to a CA outside the local PKI in this manner, it cannot be restricted from issuing certificates for its own trust domain. This property is a severe security issue if CAs are certified which become compromised or act maliciously. Research in this regard yields that this threat is not theoretical [64]. The possibility of cross certification is an important difference to DNS namespace delegation where only the authority over a subnamespace can be delegated. X.509 naming schemes follow the conventions of X.500 and not DNS. Yet, X.509 does have an infamous relationship with domain names: Commonly, X.509 certificates are used to bind DNS domain names to the public keys used by web servers in the Transport Layer Security (TLS) protocol. In this case, the common name[2] contains an FQDN. In web use cases the domain name of the server must match this entry defined in the certificate.

Alternatives to X.509 PKIs exists in the form of Simple Distributed Security Infrastructure (SDSI) [107] or RT_0 [85]. Both try to mitigate some of the above mentioned shortcomings of X.509 PKIs through the use of attribute-based delegation (ABD). ABD is a suitable trust establishment mechanism especially in the context of distributed and decentralized systems. In our discussion of state of the art trust establishment in distributed systems in Chapter 3 we take a closer look at those approaches and how they can be used in self-sovereign identity systems.

We conclude that, generally, traditional evidence and policy-based systems are also suitable for use in combination with self-sovereign identity systems. In this thesis, we investigate to what degree alternative approaches such as SDSI and RT_0 can be used to support trust establishment in self-sovereign systems.

2.4. Relevance in the context of this thesis

We use the background presented in this chapter in order to establish the technical context of this thesis. We revisited existing protocols and technologies in order to separate current identity management into three functional areas:

- The storage of user identity information in *directory services*.
- Identity information sharing between users and RPs through *identity services*.
- Verification and assertion of identity information through *trust establishment*.

This deliberate breakdown enables us to effectively address the challenges we stipulated in our motivation. It allows us to split up current identity management models which revolve around centralized service providers into disjoint functional components. Accordingly, our contributions as presented in the main body of this thesis address the respective functionalities. Further, it facilitates our discussion and classification of the state of the art in decentralized identity management in the following chapter.

[2]Recently, domain names are also found in the "SubjectAltName" attribute.

CHAPTER 3

State of the art

In this chapter, we present existing research in the areas of privacy-preserving and self-sovereign identity management, data sharing and trust establishment. Our goal is to differentiate our contributions from the state of the art.

We organize this chapter as follows: First, we discuss research on privacy-preserving identity management systems built on top of centralized infrastructures and standardized services. In the second part, we present works on decentralized and self-sovereign identity management. The respective approaches rely on peer-to-peer systems and distributed ledger technologies such as blockchains. In contrast to the centralized approaches, the security goals of such systems are usually modeled with stronger attacker models in mind that necessitate the decentralization of the underlying infrastructure and services.

Third, we present related work in the area of trust establishment in distributed and decentralized systems. Works in this research area complement the inherent lack of attribute validation and assertions in self-sovereign identity systems. Most publications in this context do not directly reference self-sovereign identity management as research in this area largely predates this topic.

Finally, we give a brief summary over the insights we draw from existing research and how the state of the art benefits from our contributions.

3.1. Centralized, privacy-preserving identity management

In this section, we present research with focus on privacy-preserving protocols which integrate or are based on centralized infrastructures and services. This includes research on protocols which extend existing standards such as OpenID Connect 1.0 (OIDC) and add security and privacy properties, for example in the form of privacy-preserving attribute-based credentials (ABCs).

3.1.1. U-Prove

U-Prove [100, 99] is a digital credential technology that allows a user to selectively disclose claims issued by an issuer to a relying party (RP). Such claims can be presented to a RP through the use of authorization protocols such as OIDC. The users are free to choose which attributes they want to present to a RP and which to withhold. In addition to selective disclosure of subsets of attributes, U-Prove also allows users to prove predicate statements on attribute properties. A prominent example use case is where users must provide to an online service that they are of legal age in order to be allowed to use the service. For the RP, it suffices to know that a user is over a certain age. The exact date of birth might be irrelevant.

U-Prove aims to improve the privacy of users through the use of privacy-preserving credentials which are managed and presented via traditional identity provider services. U-Prove primarily excels in protecting the user from malicious or data greedy RPs. It is currently developed by Microsoft Research [106], but has not matured enough to be part of the regular commercial offerings.

Differentiation In U-Prove, the user must request claims and respective predicate statements through an interactive protocol with an issuer. Usually, this is the same entity as the Identity Provider (IdP). The IdPs must be trusted with all attribute data and they continue to be able to track the user across services. For example, for users to prove to a RP that they are over 18 years old, they must request a claim which states this information as part of a U-Prove token from the issuer. The proven statement cannot be modified by the user at a later stage, even within the validity of the underlying attribute value.

In this thesis, we present an approach that allows users to generate arbitrary statements themselves as long as they are true with respect to the underlying attribute claims. The user only needs to retrieve the asserted attribute – e.g. a certificate that states he is born in a certain year – from the issuer. This claim is sufficient for the user to prove statements based on this claim such as "is not 20 years old", "is 24 years old" or "is over 18 years old".

3.1.2. Identity Mixer

Identity Mixer (Idemix) [22] is and approach similar to U-Prove. It is built using novel cryptographic techniques based on the mathematical principles behind RSA and Diffie-Hellman. Idemix is a sophisticated credential system that in addition to privacy-preserving attribute-based credentials (PP-ABCs) also provides some anonymity functionality, which is what separates it from U-Prove. It provides an extensive feature set including support for attribute predicates, revocation and selective disclose of attributes. Idemix heavily relies on centralized services which are used by the user and RPs. The zero-knowledge presentation protocols required for Idemix credentials allow the same applications as those discussed above in U-Prove.

In Idemix, users manage a local "credential wallet" containing ABCs issued by an Idemix IdP. Users engage with RPs in an interactive presentation protocol which allows them to prove predicates and statements over the Idemix credentials in the wallet. Presentation of the credentials does not require interaction between user and IdP or RP and IdP unless full disclosure of the attribute contents are required. User studies show, that while the idea of credential wallets is easy to use, trust in such complicated systems is difficult to establish [109].

Differentiation The attacker model of Idemix, especially with respect to attribute sharing and disclosure as well as user tracking across RPs is not particularly strong. In our contributions, we aim to provide technological means which are resistant to data collection and behavioral profile building while at the same time provide PP-ABC features similar to those of Idemix.

In Idemix, while users are able to provide credentials in a non-interactive fashion, the system design does not include a decentralized store which would require this feature per se. By design, Idemix does not address the use case of offline presentation of credentials to RPs. What Idemix gains from this restriction, however, is that an already presented proof cannot be replayed by a RP unless the issuer consents to this. This feature is particularly relevant since Idemix supports the use case in which the user stays anonymous and only provides credentials. In this case, mitigating replay attacks is important to prevent impersonation attacks. A rather odd choice in the design of Idemix (which otherwise emphasizes user privacy) is that RPs may request disclosure of credential values at the IdP, potentially without the consent or knowledge of the user.

In our research efforts, we focus less on anonymity and more on empowering the user to regain control and sovereignty over potentially pseudonymous identities. Unlike Idemix, we consider transparency of paramount importance and the user should have full disclosure over what happens with the shared personal data. Further, we assume that interactive sessions between user and RP are authenticated. The user authentication context is then bound to the presented credentials anyway, for example through an attribute holding his public key. However, we built on the idea that users manage their own credential collections on their local devices and either interactively or non-interactively present them to the RP. In Chapter 5, we investigate how recent advancements in non-interactive zero-knowledge (NIZK) protocols can be used in order to design PP-ABCs.

3.1.3. SPRESSO

The authors of SPRESSO [44] propose a secure, privacy-preserving identity architecture primarily focused on the use case of Single Sign-On (SSO) on the Internet. At the expense of standardization and the associated ready to use tooling, the authors propose a protocol redesigned from the ground up in order to mitigate some of the inherent privacy issues of legacy versions of OAuth and OpenID[1] [139, 131]. The SPRESSO protocol allows users to login at any RP using their email address. Here, it is irrelevant what email address is used as long as it is possible to discover an SPRESSO IdP with it: SPRESSO RPs use the domain part of the user's email address to discover the user's IdP service. This process is similar to OIDC IdP discovery [111] which did not yet exist at the time. Due to this feature, RPs are not required to register at IdPs a priori, which is something that other designs such as OIDC do not support. Because of this, the authors present SPRESSO as a "decentralized" system. However, it fundamentally relies on traditional, centralized identity service providers. Hence, it is not designed in a way which would allow categorization as a self-sovereign identity system. User data is still under the control of IdP services.

In order to hide the relation of user and RP, the protocol intends so-called *fowarders* which proxy the communication between user, RP and IdP. The authors claim that SPRESSO can be used by users without further software or browser plugins as a major advantage over approaches like Mozilla's decentralized authentication system *Persona*[2]. But, the authors assume and require that forwarders are not malicious and do not collaborate with RPs or IdPs in order to diminish the privacy of participants. Finally, the exchange of attributes is not possible if the user is offline as this would destroy the privacy provisions of SPRESSO.

Differentiation Unlike the authors of SPRESSO, we concede that in order to achieve truly privacy-preserving, user-controlled identity management, additional software is required. The protocols criticized by its authors have been superseded and today's incarnations, namely OAuth 2.0 and OIDC, remedy most of the identified security flaws. However, privacy flaws inherent to the design around centralized service providers remain in both protocols. That is, if the services are not operated by entities which must be trusted by both users and RPs. In this thesis, we present our approaches with which we aim to combine the best of two worlds: Standardized protocols like OpenID Connect built on top of self-sovereign identity and directory services.

3.1.4. UnlimitID

The authors of UnlimitID [70] propose privacy-preserving credentials based on algebraic MACs (aMACs). The system offers unlinkability between the user IdP and the RPs. The high-level idea is that the user is enabled to derive pseudonymous identities from a main identity registered at the IdP. Different pseudonymous identities are used when interacting with RPs. The use of aMACs prevents IdPs from linking derived pseudonyms to the original identity. The authors of UnlimitID acknowledge that access to user attributes must be possible without user interaction. Hence, they include the IdP

[1]We note that the authors specifically address the legacy OpenID specification and not the more recent, significantly different OpenID Connect specifications.

[2]The service and support by Mozilla has been decommissioned in 2016: `https://wiki.mozilla.org/Identity/Persona_Shutdown_Guidelines_for_Reliers`, accessed 2019/2/6.

in their attacker model and implement a protocol which protects the user's privacy even in the face of malicious service providers. UnlimitID can be used in combination with standardized protocols such as OIDC which is a major advantage over approaches like SPRESSO [44] or PseudoID [33].

Differentiation The security model of UnlimitID explicitly excludes the linking of identities through the disclosure of attribute values, such as email addresses. Hence, the unlinkability of identities is only guaranteed if the attributes shared with RPs do not allow to uniquely identify a user. For example, an email address is a unique identifier which would allow an attacker to easily discover an identity even if UnlimitID pseudonyms are used.

We consider this limitation not practical as one of the most common attributes which are shared with RPs today are email addresses. In addition, UnlimitID integrates with OIDC, which does not specify the used authentication mechanisms. However, the authentication may also allow to de-anonymize the user. This means that UnlimitID still depends on the IdP's honesty when it comes to the privacy of user attributes. Consequently, the threat model of UnlimitID addresses a "curious but honest" IdP. The IdP in the design is allowed complete access over all user attributes.

We conclude that while UnlimitID is an innovative approach and the idea of aMACs is an effective method to prevent tracking of users. However, our motivation and challenges are only partially addressed by this approach.

3.2. Decentralized directories and self-sovereign identity management

In the following, we present related work in decentralized directory services and alternative data sharing systems relevant to our research. We argue that in order to decentralize an identity sharing service, we must first design a decentralized directory which allows users to manage identity data.

The concept of decentralized identity services emerged from the idea of self-sovereign identity management as discussed in Section 2.2.3. Various approaches based on distributed ledgers and blockchains try to address this challenge.

3.2.1. NameID

A recent and prominent approach to decentralized identity management is NameID [79]. NameID is a blockchain-based identity system that allows users to register identities and manage identity attributes. NameID is an interesting approach as it illustrates the distinction between a decentralized directory service and the identity service. The NameID identity service, which is an OIDC implementation, is used to facilitate data sharing between users and RPs through a standardized protocol.

The service uses the Namecoin name system as a decentralized directory service for its user population. In addition, the OIDC service provides relying parties with identity information found in the blockchain. It also authenticates users through a proof-of-possession authentication mechanism. The user provides proof that he is in possession of a specific private key which is used to register names in the name system.

In NameID, end users register identities and store attributes in the Namecoin [94] name system. The end user is in complete control over this information and responsible for the management of attributes. This is done exclusively via blockchain transactions which bind the user's public key identity to the attributes. The proof-of-work consensus model in use by the Namecoin blockchain makes it difficult and economically infeasible for any third party to edit or delete the respective information. Through this mechanism, registration of identities and management operations on attributes are completely decentralized. Figure 3.1 illustrates the architecture of NameID.

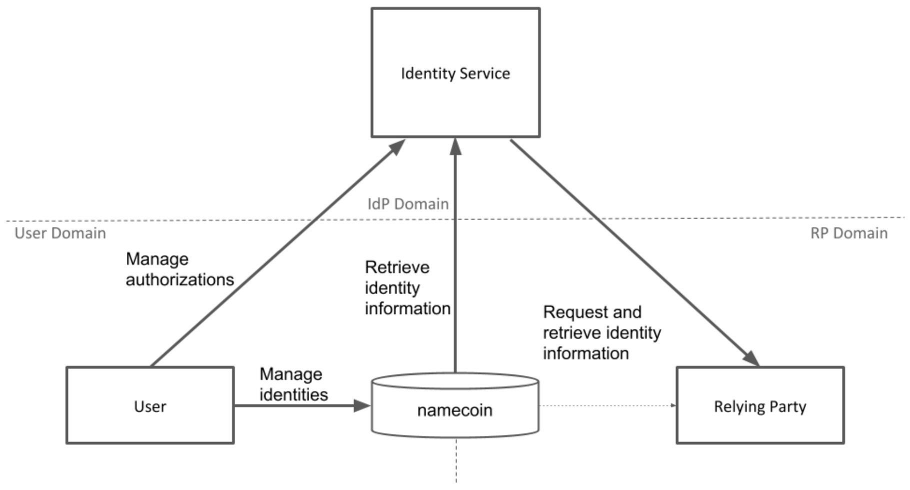


Figure 3.1.: The NameID architecture.

Finally, identity attributes in NameID are self-asserted by the user. This means that there is no third party authority issuing or certifying the users' statements over himself. While this does not exclude the possibility of a third party issued attribute to exist inside NameID, it requires further considerations.

Differentiation The central OIDC service enforces the users' access control decisions within the specification. However, it is not possible for the service to effectively enforce this selective authorization of specific RPs to access identity attributes: Identity attributes stored in the name system are inherently public and replicated across all participants of the Namecoin peer-to-peer network. All user identities and attributes are stored in plain text and are effectively disclosed in a public directory. This renders the access control decisions that a user makes in the process of an authorization pointless.

As illustrated in Figure 3.1, reading all the information stored in the directory is trivial for the RP by skipping any interactions with users and the OIDC service. Furthermore, as we discussed in the background chapter on centralized IdPs, all requests by RPs are relayed over a single entity: The OIDC IdP service. The IdP constitutes a single point of failure and is omniscient to all interactions between users and RPs.

To summarize, NameID has two significant drawbacks: First, the data stored in the blockchain is public information. As such, storing sensitive personal information is not viable. Second, while the directory service is decentralized, the identity service relies on a centralized service instance which is needed to enforce access control decisions made by users. In addition to those inherent drawbacks of the design, the use of the Namecoin blockchain is expensive both financially and from a resource perspective. Each transaction, such as the modification of an attribute, incurs costs at the expense of the user. Transaction fees as well as power consumption and the resulting electricity bills are significant factors.

In our contributions detailed in Chapter 4, we also propose a secure name system as directory service. Unlike NameID, we use cryptographic means to protect the data stored in the name system in order to facilitate decentralized access control. Further, as we do not rely on a blockchain, we mitigate any resource-intensive and possibly unsustainable consensus mechanisms.

3.2.2. DecentID

Like NameID, DecentID [47] is an approach to design a decentralized identity and personal data sharing system through the use of a blockchain. The blockchain in this case is Ethereum [40]. Instead of using the blockchain like a name system-based directory service, the authors of DecentID implement procedures for the management of user identities as smart contracts. Like NameID, DecentID does not directly address the problem of third party attribute assertions.

Identities in DecentID are represented by a public-private key pair. The public key must be registered in a global registry which is represented by a smart contract. This mapping is strikingly similar to the identity namespaces of NameID and is not far off a decentralized name system itself. The registry is the first of a number of indirections which eventually form a link from identity to attributes.

Each identity is associated in the ledger with a *root identity contract* (RIC). Identity information is shared by the user with RPs by creating *shared identity contracts* (SICs). Identities and attributes are accessed through so-called identity locator files (ILFs) and attribute locator files (ALFs). Both contain metadata and are encrypted using symmetric keys and stored directly on the blockchain. The actual attribute data, such as an email address, is stored off-chain in a distributed hash table (DHT). The confidentiality of attribute data is ensured through traditional symmetric key encryption. Regarding revocation, a rollover of the symmetric keys which is used to encrypt the attribute data in combination with a voiding of the respective SIC of the revoked RP is sufficient. Deep key hierarchies and indirections between the SIC and the actual attribute data ensure that other RPs do not need updates to their respective SICs. Figure 3.2 illustrates the architecture of DecentID.

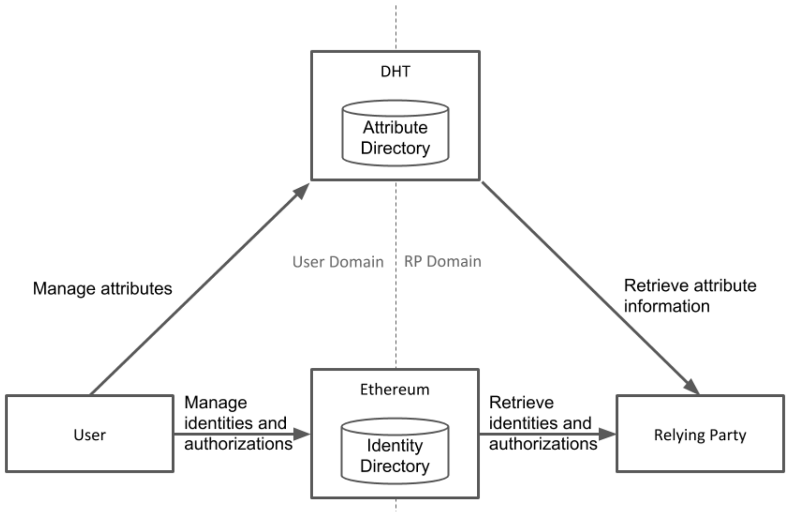

DHT
Attribute Directory
User Domain
RP Domain
Manage attributes
Retrieve attribute information
Ethereum
Identity Directory
User
Manage identities and authorizations
Retrieve identities and authorizations
Relying Party

Figure 3.2.: The DecentID architecture.

Differentiation In DecentID, analog to NameID, executing smart contracts which modify the state of an identity incur costs for the user. According to the authors this is intentional as it mitigates the problem of Sybil identities and squatting. The design is tied to the underlying Ethereum technology and its smart contract specifications. Further, it relies heavily on a multi-level symmetric key hierarchy in order to realize access control.

Instead of deep and complex key hierarchies, our approach uses attribute-based encryption (ABE). This allows us to greatly reduce the number of required keys. DecentID requires one key for every shared attribute with a single RP. Considering that keys are stored on the blockchain, and that storage on Ethereum is not free, this is a significant drawback. Finally, it is unclear if it is possible to use DecentID through standardized protocols such as OIDC as its authors do not propose anything in this direction.

3.2.3. Sovrin

Sovrin [36, 46] is a decentralized, hyperledger-based [59] identity system. It is driven by the idea of an open, accessible and decentralized system for users to manage their digital identities. It implements the DID [105] specification and explicitly allows users to establish pseudonymous identities. It integrates the use of privacy-preserving credentials which are also used in the context of Idemix [22].

The hyperledger in Sovrin consists of a multi-tier trust model. There are highly trusted nodes, called "validator nodes" which are in charge of ledger updates. This model allows hyperledger-based architectures to avoid resource-intensive consensus protocols at the expense of partial centralization. Users share claims and attributes off-ledger via proxy applications, called edge applications. Consequently, the privacy issues which arise in NameID are mitigated at first glance.

Figure 3.3 illustrates attribute sharing in Sovrin.

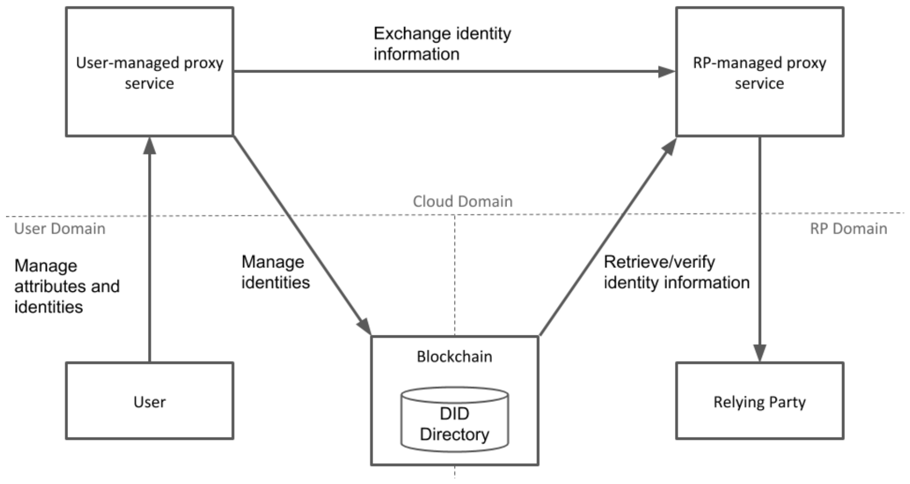

User-managed proxy service
Exchange identity information
RP-managed proxy service
Cloud Domain
User Domain
RP Domain
Manage attributes and identities
Manage identities
Retrieve/verify identity information
Blockchain
DID Directory
User
Relying Party

Figure 3.3.: The Sovrin architecture.

Differentiation Client applications in Sovrin include mobile applications on the users phone and dedicated cloud services. While the authors of Sovrin claim that such services are decentralized and peer-to-peer, the hoster of the cloud service is not meant to be the user. Consequently, this design faces a dilemma we identified initially: Users may manage and share their data via locally installed software and accept that their data is not accessible to the RPs if they are offline. Or, users use external service providers in the form of cloud infrastructure providers that run their proxy edge applications. In the latter case, availability and privacy in the face of strong attackers are significantly weakened.

Our approach tries to mitigate the dilemma that Sovrin faces with respect to decentralized sharing of attribute data. We share the ethical motivation and initial, high-level technical approach of Sovrin. However, our contribution shows that sharing of attribute data including the disclosure of attribute contents is possible without the user being online. Further, we show that trust can be established without the use of a distributed ledger and potentially associated maintenance overhead.

Finally, by using a general-purpose name system as opposed to a dedicated blockchain network only used for identity management, we are able to benefit from dissemination synergies due to a wide variety of use cases.

3.2.4. uPort

uPort [137] is an Ethereum-based [40] identity system that also implements the Decentralized ID specifications [105]. Ethereum is used to implement various smart contracts that deal with identity and attribute management as well as attestation of attributes. Disclosure and sharing of identity attributes is done interactively between users and relying parties. In uPort, public identity profile data is stored in the Interplanetary File System (IPFS) [69]. The user binds his public key to a hash link which points to the public profile data in IPFS in the Ethereum blockchain. The blockchain guarantees that this link can only be modified by the owner of the respective private key and IPFS data is always referenced by a hash of the data. While this ensures integrity of the public profile data, private data which the user wants to selectively disclose must be transferred interactively between user and RP. According to an older version of the whitepaper [138], it is envisioned to implement selective disclosure of attributes stored encrypted in IPFS. However, we assume that the proposed scheme would inevitably lead to issues with authorization management and access control as a single symmetric key is shared with all authorized parties. The architecture of uPort is similar to that of Sovrin with the only difference that identity data is not stored in the ledger, but in the IPFS DHT.

Differentiation uPort is a representative of the blockchain-based self-sovereign identity systems. Its authors propose an approach which is similar to our contributions in that it is focused on user managed identity data stored in a decentralized directory. In the case of uPort, this decentralized directory is IPFS. At the same time, its design includes third party identity providers which issue user credentials. uPort's design exhibits a few weaknesses: While it allows users to store public profile data in IPFS, they must manage and present private identity attributes interactively to RPs. uPort also requires a central service to share private identity attributes between user and RP which according to older specifications [138] does not provide end-to-end encryption between user and RP. Finally, users bear the bulk of Ethereum-induced transaction costs and must provide resources for ledger upkeep and storage which is not negligible. In addition to the ledger storage, users are also required to use IPFS which means they need to provide storage for two different distributed peer-to-peer infrastructures, albeit IPFS exhibiting a smaller, manageable footprint.

3.3. Distributed and decentralized trust establishment

In this section, we present state of the art research in trust establishment with a focus on distributed and decentralized systems. The approaches presented below either explicitly or implicitly define public key infrastructures (PKIs) and corresponding trust establishment algorithms and protocols.

In the following, we discuss related works into two areas: First, we present attribute-based delegation (ABD), a research area which includes older, mature approaches to alternative PKIs. Second, we take a look at more recent approaches with a focus on privacy-preserving key transparency protocols.

3.3.1. Attribute-based delegation

In the literature, the complexity and shortcomings of X.509 were identified and a flurry of alternative approaches proposed [15, 16, 25]. This includes approaches which advocate for the decentralized management of attributes using attribute-based delegation (ABD). ABD is a technique to delegate authority over attributes from one entity to another in a decentralized way. Prior research [14, 81] indicates that ABD is a suitable solution to model complex, decentralized trust relationships. In the context of access control and authorization, trust establishment through attribute delegation allows us to eliminate the need for central trust anchors and it facilitates scalable, decentralized provisioning of attributes.

The Simple Distributed Security Infrastructure (SDSI) [108] by Rivest et al. is one example for an ABD system. It was designed in direct response to issues identified in X.509. Unlike an X.509 PKI, which assumes globally unique identifiers, SDSI follows the concept of locally scoped identifiers. Local identifiers can be linked to external domains through delegation of local identifiers. This idea aims to reduce the complexities introduced by X.509 as the need for cross certification and enforcement of globally unique identifiers is mitigated.

SDSI attributes can be interpreted as access policies: In SDSI, authorization to access a resource is obtained when a collection of user attributes is in compliance with a requested resource security policy. A credential in SDSI consists of an authorization certificate called *auth cert*. The authors define $A.a \rightarrow B$ to denote that a local entity A issues the attribute a to a subject B. In turn, B itself is able to issue and further delegate the attribute a. In addition to the attribute delegation itself, SDSI allows to control the depth of delegation. For example, A is able to limit the number of times an attribute can be sub-delegated. Delegations in SDSI take the following form:

$$A\ bob \rightarrow B \tag{3.1}$$
$$A\ friends \rightarrow A\ bob \tag{3.2}$$
$$A\ friends \rightarrow A\ bob\ myfriends \tag{3.3}$$
$$B\ dave \rightarrow D \tag{3.4}$$
$$B\ carol \rightarrow C \tag{3.5}$$
$$B\ myfriends \rightarrow B\ dave \tag{3.6}$$
$$B\ myfriends \rightarrow B\ carol \tag{3.7}$$

Given the above examples, we can see that unlike X.509 or Domain Name System (DNS), there are no globally unique identifiers in SDSI such as DNs or FQDNs. Identifiers must be interpreted from within the local domain they are defined in. In order to discover the entity or entities identified by $Afriends$, the authors of SDSI define a chain discovery algorithm [25]. The algorithm translates each delegation into a "rewriting" rule and in turn treats chain discovery as a term-rewriting problem.

For example, the righthand term in 3.3 is first rewritten using the term in 3.1 into the new term $Afriends \rightarrow Bmyfriends$. This new term is rewritten using the terms 3.6 and 3.7 into $Afriends \rightarrow Bdave$ and $Afriends \rightarrow Bcarol$, respectively. Finally, combining the terms 3.4 and 3.5 we get the terms $Afriends \rightarrow D$ and $Afriends \rightarrow C$. In combination with term 3.1, we now successfully resolved the delegation of attribute $friend$ to the entities B, C and D. In general, term-rewriting is performed until no more rewriting rules can be applied. If the attribute is treated as an authorization policy, it might be sufficient if an attribute that satisfies a policy is found. For example, if we simply want to know if an entity is friend of A.

However, SDSI relies on the availability of the attribute delegation information prior to chain discovery. Resolution and distributed storage of attributes and credentials is considered out of scope in SDSI by its authors. Where X.500 defines Directory Access Protocol (DAP), no dedicated delivery mechanism for SDSI exists. However, SDSI certificates can be stored in a special DNS record type [72] which is the same record format that also supports X.509 and OpenPGP.

An example for a system that uses SDSI as a model for its resolution mechanics is the GNU Name System (GNS) [142]. The design of GNS is consistent with the concept of linking local identifiers and is designed as an alternative to the hierarchical DNS similar to how SDSI is an alternative to the hierarchical X.509.

An attribute delegation approach which is similar to SDSI is proposed by Li et al. [81]. The authors envision an authorization mechanism which is built on the concept of ABD and supply a *trust management language* RT_0. RT_0 allows entities to express different types of credentials as attributes. In order to model the attribute delegations, they propose the use of a *credential graph* which can be traversed using a variety of strategies. Similar to SDSI and its rewriting algorithm, finding a path in a credential graph from an attribute to an entity is a way to establish trust in an entity with respect to the attribute. RT_0 implicitly supports SDSI style delegations, since the authors claim that SDSI auth certs can be translated into RT_0 with the exception that the arrows are reversed in the notations.

At a glance, four types of attribute delegations are defined in RT_0:

$$A.a \leftarrow B \tag{Type 1}$$

$$A.a \leftarrow B.b \tag{Type 2}$$

$$A.a \leftarrow B.b.a \tag{Type 3}$$

$$A.a \leftarrow \bigcap_{i=1}^{n} f_i \tag{Type 4}$$

The similarities between RT_0 and SDSI become glaringly obvious by putting the notations side by side and the semantic equivalence of both approaches are evident: Type 1 delegations are interpreted as "A calls B a". It directly associates the identifier a with B within the local domain of A. Type 2 delegations are read as "all entities that

B calls b are called a by A". Type 3 delegations are used to express that "all entities that are called a by all entities that B calls b, are also called a by A". Finally, a type 4 delegation is interpreted as "A calls all entities a that satisfy each f_i" with f_i being any right-hand expression of either Type 1, Type 2 or Type 3.

In order to resolve delegations chains, Li et al. propose a chain discovery algorithm which defines how to build and traverse a credential graph [86]. The algorithm combines a backward search from attributes to the entities with a forward search from entities to the issued attributes into a bidirectional search. According to the authors, this approach is necessary in order to ensure that credential chains can be found even if storage of credentials is arbitrary. Arbitrary means here that it is possible to store delegations either at the issuer or subject in distributed storage architectures. However, chain discovery only succeeds as long as strict and complex constraints on the credential storage are enforced. Li et al. formalized this set of constraints into a type system which allows users to ensure what they call "well-typedness" of a RT_0 delegation. Unfortunately, especially in decentralized or even distributed scenarios, entities are unlikely to find a consensus where to store delegations as delegation information is always local and notoriously incomplete.

Differentiation The approaches taken by RT_0 and SDSI are partial solutions to solve the problem of making authorization decisions in distributed or decentralized scenarios. In this work, we show that the existing research in the area of ABD can be used in combination with a secure resolution mechanism based on name systems. In Chapter 5, we present a practical ABD authorization system by addressing the issue of delegation storage and resolution in a name system directory service. Further, we show how ABD can be combined with ABCs and use it to support trust establishment into attributes served through the self-sovereign identity system we propose in Chapter 4.

3.3.2. Privacy-preserving key transparency

A central problem in any PKI, is the establishment of trust in mappings between identifiers and public keys. Existing PKIs address this problem in various different ways: For example, Pretty Good Privacy (PGP) spans a "web of trust" consisting of trust relationships established ad-hoc between users. Trust is then calculated through transitive evaluation of relationships using trust metrics. The X.509 PKI for the Internet on the other hand requires the use of trust anchors commonly found in trust stores of operating systems or browsers. Unlike PGP, trust in X.509 is absolute and there is no metric that supports the continuous gradation of trust. What both PGP and X.509 share is that key material and mappings are public. This exposes trust relationships between entities and is a possible privacy concern.

The authors of CONIKS [93] address this concern by providing a privacy-preserving key verification service. The approach revolves around the idea of user-centric key transparency. The authors propose a system that does not require a single, centralized third-party to monitor and author mappings from names to keys, such as from an email address to its associated public key. Rather, CONIKS allows users and services to participate in a privacy-preserving protocol that allows them to audit providers of such mappings for non-equivocation. The authors design a data structure which consists of so-called *private indices* organized in a Merkle tree. Private indices are built using *verifiable random functions* (VRFs) which allow to verify if a specific entry exists withing the tree. The resulting objects are stored at IdPs and regularly queried by users. While the system is not designed to mitigate malicious IdPs, it is able to expose those that modify or withhold information.

The authors of ClaimChain [80] aim to mitigate the need for central IdPs while relying on the same building blocks as CONIKS. While ClaimClaim also primarily addresses the key verification use-case through the use of hash chains and cross-referencing, the authors claim that it could be used for any generic claim type. In order to support key propagation in fully decentralized settings, the authors propose that users exchange locally managed, user-specific ClaimChains though gossiping protocols. In this regard, the authors of ClaimChain follow the trust model of SDSI and Role-based Trust Management (RT). Trust evaluation is initiated locally using locally established relationships as trust anchors. This process continues until a statement regarding a specific key can be made.

Differentiation CONIKS relies on centralized IdPs which manage and serve key mappings. While users are enabled to detect misbehavior of IdPs, COINKS does not prevent attacks by or on these central services. This issue is partially addressed by ClaimChain, which accommodates flexible, decentralized data structures for key bindings.

However, ClaimChain does not address chain propagation outside of ad-hoc gossiping over out-of-band protocols. This is particularly problematic when it comes to offline access of claims. For example, missing chains are not resolvable through any mechanism if the authoritative user is offline and cannot be distinguished from expired chains. The authors of ClaimChain state that in this case, claims could be stored at online services. However, this would again introduce a centralized component and essentially degrade back into CONIKS. In Chapter 5, we propose to addresses this issue by building a transportation layer and respective protocols for offline access of trust relationships through name systems.

3.4. Summary

In this chapter we discussed relevant related work and the state of the art with respect to the context of this thesis. We identified shortcomings of existing approaches and research areas which we address in the remainder of this thesis as part of our contributions.

With respect to the broader scope of our initial research questions we can make the following observations:

- Self-sovereign identity systems are an effective mitigation of liability risks for IdPs and RPs alike.

- Self-sovereign identity systems empower users to take and exercise control over their personal data.

As part of our literature research, we also identified shortcomings of existing approaches to decentralized, self-sovereign identity systems: Providing the feature set and convenience of existing systems and services proves challenging. In particular, integration of effective, user-centered access control and offline availability of user data is an open issue. Further, existing approaches struggle with trust establishment in self-sovereign identity data while maintaining its decentralized, user-centered properties. In the following two chapters, we present our contributions which address our research questions while taking into account the shortcomings of the related work presented in this chapter.

CHAPTER 4

Decentralizing self-sovereign personal data sharing

In this chapter, we present re:claimID: Our approach to decentralized, self-sovereign personal data sharing. We argue that users must be in charge of assembling attributes and enforcing access control when selectively sharing them with relying parties. In our design, attributes may be either self-attested or verified and asserted by trusted third parties. Our goal is to enable users to reclaim authority and control over their digital identities in the spirit of informational self-determination. In order to address the issues of traditional identity management, we avoid relying on a centralized service for attribute sharing. For re:claimID, we introduce a mechanism that enables users to provision identity attributes in a way that facilitates non-interactive presentation as well as selective disclosure in a decentralized fashion.

In the following sections, we define an adversary model and present our design of re:claimID. We then discuss how the design of a decentralized self-sovereign identity service is able to satisfy a set of security properties which allow it to withstand attacks from our adversary. We propose the use of a name system as a distributed identity directory service. The name system allows relying parties (RPs) to asynchronously access attributes whenever needed even if the respective user is offline. re:claimID allows users to retain control over their identities at all times through the use of a decentralized directory service consisting of user-controlled namespaces. Users are enabled to selectively authorize RPs to access attributes while their authorization decisions are enforced through a cryptographic access control layer. We test the viability and practicality of our design on the basis of a prototype implementation on top of the GNU Name System (GNS) [142, 143]. As part of the implementation, we design an OpenID Connect 1.0 (OIDC) compatibility layer in order to facilitate standardized client implementations. This enables RPs to integrate re:claimID using off the shelf libraries. Apart from our final design, we also present derivative approaches that were conceived in the wake of this research effort which culminated in re:claimID.

The major contribution of this chapter, re:claimID, is based on the publication "re-claimID: Secure, Self-Sovereign Identities Using Name Systems and Attribute-Based

Encryption", published 2018 in the 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications (TrustCom) [114]. Earlier foundational work can be found in the position paper "Managing and Presenting User Attributes over a Decentralized Secure Name System" published 2016 in the 11h International Workshop on Data Privacy Management and Security Assurance (DPM) [123]. Finally, we published accompanying research [57] which includes use case and usability studies in the context of a proposal to an EU Horizon 2020 contest calling for seamless, privacy-preserving personal authentication for users [26].

4.1. Adversary model and security goals

In the following, we define the capabilities of an attacker in our adversary model. We take into account recent events and disclosures around mass surveillance and accompanying efforts by nation states as well as commercial interests which are potentially harmful to users' privacy. In addition, we define the security goals of our system in the form of desired security properties.

4.1.1. Adversary model

Contemporary attacker models must address the issue of nation-wide manipulation and surveillance of directory services such as the Domain Name System (DNS) [103, 41]. At the same time, data leaks and the surveillance of global service providers including social networking services and Identity Provider (IdP)s are a matter of fact today [30, 49, 66].

Hence, our adversary model must include attackers with the ability to collect any data in transit between all participants. This includes man-on-the-side as well as man-in-the-middle attacks, but also coercion of service providers into submission of data within the legal domain of the attacker [91]. We further assume the attacker is able to manipulate a limited but large number of nodes in the network. However, we assume that the attacker is not able to break cryptographic primitives that are considered secure by the research community today.

4.1.2. Security goals

In this section, we discuss security goals we aim to achieve in the context of user attribute provisioning and sharing while withstanding an adversary as defined above.

Availability: As we assume that a single entity can be coerced or forced to cooperate by powerful attackers such as rouge nation states, the first measure to ensure availability in the face of our adversary model is to not rely on a centralized service. This approach provides higher availability guarantees especially when confronted with powerful attacks as employed by some nation states such as the QUANTUM family of hacking operations [68].

In order to address this issue, we propose the use of a decentralized service for provisioning of user attributes. This service must include functionality which enforces authorization decisions made by the users. As the basis for a decentralized directory service, we propose the use of a name system. The user stores attributes in a name system acting as the main authority over the data. User attributes are stored in a distributed fashion and can be queried even if the user is not online. It should be noted that in particular peer-to-peer-based decentralized services suffer from some limitations in terms of availability: High node churn – e.g. if peer fluctuation is high – can be detrimental to data availability. This is a problem in our use case, as attributes should stay accessible to authorized RPs. To address this problem, a reliable name system with contingency mechanisms such as caching and data replication is required. It is thus important to take into account the security properties of available name systems as discussed in Section 2.1.2 for our prototype implementation of re:claimID. Further, aiming for security properties such as authenticity through the use of cryptography in distributed systems can often have a detrimental effect on availability. Herzberg et al. have shown how this is particularly true for DNS and its security extensions [62].

Authenticity: Regarding identity data authenticity, we must distinguish between data origin authenticity and third party data assertions. In self-sovereign identity systems, the data origin is always the user. The actual data may be *additionally* asserted by a third party. This assertion may be provided out-of-band or implicitly, for example through the use of cryptographic signatures.

To ensure data origin authenticity, the user manages identity attributes in his namespace. The records holding the attributes must be cryptographically signed using a private key of the user associated with the namespace. When resolving an attribute record, a RP is able to verify the record signature and with it that this value was stored by the namespace owner.

In some cases, the RP additionally requires that attributes are asserted by a trusted[1] third party. In re:claimID, we do not define the form factor of third party asserted attribute values. By default, attributes are self-asserted by the user, but the design is flexible enough to accommodate other formats. In general, the use of attribute-based credentials (ABCs) is possible – for example in the form of X.509 certificates as attribute values. Self-issued credentials, such as those defined by OpenID Connect in the sections on self-issued ID tokens [112] are also feasible. We dedicate Chapter 5 to the challenge of trust establishment and discuss possible privacy-preserving trust models and credential systems which complement re:claimID in more detail.

Privacy and Confidentiality: In our adversary model, an attacker is able to coerce centralized identity service providers into submission of all identity data. In order to address this fact, our approach must ensure confidentiality of attributes through the use of end-to-end encryption of user attributes. We propose the use of attribute-based encryption (ABE) as a flexible, reasonably efficient mechanism to implement this requirement in re:claimID. The user encrypts attributes before they are stored in the

[1]trusted by the relying party

name system and only authorized RPs are given the means to decrypt and read the data. For this, we design a cryptographic access control layer that enables users to authorize RPs to access subsets of attributes.

Through the encryption layer, we achieve two goals for re:claimID: First, it ensures confidentiality of user attributes in the otherwise public namespace of a name system. Second, it allows users to enforce access control decisions without having to rely on a trusted service provider. This is realized by having users issue individual decryption keys to each authorized RP. The use of ABE allows users to create keys where it is ensured that only the decryption of specific attribute sets is possible. We note that conventionally, ABE systems require a trusted third party key issuer that is in charge of issuing and managing decryption keys. However, such a setup would violate our decentralized design and make it susceptible to our adversary model. Instead, we propose a user-centered derivation: In re:claimID, every user acts as his own key issuer. This enables the user to exercise complete and exclusive control over all decryption keys and authorization decisions. We specifically note here that we consider compromised end-user devices out of scope. Research in the area of endpoint security can greatly increase resilience to attacks on devices. However, we consider a successful compromising attack on a single end-user device to be less impactful than the compromise of a central identity service. The former only exposes a single user while the latter potentially compromises the identities and personal data of millions of users.

In terms of privacy, coercion of a centralized IdP into straight up aiding and abetting in surveillance efforts is also an issue as it results in the attacker to have knowledge over all interactions between users and RPs. This allows the attacker to learn which services users are accessing over time. Hence, it is reasonable to ensure that access to user attributes by authorized parties cannot be trivially monitored by an attacker. We note that the cryptographic access control layer does not directly mitigate the issue of active and passive surveillance. However, the issue can be tackled through a security property that is found in selected name systems called “query and response privacy”. Name systems such as Namecoin and GNS offer such protections as discussed in Section 2.1.2. We take a closer look into this aspect as part of our prototype implementation in Section 4.4.1 later in this chapter.

Related to the aspect of attribute value attestations as discussed above is the fact that users might not be willing to disclose attribute values to RPs. This is a common use case for privacy-preserving credentials. We discuss possible approaches to present zero-knowledge credentials non-interactively as part of re:claimID in Chapter 5.

4.2. Towards a decentralized, self-sovereign identity service

We argue that truly self-sovereign identity management cannot be achieved through centralized IdPs. In order to address our research questions, we design a decentralized, self-sovereign identity management system which mitigates the identified shortcomings of related work as presented in Chapter 3. We divide our approach into two components we identified in the beginning of this thesis:

First, we present the design of a decentralized directory service. Our approach is to build this service on top of a decentralized peer-to-peer infrastructure in the form of a secure name system. Second, we design an attribute sharing mechanism which allows users to enforce access control on data stored in the directory service. To achieve this, we propose the use of an cryptographic access control layer.

In the following, we outline the high-level design of the above components which will serve as the basis for our main contribution: re:claimID.

4.2.1. Identity directory service

Secure name systems allow users to manage their identities and attributes without the need of a central service. Accordingly, RPs query the name system to retrieve user data. This approach does not require a trusted third party IdP. In the following, we discuss how name systems can be used as identity directories and how access control can be facilitated through cryptographic means.

Name systems as decentralized attribute storage

The integration of identity management with recent secure name systems has several advantages: As elaborated in Section 2.1, name systems fundamentally are directory services that are commonly used in the context of address resolution. However, it is in the nature of directory services that we can use the basic feature set of name systems in order to build an open identity directory. Depending on the characteristics of the name system, the resulting service is distributed or even decentralized.

In the following, we discuss a number of constraints and inherent properties of name systems when used as identity directories. Specifically, we observe the following equivalences and differences between name systems and traditional identity directories:

Namespaces: Name systems with security properties such as data origin authenticity allow for a close coupling between namespaces and cryptographic keys. This coupling allows us to make a connection from a unique cryptographic digital identity in the form of a public key to the namespace owner. Examples of name systems with respective properties include Namecoin, GNS as well as DNS Security Extensions (DNSSEC).

Attribute records: Namespaces contain mappings to sets of *resource records* with well-defined values associated with them. The semantics of a value are defined through a *record type*. In order to use a name system as an identity directory, thus we need to define record types which allow us to distinguish between identity attribute data and other records in the directory. As name systems accommodate for future uses and additional record types, the interpretation of record data itself is left to applications. For example, an identity record may contain structured data, such as X.509 certificates or simply plain text. Since we equate namespaces with identities this allows us to define *attribute records* with dedicated record types that hold identity information. Within a namespace, all *attribute records* are self-issued attributes of a user: While the values of such *attribute records* may also be asserted by a third party, the mapping itself is first and foremost asserted only by the namespace owner. This conforms with the idea of self-sovereign identity systems. We consider attribute records as the equivalent to attribute names and values in other identity systems, regardless of how they are stored and organized.

Open name and identity registration: A core property which influences whether the service is open to all users is *name registration*. In order to enable users to manage their identities in a self-sovereign fashion, name registration must be open to anyone and ideally possible through technical means only. Ideally, registration in the traditional sense is not required at all. This disqualifies DNS-style registration mechanisms which are commonly out-of-band, not to mention often bureaucratic and costly. Especially the last factor is particularly damning in the case of most Distributed Ledger Technology (DLT)-based name systems including Namecoin. While users may be willing to cover financial costs in exchange for privacy and security guarantees, acceptance of such systems is difficult to evaluate.

Name system security properties

In the following, we discuss how our adversary model influences the choice of name system according to the respective security properties it can exhibit. We presented relevant related work in this regard which formalizes security properties of name systems in Section 2.1.2. Most of the security goals presented in the respective surveys of the background chapter are relevant in the scope of re:claimID as well. In the following, we discuss how those properties impact re:claimID.

Protection against Man-in-the-Middle: Protection against Man-in-the-Middle attacks is commonly achieved through the use of *data origin authentication* in name systems. For a self-sovereign, decentralized identity directory service, data origin authentication is essential as it provides a RP with the assertions that a specific identity attribute is indeed provided by the user. Integrity of records in a name system is commonly ensured through the use of digital signatures over the resource records. If the name system does not provide data origin authentication, the directory service must allow users to establish secure bindings from identities to attribute records in other ways. If a name system based directory service implements attribute types which provide data origin authentication, such a system implicitly retrofits data origin authenticity to the name system.

Protection against zone walks: In order to achieve this property, the name system must ensure that an attacker is not able to enumerate all records in a namespace. In our context, this includes identity attributes. However, identity attribute names such as “email” are inherently easy to guess as they carry semantic meaning. This implies that they can be easily enumerated and their existence verified through a simple query. As such, protection against zone walks is only relevant if we are able to obfuscate attribute names in some way. Zone confidentiality could be used in order to implement access control through the use of shared secret names. However, for the design of re:claimID we opted to create a cryptographic access control layer which is independent of the underlying name system in order to make the design name system agnostic.

Protection against client observation: An important security goal for re:claimID is the protection against client observation through either a passive attacker in the network or the network operator itself in order to protect the users' privacy. *Query origin anonymity* in combination with *query and response privacy* can be employed to achieve this goal. Without this property, it is possible for an attacker to collect metadata that exposes which user interacts with a given RP. This metadata then allows the attacker to build usage profiles or follow up using advanced social engineering attacks. Monitoring of social movements as well as identity attributes is unwanted, as it exposes organizational structures and trust relationships.

Protection against censorship and denial of service: An insufficiently resilient name system is, like any service, subject to denial of service attacks. This includes legal attacks on the structure of the namespace. Consequently, properties such as *censorship resistance* and resilience against denial of service attacks are crucial for name systems. In [58], the authors argue that it in this regard it is important to consider not only the technological but also the organizational structure of the name system. Hierarchical name systems are particularly prone to legal attacks which aim to censor certain information in the namespace. Even DLT-based decentralized systems are potentially prone to such attacks [89, 51]. Unavailable user data can result in the disruption of data processing at RPs.

Confidentiality of records: Finally, as name systems are designed to provide name-record mappings, information which is published unencrypted in the name system is often considered to be considered public. It is not obvious why this should be the case, as name systems such as DNS at least try to offer so-called "zone confidentiality", a property that protects the namespace owner from attackers that attempt to enumerate all names and records. Given this property, records may be kept secret to anyone who does not know of the existence of the name a priori. Unfortunately, in the case of identity attributes the name to query is usually easy to guess. This becomes evident in the case of an email address, where the record name would likely simply be "email". Consequently, it is our task to protect the confidentiality of attribute record contents through other means independent of the name system.

On the basis of our discussions in the background chapter, we conclude that name systems such as GNS and Namecoin are potential candidates which fit into our adversary model and exhibit the security properties above with respective caveats. Hence, when it comes to our design of re:claimID, we concur with the analysis by the authors of [58] that peer-to-peer-based approaches, such as Namecoin or GNS, should be preferred over semi-centralized, distributed systems such as DNS.

At the same time, both are rather disruptive approaches which do not have the same level of maturity and pervasiveness as DNS. While we are aware that this can negatively impact usability and acceptance of a design such as re:claimID, adequate security guarantees are of paramount importance in this use case. Further, we have shown in user studies [57] that usability wise there is no difference between the use of DNS and GNS.

In our reference implementation of re:claimID, we have settled on the use of GNS and dive deeper into its design in Section 4.4.1.

4.2.2. Cryptographic access control for attributes

A remaining challenge that we still need to address after we settle on a name system in order to realize a directory service is how to facilitate access control on user attributes. The problem we must solve in this context is closely related to the problem of broadcast encryption [45], which is commonly associated with digital restrictions management [130] use cases. In a nutshell, we must provide a way for the user to enforce access control decisions on his data which is published in an open, shared directory: the name system. Since we cannot prevent access to the directory itself, we require a mechanism that ensures only qualified, authorized entities are able to read the stored attribute data.

Naively, we may approach this problem through the use of public key cryptography. For each authorized RP, a user could encrypt the data – in our case user attributes – with the respective public key. This would mean that if a user decides to share a subset of attributes with multiple requesting parties, specific ciphertexts for each requesting party must be published in the directory. More sophisticated approaches in the context of broadcast encryption include the use of specialized cryptographic approaches such as those proposed by Kogan et al. [78].

Our approach leverages advancements in the area of pairing-based cryptography (PBC) [77] which allow the creation of efficient ABE schemes in order to address the broadcast encryption problem in re:claimID. We consider ABE to be a particularly good match in our case as it allows us to use attributes for key and access policy creation.

Regarding ABE schemes, there are two distinct variations: Ciphertext-policy ABE (CP-ABE) and key-policy ABE (KP-ABE). In CP-ABE, a set of attributes is provided to derive a decryption key. In our case, this could be a set of user attribute keys such as "email". In order to encrypt a plaintext, we provide an encryption policy that limits decryption capability to those decryption keys that are derived from a matching attribute key set. In KP-ABE it is the other way around. Attributes keys are used to encrypt the plaintext and the policy is provided to derive an decryption key.

In re:claimID, ABE allows the user to publish encrypted identity attributes in the directory and thus prevent unauthorized access. In order to authorize an RP, the user derives an ABE decryption key and transfers it out-of-band to the RP. The RP can use this key to decrypt a subset of attributes that reside in the directory. At the time of authorization no additional data such as a ciphertext encrypted specifically for the RP needs to be be published by the user.

As mentioned above, ABE is commonly realized through the use of PBC such as the schemes proposed by Bethencourt et al. [12] as well as Agrawal et al. [4]. Unfortunately, PBC is notoriously expensive in terms of computing power even when compared to traditional public key cryptography. However, for name systems, we expect the delay introduced by network latency to dwarf the effect heavy cryptographic operations have on the system overall. We have confirmed this assumption through our experimental evaluations in Section 4.4.4. The only decision left with respect to ABE that we must make in the context of re:claimID is between KP-ABE and CP-ABE. Borgh et al. [17, 18] argue that the use of CP-ABE is more intuitive when encrypting a plaintext. The reason for this is that traditionally encryption keys are issued by a third party: the key authority. In the case of KP-ABE, this means that the encryption policy is set not by the encrypting entity, in our case the user, but by the key authority. For the user, it is then not obvious or transparent who has access to the plaintext. While semantically equivalent, the usage experience of the two schemes varies significantly when used with a third party key authority. In our design, which we present in the following sections, the encrypting entity and the key authority are reduced to a single entity: the user. The encryption policy is thus always transparent to the encrypting entity.

Consequently, the chosen ABE scheme can actually be considered independent from our high-level design. We expect that a change in ABE scheme or type that the system properties do not change and that essentially, it can be considered merely an implementation detail.

4.3. The re:claimID system

In this section, we present our core contribution of this chapter: The re:claimID system. We show how it is possible to combine a name system and ABE scheme to create a decentralized, self-sovereign identity service. Notably, we provide an answer to the question regarding self-sovereign access control on attributes stored in such a system. This includes enabling users to grant and revoke access authorizations of other parties to access their attributes.

Our design provides all typically required functions such as creation, retrieval, updates, and revocation of user identities and attributes in a decentralized fashion. As illustrated in Figure 4.1, re:claimID does not rely on any centralized component for attribute sharing and access control enforcement. The concept of re:claimID can be implemented on top of any name system, inheriting its security properties. re:claimID features an authorization layer using ABE on top of the name system to ensure confidentiality and to facilitate policy-based access control to user attributes.

The design of re:claimID aims to satisfy the following requirements:

- The user must be able to manage one or more identities including attributes. This must be possible through the use of an open system which prevents third parties to restrict its usage.

- The user must be able to selectively authorize RPs to access those identities. This includes fine-grained access to subsets of identity attributes.

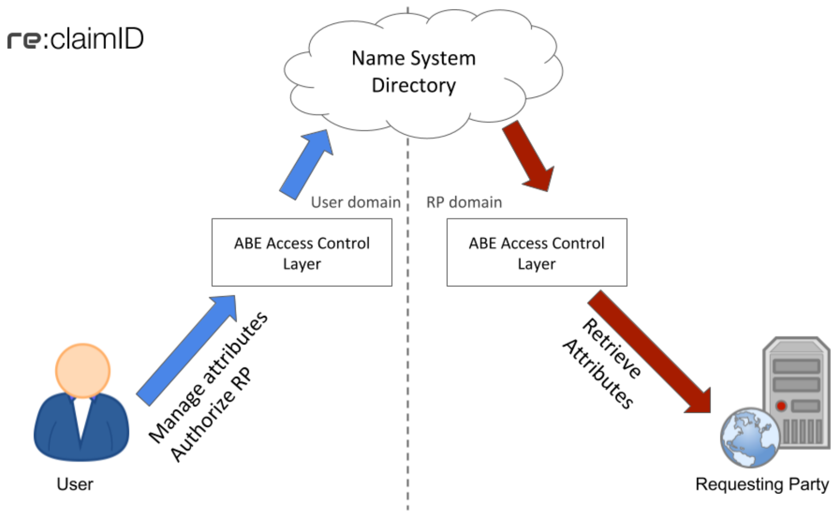


Figure 4.1.: The re:claimID system architecture.

- A RP must be able to request access to identity attributes of an end user. The attributes must be retrievable without a direct communication channel between the user and the RP.

In the following, we first define the basic building blocks of our design. Then, we provide an overview over the design and architecture of re:claimID and a discussion of high-level protocols and procedures.

4.3.1. Definitions and foundations

In this section, we formalize the two major building blocks of our design: Name systems and ABE schemes. We give a definition of the relevant procedures and primitives required for each and present a formal, high-level definition of name systems and relevant cryptographic schemes.

Cryptography

As discussed above, there are two major categories of ABE: Ciphertext-policy ABE (CP-ABE) and key-policy ABE (KP-ABE). In the context of our design both variants are equally suitable. For the sake of brevity and to provide consistency with our reference implementation, we assume in the following the use of CP-ABE. Further, we provide drop-in replacement definitions that would come into play for a KP-ABE-based approach to re:claimID in Appendix B.2.

Below, we define the high-level functions and objects for a CP-ABE scheme:

$$\begin{aligned}\mathbf{setup}_{\mathsf{ABE}}() &\rightarrow (msk, pk)\\ \mathbf{keygen}_{\mathsf{ABE}}(msk, \mathcal{A}) &\rightarrow sk\\ \mathbf{enc}_{\mathsf{ABE}}(pk, \mathcal{M}, \mathcal{P}) &\rightarrow \mathcal{C}_{\mathcal{P}}\\ \mathbf{dec}_{\mathsf{ABE}}(sk, \mathcal{C}) &\rightarrow \mathcal{M}\end{aligned} \tag{4.1}$$

In any ABE scheme, the key and attribute authority initially creates a master secret key *msk* and a public key *pk*. This creation is commonly referred to in the literature as "setup". Hence, we define the procedure $\mathbf{setup}_{\mathsf{ABE}}$ which yields both *msk* and *pk*.

The public key *pk* is used in conjunction with a policy $\mathcal{P}$ to encrypt a plaintext message $\mathcal{M}$ into a ciphertext $\mathcal{CC}_{\mathcal{P}}$ using the encryption function $\mathbf{enc}_{\mathsf{ABE}}$. In CP-ABE, $\mathcal{P}$ is commonly a boolean expression that specifies what combination of attributes are required to decrypt $\mathcal{CC}_{\mathcal{P}}$ using the decryption function $\mathbf{dec}_{\mathsf{ABE}}$. Hence, $\mathcal{P}$ is a policy that is bound to the ciphertext $\mathcal{CC}_{\mathcal{P}}$. The decryption key *sk* is derived from the *msk* and a set of attributes $\mathcal{A}$ using the $\mathbf{keygen}_{\mathsf{ABE}}$ procedure. In most schemes decryption keys are referred to as "user keys".

In addition to the ABE scheme above, we define a simple public key scheme: Let the private key *d* and the public key *e* be a key pair of an asymmetric cryptographic scheme. The respective encryption and decryption functions are **enc** and **dec**, respectively:

$$\begin{aligned}\mathbf{keygen}() &\rightarrow (d, e)\\ \mathbf{enc}(e, \mathcal{M}) &\rightarrow \mathcal{C}\\ \mathbf{dec}(d, \mathcal{C}) &\rightarrow \mathcal{M}\end{aligned} \tag{4.2}$$

Using the above two schemes and key definitions, we define a user identity key set $\mathcal{I}$ as:

$$\mathcal{I} := (msk_{\mathcal{I}}, pk_{\mathcal{I}}, e_{\mathcal{I}}, d_{\mathcal{I}}) \tag{4.3}$$

In the following, we use the identity key set as part of user identities in the relevant procedures.

Name system

We already established that name systems consist of namespaces that are owned and controlled by users or legal entities. Further, name systems provide users with a storage and retrieval mechanism for self-issued attributes. We define a name system to consist of the following procedures:

$$\begin{aligned}\mathbf{create}(\mathcal{I}) &\rightarrow \mathcal{N}_{\mathcal{I}}\\ \mathbf{seek}(e_{\mathcal{I}}) &\rightarrow \mathcal{N}_{\mathcal{I}}\\ \mathbf{resolve}(\mathcal{N}_{\mathcal{I}}, name) &\rightarrow \mathcal{R}\\ \mathbf{publish}(\mathcal{N}_{\mathcal{I}}, name, \mathcal{R}) &\rightarrow \mathcal{N}'_{\mathcal{I}}\\ \mathbf{depublish}(\mathcal{N}_{\mathcal{I}}, name) &\rightarrow \mathcal{N}'_{\mathcal{I}}\end{aligned} \tag{4.4}$$

In our design, name systems must establish a mapping between the user identity key tuple $\mathcal{I}$ and the namespace $\mathcal{N}_{\mathcal{I}}$. Commonly, this is realized through the use of digital signatures using public-key cryptography. We define **create** to be a procedure that creates a namespace $\mathcal{N}_{\mathcal{I}}$ in the name system and cryptographically binds it to a user identity key tuple $\mathcal{I}$. $\mathcal{N}_{\mathcal{I}}$ represents the state of the namespace and is initially empty. It is populated with resource records using the **publish** procedure. This binding ensures that any resource record $\mathcal{R}$ in $\mathcal{N}_{\mathcal{I}}$ is signed using the private key $d_{\mathcal{I}}$. We define a resource record $\mathcal{R}$ as:

$$\mathcal{R} := (type, payload, ttl) \tag{4.5}$$

The *type* of a resource record determines the semantics of the *payload*. It helps the resolving entity to interpret the contents of the resource record. For example, a common type in DNS is "A", indicating that the payload contains an IP address. The *ttl* specifies the duration of validity. This is used by resolvers and caches in the name system to verify the freshness of the resource record.

The **seek** procedure maps an identity public key $e_{\mathcal{I}}$ to the respective created namespace if it exists. We define the procedure **publish** to add $\mathcal{R}$ to the namespace $\mathcal{N}_{\mathcal{I}}$. This allows users and RPs to resolve the record under the specified *name* using the **resolve** procedure. We assume that the resolver verifies the signatures provided with $\mathcal{R}$ using the respective public key $e_{\mathcal{I}}$ which is associated with the namespace $\mathcal{N}_{\mathcal{I}}$. Finally, in order to support deletion of a published record $\mathcal{R}$, we define the procedure **depublish**.

We note that resolution requires both *name* as well as $\mathcal{N}_{\mathcal{I}}$. Name system resolvers usually query for names inside namespaces. Internal delegation mechanisms in the name system may trigger the resolver to successively resolve in a number of namespaces: For example, in DNS a "fully qualified domain name" (FQDN) is resolved by iteratively trying to find the authoritative namespace for a specific name. This is commonly hidden from the user but relevant in our design and hence we include it in our definitions.

4.3.2. Overview

In this section, we first identify and formalize the relevant high-level procedures and objects required in our self-sovereign identity system re:claimID. We define our identity system to consist of the following procedures:

$$\begin{aligned}
\mathbf{register}() &\rightarrow \mathcal{U} \\
\mathbf{deregister}(\mathcal{U}) &\rightarrow \perp \\
\mathbf{add}(\mathcal{U}, a) &\rightarrow \mathcal{U}' \\
\mathbf{update}(\mathcal{U}, a) &\rightarrow \mathcal{U}' \\
\mathbf{remove}(\mathcal{U}, a) &\rightarrow \mathcal{U}' \\
\mathbf{authorize}(\mathcal{U}, e_{\mathcal{RP}}, \mathcal{A}_{\mathcal{T}}) &\rightarrow (\mathcal{U}', \mathcal{T}_{\mathcal{RP}}) \\
\mathbf{revoke}(\mathcal{U}, \mathcal{T}) &\rightarrow \mathcal{U}' \\
\mathbf{retrieve}(\mathcal{RP}, \mathcal{T}_{\mathcal{RP}}) &\rightarrow \mathcal{A}_{\mathcal{T}}
\end{aligned} \tag{4.6}$$

The user initially bootstraps the namespace and identity key tuple using the registration procedure **register**. Further, we define the procedure **deregister** which allows users to delete their identity from the service. From now on, we refer to a re:claimID user identity as $\mathcal{U} = (\mathcal{I}_{\mathcal{U}}, \mathcal{N}_{\mathcal{U}}) \leftarrow \textbf{register}()$ and a re:claimID RP as $\mathcal{RP} = (\mathcal{I}_{\mathcal{RP}}, \mathcal{N}_{\mathcal{RP}}) \leftarrow \textbf{register}()$.

The **add** procedure allows a user to store an attribute a for the respective user identity $\mathcal{U}$ in the directory. Correspondingly, the **delete** procedure removes the attribute from the directory. We define an identity attribute a as follows:

$$a := (key, value, version) \tag{4.7}$$

The *key* is the attribute identifier. An example for an identity attribute key is "email". The attribute further contains a *value*. A semantically fitting value for the above key could be "john@doe.com". The value can contain other, more complex data structures as well. Besides plain text, attribute-based credentials issued by third parties is another conceivable option. We discuss possible form factors of privacy-preserving credentials in Chapter 5. Finally, the attribute also contains a monotonic number *version*. The version is crucial for access revocation through re-encryption which we discuss as part of the respective procedure in the following.

In order to authorize an RP to access a set of attributes $\mathcal{A}$, we define the **authorize** procedure. After authorization, the user can revoke access using the **revoke** procedure.

The result of an authorization procedure is a ticket $\mathcal{T}_{\mathcal{RP}}$ as well as a modified user identity $\mathcal{U}' = (\mathcal{I}_{\mathcal{U}}, \mathcal{N}'_{\mathcal{U}})$. A ticket is the proof and reference of a user authorization. We assume that through an out-of-band protocol, the user transfers the ticket to the respective authorized RPs. The $\mathcal{RP}$ in turn uses the ticket $\mathcal{T}_{\mathcal{RP}}$ to retrieve the shared attributes using the "**retrieve**" procedure. We define a ticket as:

$$\mathcal{T}_{\mathcal{RP}} := (e_{\mathcal{U}}, e_{\mathcal{RP}}, \mathcal{A}_{\mathcal{T}}, n_{\mathcal{T}}) \tag{4.8}$$

$\mathcal{T}_{\mathcal{RP}}$ contains the public key $e_{\mathcal{U}}$ of the authorizing identity $\mathcal{U}$ as well as $e_{\mathcal{RP}}$. It also includes the set of attributes $\mathcal{A}_{\mathcal{T}}$ that $\mathcal{U}$ authorized $\mathcal{RP}$ to access. In an initial exchange, the ticket may either contain a full set of attributes including attribute values or only the attribute keys. However, in order to avoid needlessly introducing complexity, we will work with the same definition of the set of attributes $\mathcal{A}_{\mathcal{T}}$ in the ticket as we do with the set that is used in the authorization procedure. Finally, the ticket nonce $n_{\mathcal{T}}$ is found in $\mathcal{T}_{\mathcal{RP}}$. We use $n_{\mathcal{T}}$ as a name to store the user key $sk_{\mathcal{RP}}$ in the namespace $\mathcal{N}_{\mathcal{U}}$. As elaborated above, we expect the user to transfer $\mathcal{T}_{\mathcal{RP}}$ in an out-of-band authorization process to the RP. Within such a process, the user is expected to prove ownership of the private key corresponding to $\mathcal{U}$. For example, through the use of public-key authentication. We propose such a mechanism on top of OIDC as part of our implementation in Section 4.4.2.

4.3.3. Registration

The following procedure defines how users and RPs register at the identity service:

```
Procedure register
    output: User identity $\mathcal{U}$
1   $(d_{\mathcal{I}}, e_{\mathcal{I}}) \leftarrow$ keygen();
2   $(msk_{\mathcal{I}}, pk_{\mathcal{I}}) \leftarrow$ setup_ABE();
3   $\mathcal{I}_{\mathcal{U}} \leftarrow (msk_{\mathcal{I}}, pk_{\mathcal{I}}, d_{\mathcal{I}}, e_{\mathcal{I}})$;
4   $\mathcal{N}_{\mathcal{U}} \leftarrow$ create($\mathcal{I}$);
5   return $\mathcal{U} = (\mathcal{I}_{\mathcal{U}}, \mathcal{N}_{\mathcal{U}})$;
```

In lines 1-2, we set up the ABE scheme and instantiate the identity key tuple. In line 3, the keys are assembled into $\mathcal{I}_{\mathcal{U}}$. In line 4, we create a namespace $\mathcal{N}_{\mathcal{U}}$ for the identity. Finally, the re:claimID identity $\mathcal{U}$ is returned.

4.3.4. Adding and updating attributes

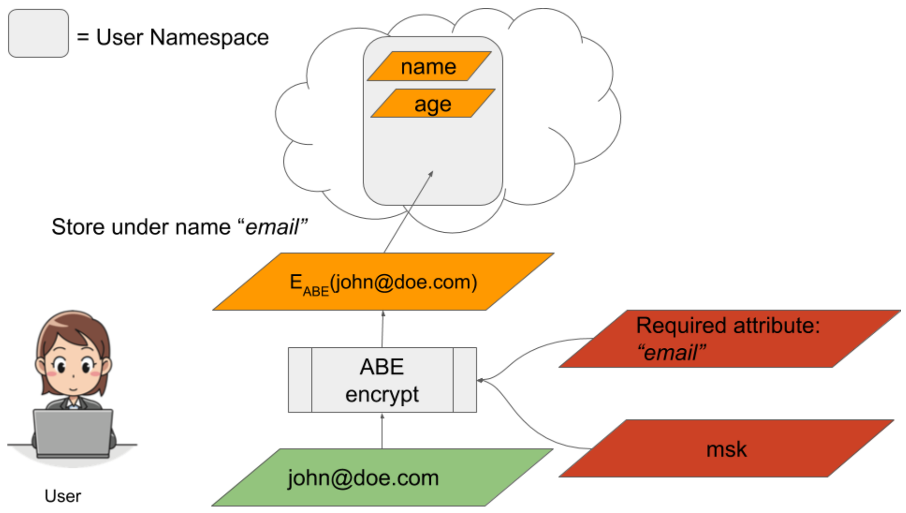


Figure 4.2.: A user adds an identity in re:claimID.

The user adds an attribute to an identity $\mathcal{U}$ by creating a mapping between the attribute key and the attribute value through a resource record $\mathcal{R}$. The user encrypts $\mathcal{R}$ using ABE with a policy $\mathcal{P}$ and stores it in the name system. Figure 4.2 illustrates this process. Respectively, we define the procedure "add/update".

In line 1, we derive the ABE access policy $\mathcal{P}$ for an attribute $a = (key, value, version)$ from the attribute *key* and *version*. This derivation allows us to efficiently revoke ABE keys by incrementing the version of attributes later. The derivation function could be a simple concatenation of the two values or even an XOR operation. In our implementation, we opted for a string concatenation. The semantics of $\mathcal{P}$ are that in order to decrypt $\mathcal{C}_{\mathcal{P}}$, a key associated with an attribute representing the attribute key in the respective version is required.

```
Procedure add/update
    input  : User attribute $a = (key, value, version)$
             User identity $\mathcal{U} = (\mathcal{I}_\mathcal{U}, \mathcal{N}_\mathcal{U})$
    output : Updated user identity $\mathcal{U}'$
1 $\mathcal{P} \leftarrow key \oplus version$;
2 $(msk_\mathcal{U}, pk_\mathcal{U}, e_\mathcal{U}, d_\mathcal{U}) \leftarrow \mathcal{I}_\mathcal{U}$;
3 $\mathcal{C}_\mathcal{P} \leftarrow \texttt{enc}_{\mathsf{ABE}}(pk_\mathcal{U}, \mathcal{M} = value, \mathcal{P})$;
4 $\mathcal{R} \leftarrow (type =' ID\_ATTR', \mathcal{C}_\mathcal{P}, ttl = 1h)$;
5 $\mathcal{N}'_\mathcal{U} \leftarrow \texttt{publish}(\mathcal{N}_\mathcal{U}, key, \mathcal{R})$;
6 $\mathcal{U}' \leftarrow (\mathcal{I}_\mathcal{U}, \mathcal{N}'_\mathcal{U})$;
7 return $\mathcal{U}'$;
```

In line 2-3, we encrypt the attribute value with the policy $\mathcal{P}$ and the ABE public key $pk_\mathcal{U}$ of our re:claimID identity $\mathcal{U}$. The ciphertext $\mathcal{C}_\mathcal{P}$ is then published as a resource record, modifying the namespace $\mathcal{N}_\mathcal{U}$ with a mapping to the attribute key. The updated user identity $\mathcal{U}'$ is returned.

We define the *type* of $\mathcal{R}$ as "ID_ATTR". The wire format of this resource record can be found in Appendix B.1 Figure B.1. The record type allows the resolver to distinguish identity attribute records from other, unrelated records such as IP addresses. In our design, all attribute resource records must have this type set. Table 4.1 illustrates the namespace of a user after the addition of attributes.

Name	Type	Value
email	`ID_ATTR`	enc_{ABE}(pk, "alice@example.com", "email:0")
full_name	`ID_ATTR`	enc_{ABE}(pk, "Alice Doe", "full_name:0")
dob	`ID_ATTR`	enc_{ABE}(pk, "1.3.1987", "dob:1")

Table 4.1.: An example namespace for a user in re:claimID.

We also note here that records in name systems expire through the use of the *ttl*. In the above procedure, we explicitly set the *ttl* to one hour. However, an implementation of re:claimID must specify an appropriate expiration time. Whether or not a *ttl* is appropriate heavily depends on choice of name system. For example, the *ttl* must be chosen in a way that allows to efficiently make use of the respective caching mechanism in the name system.

In order to update an attribute, the user simply adds the updated attribute. Hence, the **update** procedure is very similar to the **add** procedure. The attribute keys and versions used to encrypt the attribute record remain unchanged. This allows previously authorized RPs to be able to decrypt the updated record data with their existing keys. The update only takes effect *after* the attribute record expires and only then will the updated value be available to authorized parties. We specifically note here that a call of

delete before **add** in the case of an update is problematic: As we will establish in the following sections, the method we use for access revocation impacts the procedures for deletion as well. Deletion potentially triggers unnecessary re-authorizations of RPs. This entails re-encryption of attributes and re-issuance of ABE secret keys which is something we aim to avoid.

4.3.5. Authorizing access

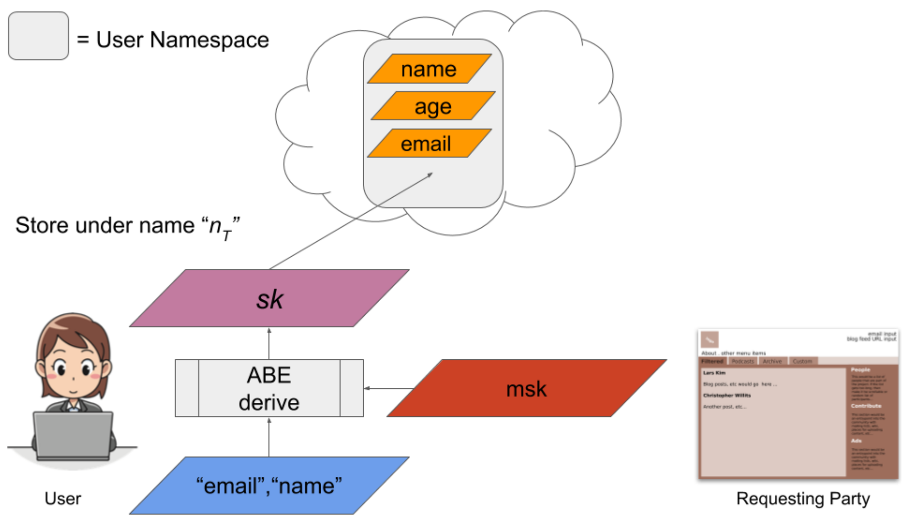


Figure 4.3.: A user authorizes an RP in re:claimID.

For a RP to be able to access user attributes, the user must first grant this access. In the following, we discuss the authorize procedure. In our design, granting access involves the user to derive an ABE key sk. This key allows a RP to resolve any attribute $a \in \mathcal{A}$ through the name system from the user namespace $\mathcal{N}_{\mathcal{U}}$ and decrypt its contents. Cryptographically, the key sk is associated with all attribute keys in $\mathcal{A}$ with the respective versions.

We expect the user to provide sk to the RP via the name system. The user publishes sk under a shared secret name. The shared secret name is a random nonce n which the user generates. The resource record R_{sk} that is published in the end user identity namespace contains sk encrypted with the public key $e_{\mathcal{RP}}$ of the RP. For this reason, $n_{\mathcal{T}}$ is included in the authorization ticket $\mathcal{T}$. This additional indirection allows the end user to update sk whenever necessary such as in case of key rotations. Figure 4.3 illustrates how a user generates sk and stores it in the name system. We define the "authorize" procedure accordingly.

First, the user derives the ABE key $sk_{\mathcal{RP}}$ from his master secret key $msk_{\mathcal{U}}$. The key is derived to the attribute keys in $\mathcal{A}_{\mathcal{T}}$ which the RP shall be granted access to. Further, we encrypt the key using the RP public key $e_{\mathcal{RP}}$, resulting in the ciphertext $\mathcal{C}$. As discussed above, a random nonce $n_{\mathcal{T}}$ is generated under which the ciphertext $\mathcal{C}$ is published in

```
Procedure authorize
    input : User identity $\mathcal{U} = (\mathcal{I}_\mathcal{U}, \mathcal{N}_\mathcal{U})$
            Requesting party public key $e_{\mathcal{RP}}$
            Set of attributes to share $\mathcal{A}_\mathcal{T}$
    output: A ticket $\mathcal{T}_{\mathcal{RP}}$ and an updated identity $\mathcal{U}'$
 1 $(msk_\mathcal{U}, pk_\mathcal{U}, e_\mathcal{U}, d_\mathcal{U}) \leftarrow \mathcal{I}_\mathcal{U}$;
 2 $sk_{\mathcal{RP}} \leftarrow \texttt{keygen}(msk_\mathcal{U}, \mathcal{A}_\mathcal{T})$;
 3 $\mathcal{C} \leftarrow \texttt{enc}(e_{\mathcal{RP}}, sk_{\mathcal{RP}})$;
 4 $n_\mathcal{T} \leftarrow_R \mathbb{R}$;
 5 $\mathcal{R} \leftarrow (type =' ABE\_KEY', \mathcal{C}, ttl =' 1h')$;
 6 $\mathcal{N}'_\mathcal{U} \leftarrow \texttt{publish}(\mathcal{N}_\mathcal{U}, n_\mathcal{T}, \mathcal{R})$;
 7 $\mathcal{T}_{\mathcal{RP}} \leftarrow (e_\mathcal{U}, e_{\mathcal{RP}}, \mathcal{A}_\mathcal{T}, n_\mathcal{T})$;
 8 $\mathcal{U}' \leftarrow (\mathcal{I}_\mathcal{U}, \mathcal{N}'_\mathcal{U})$;
 9 return $(\mathcal{U}', \mathcal{T}_{\mathcal{RP}})$;
```

the user namespace $\mathcal{N}_\mathcal{U}$. The set of published key records allow the user to keep track of and modify authorizations. The user public key $e_\mathcal{U}$, the RP public key $e_{\mathcal{RP}}$, the nonce $n_\mathcal{T}$ and the attributes $\mathcal{A}_\mathcal{T}$ are assembled into a ticket $\mathcal{T}_{\mathcal{RP}}$. We note here that having the attribute set $\mathcal{A}_\mathcal{T}$ seems unnecessary because the RP is supposed to be able to retrieve $\mathcal{A}_\mathcal{T}$ from the name system. There is a simple reason why we include $\mathcal{A}_\mathcal{T}$ in the ticket: As the out-of-band transfer of $\mathcal{T}_{\mathcal{RP}}$ to the RP presumably occurs during an authorization request, it is reasonable to assume that the RP is in acute need of the attributes. In future occasions where the RP requires fresh attributes, it must retrieve them from the name system as it is likely that the interactive session with the user has ended by then.

For the same reasons as discussed above with attribute records, we specify a unique record type for ABE keys: "ABE_KEY". The wire format can be found in Appendix B.1 Figure B.3. Table 4.2 illustrates the namespace of a user after an authorization.

Name	Type	Value
email	`ID_ATTR`	enc_{ABE}(pk, "alice@example.com", "email:0")
dob	`ID_ATTR`	enc_{ABE}(pk, "1.3.1987", "dob:1")
n	`ABE_KEY`	$enc(e_{\mathcal{RP}}, sk_{email,dob})$

Table 4.2.: An example namespace containing authorization tickets of a user in re:claimID.

4.3.6. Retrieval

Once authorized, a RP can retrieve a set of attributes by performing lookups in the name system. However, before resolving attributes the RP must use the nonce $n_\mathcal{T}$ from the authorization ticket $\mathcal{T}_{\mathcal{RP}}$ to retrieve its ABE key $sk_{\mathcal{RP}}$. Only then is it possible for the RP to decrypt attribute records. The RP can only decrypt attribute records of attributes that are also contained in $\mathcal{T}_{\mathcal{RP}}$. Figure 4.4 illustrates this process. Accordingly, we define the procedure "retrieve".

```
Procedure retrieve
    input  : Requesting party $\mathcal{RP} = (\mathcal{I}_{\mathcal{RP}}, \mathcal{N}_{\mathcal{RP}})$
             Ticket $\mathcal{T}_{\mathcal{RP}} = (e_{\mathcal{U}}, e_{\mathcal{RP}}, \mathcal{A}_{\mathcal{T}}, n_{\mathcal{T}})$
    output : Attributes $\mathcal{A}'_{\mathcal{T}}$
1  $\mathcal{N}_{\mathcal{U}} \leftarrow \texttt{seek}(e_{\mathcal{U}})$;
2  $(type, esk, ttl) \leftarrow \texttt{resolve}(\mathcal{N}_{\mathcal{U}}, n_{\mathcal{T}})$;
3  $(msk_{\mathcal{RP}}, pk_{\mathcal{RP}}, e_{\mathcal{RP}}, d_{\mathcal{RP}}) \leftarrow \mathcal{I}_{\mathcal{RP}}$;
4  $sk_{\mathcal{RP}} \leftarrow \texttt{dec}(d_{\mathcal{RP}}, esk)$;
5  $\mathcal{A}'_{\mathcal{T}} \leftarrow \emptyset$;
6  for all $(key, value, version) \in \mathcal{A}_{\mathcal{T}}$ do
7      $(type, evalue, ttl) \leftarrow \texttt{resolve}(\mathcal{N}_{\mathcal{U}}, key)$;
8      $\mathcal{A}'_{\mathcal{T}} \leftarrow \mathcal{A}'_{\mathcal{T}} \cup (key, \texttt{dec}_{\mathsf{ABE}}(sk_{\mathcal{RP}}, evalue), version)$;
9  end
10 return $\mathcal{A}'_{\mathcal{T}}$;
```

We note that the for-loop found in this procedure which is used to resolve attribute records can be executed in parallel. As this includes the network heavy **resolve** call, parallelization of this loop can be advantageous depending on the underlying name system resolution mechanism.

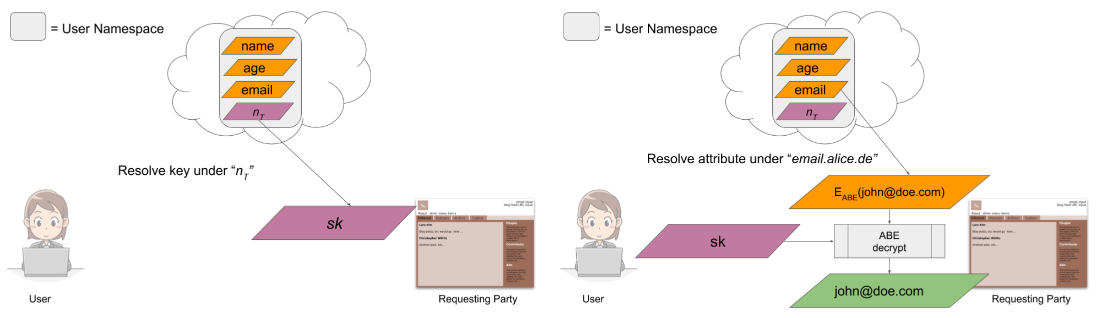


Figure 4.4.: Left: A RP retrieves an ABE key from re:claimID.
Right: A RP retrieves and decrypts attributes from re:claimID.

4.3.7. Revoking access

In re:claimID, revocation of access rights to a set of attributes $\mathcal{A}$ is realized by updating the ABE encryption policy $\mathcal{P}$ on the attribute values.

We define revocation as the process to revoke access of a specific RP to a set of user attributes $\mathcal{A}$ in re:claimID. Unfortunately, revocation schemes for ABE such as [98, 95] are often quite complex and inefficient and mitigations such as [83] effectively just modulate the space-time tradeoffs. This issue is amplified in our case as we also have to take into account the resulting key redistribution via the name system.

Hence, we want to minimize use of computationally heavy cryptographic revocation mechanisms as well as unnecessary re-keying. In this context, we point out two characteristics in our design: For one, any re-keying and re-encryption of data is essentially happening locally on a users machine. Also, revocation is not a time critical issue; If the access of a RP is revoked, we only have to ensure that access to fresh, potentially updated data is no longer possible. Such is the nature of any system that enforces access control but not usage control. Hence, for already published data we can rely on cache expiration in the name system.

In technical terms, a user revokes access to an attribute $a \in \mathcal{A}$ record by preventing its decryption with the ABE key sk already issued to the RP. The user must assume that attribute values that were accessible to the RP in the past have been retrieved and possibly stored by the RP locally. Consequently, from the point of revocation on the RP must be prevented from two things: First, the RP must be unable to retrieve freshness guarantees regarding the attribute value, i.e. "is this data still correct". Second, the RP must be unable to retrieve updates or modifications to the attribute value, i.e. a changed email address. To achieve those two goals, we propose the use of ABE attribute versions.

Attribute versioning When encrypting an attribute using ABE, instead of just using a simple attribute key in the policy, we concatenate the attribute key with the version. This requires us to keep track of attribute versions across revocations.

When revoking access of a RP to an attribute a, we increment the attribute version. From that time on, all attribute records $\mathcal{R}$ that contain the attribute value are encrypted using the updated attribute version and key combination. This prevents the RP from using an old sk to decrypt the data.

A problem that arises from this approach is that other RPs that are still authorized to access attribute a require new keys which reflect the updated attribute versions. Naively, we could simply setup new ABE key pairs (pk, msk), re-encrypt all attribute record payloads and issue new keys for all authorized RPs. However, the user only needs to create new ABE keys for the affected RPs. Instead of initiating a direct communication channel with all of those RPs, he publishes updated key records under the name $n_{\mathcal{T}}$. It is important the $n_{\mathcal{T}}$ does not change from one key to another when we perform re-keying so that the RPs may use the same ticket $\mathcal{T}$ they were given in the initial authorization to retrieve and decrypt the user attributes. This limits the group that require updated ABE keys to those RPs that were authorized to at least one attribute that the revoked RP also has access to. We define our revoke procedure accordingly.

Alternative 1: Re-bootstrapping

As mentioned above, a naive revocation approach is to re-bootstrap the ABE system. After creating a fresh key pair (pk, msk) and re-encrypting the attribute values, we

```
Procedure revoke
    input  : User identity $\mathcal{U}$
             Ticket $\mathcal{T}_{\mathcal{RP}} = (e_{\mathcal{U}}, e_{\mathcal{RP}}, \mathcal{A}_{\mathcal{T}}, n_{\mathcal{T}})$
    output : Updated identity $\mathcal{U}'$
$\mathcal{U}' \leftarrow \mathcal{U}$;
for each attribute $a = (key, value, version) \in \mathcal{A}_{\mathcal{T}}$ do
    $a' \leftarrow (key, value, version + 1)$;
    $\mathcal{U}' \leftarrow$ update($\mathcal{U}'$, $a'$);
end
for each ticket $\mathcal{T}_i \neq \mathcal{T}_{\mathcal{RP}}$ issued by $\mathcal{U}$ do
    $(e_{\mathcal{U}}, e_{\mathcal{T}_i}, \mathcal{A}_{\mathcal{T}_i}, n_{\mathcal{T}_i}) \leftarrow \mathcal{T}_i$;
    if $\emptyset \neq \mathcal{A}_{\mathcal{T}_i} \cap \mathcal{A}_{\mathcal{T}}$ then
        $A_{\mathcal{T}*} \leftarrow \{a = (key, value, version) \mid a \in \mathcal{A}_{\mathcal{T}_i}\}$;
        $(\mathcal{U}', \mathcal{T}_i') \leftarrow$ authorize($\mathcal{U}'$, $e_{\mathcal{T}_i}$, $A_{\mathcal{T}*}$);
    end
end
return $\mathcal{U}'$;
```

republish all attribute records $\mathcal{R}$. All keys that were previously issued by the user and published in the name system are then void and can no longer be used to decrypt attributes. We must then replace the RP decryption keys by generating new ABE key pairs. Of course, we omit generating a new key for the RP for which the user wants to revoke access.

This approach comes with a few drawbacks: First, it is unlikely that the revoked RP has access to all attributes. At the same time, it is possible that there are other RPs which do not have access to attributes that the revoked RP has access to. As we want to minimize the number of attributes we have to re-encrypt as much as possible, the approach is detrimental to this goal. Further, we want to avoid recreating keys for RPs which are completely independent of the revoked RP key in terms of attributes. The ultimate goal must be to limit the induced updates over the name system that require RPs to retrieve updated key material. On the other hand, the above approach with attribute versioning achieves both goals quite well.

Alternative 2: RP-specific attributes

Another alternative is to change the semantics of the key and policy attributes: For example, by using RP-specific attributes for the ciphertext policy $\mathcal{P}$. We encrypt the value of an attribute with key "email" using a policy $\mathcal{P}$ defined as:

$$\mathcal{P} \leftarrow \mathcal{I}_{\mathcal{RP}1} \vee \mathcal{I}_{\mathcal{RP}2} \vee \mathcal{I}_{\mathcal{RP}3} \tag{4.9}$$

This example policy allows the RPs $\mathcal{I}_{\mathcal{RP}1}$, $\mathcal{I}_{\mathcal{RP}2}$ and $\mathcal{I}_{\mathcal{RP}3}$ to decrypt the attribute record. When the user revokes the access of $\mathcal{I}_{\mathcal{RP}2}$, the attribute record payload is re-encrypted using an updated policy $\mathcal{P}'$:

$$\mathcal{P}' \leftarrow \mathcal{I}_{\mathcal{RP}1} \vee \mathcal{I}_{\mathcal{RP}3} \tag{4.10}$$

Revocation in this manner limits the number of cryptographic operations to one: the re-encryption of the attribute record payload. However, a problem emerges when access to the attribute is granted to a new RP $\mathcal{I}_{\mathcal{RP}4}$: When $\mathcal{I}_{\mathcal{RP}4}$ requests access to the attribute, the policy $\mathcal{P}''$ must be extended accordingly:

$$\mathcal{P}'' \leftarrow \mathcal{I}_{\mathcal{RP}1} \vee \mathcal{I}_{\mathcal{RP}3} \vee \mathcal{I}_{\mathcal{RP}4} \tag{4.11}$$

The re-encrypted attribute record is then published in the name system. At this point, the name system caches still contain the old attribute record encrypted using $\mathcal{P}'$ or even $\mathcal{P}$ until they expire. In other words: Immediately after authorization through the user, the RP will most likely be unable to retrieve the correct decryption key and decrypt the attribute record payload. In our use case the initial authorization is almost always initiated due to an imminent need for attributes by the RP. Hence, we consider this approach to be inadequate.

4.3.8. Deletion and update

The remaining operations and procedures on our design are the deletion and update of attributes. In order to delete an attribute, we must ensure the following:

First, the attribute must be removed from the name system. In most name systems, this means that the records are simply no longer re-published when they expire. Hence, the time it takes to delete an attribute from the directory is bound to the *ttl* of the respective resource record $\mathcal{R}$.

Second, we must make sure that relying parties that were previously authorized to access this attribute are prohibited from accessing it in the future. The latter task is crucial. If it is omitted unwanted side-effects occur when the attribute is re-added at a later time. Then, RPs authorized in the past may suddenly have retroactive access to the attribute. To prevent this, the attribute version must be incremented. This is counterintuitive, as we want to delete the attribute anyway. But, to be able to consistently enforce access control, we must keep track of all attribute versions *including* deleted ones.

We define the remove procedure which ensures the above requirements.

Initially, we depublish the attribute to delete a from the name system. Then, the attribute version is incremented and a shadow a' of the attribute is retained but never published. Finally, we re-authorize all RPs that have been issued tickets which include the key of the depublished attribute. The re-authorization ensures that the RPs have access to the same attributes they had access to before with the exception of the now deleted a.

```
Procedure remove
    input : User identity $\mathcal{U} = (\mathcal{I}_\mathcal{U}, \mathcal{N}_\mathcal{U})$
            User attribute $a = (key, value, version)$
    output: Updated identity $\mathcal{U}'$
1 $\mathcal{N}'_\mathcal{U} \leftarrow$ depublish($\mathcal{N}_\mathcal{U}$, $key$);
2 $a' \leftarrow (key, \perp, version + 1)$;
3 $\mathcal{U}' \leftarrow$ add($\mathcal{U}$, $a'$);
4 for each ticket $\mathcal{T} = (e_\mathcal{U}, e_\mathcal{RP}, \mathcal{A}_\mathcal{T}, n_\mathcal{T})$ issued by $\mathcal{U}$ do
5     $\mathcal{A}'_\mathcal{T} \leftarrow \{x \mid x \in \mathcal{A}_\mathcal{T} \setminus a\}$;
6     $(\mathcal{U}', \mathcal{T}') \leftarrow$ authorize($\mathcal{U}'$, $e_\mathcal{RP}$, $\mathcal{A}'_\mathcal{T}$);
7 end
8 return $\mathcal{U}'$;
```

4.3.9. Identity escrow and key management

In self-sovereign identity systems, the user is responsible for the management of private keys as well as identity attribute data. This includes critical processes usually taken for granted. Most notably, a reliable identity attribute storage which can be synchronized across devices.

Further, the fact that identities are basically public/private key pairs, a whole string of necessary key management aspects fall onto the user to take care of. Synchronization of keys across devices, recovery and revocation are the three most important processes required in our design.

On the surface, managing cryptographic keys in trusted execution environments of devices is an option. However, this severely limits crypto agility and also strongly binds an identity to a single physical device, a property that is likely counter-intuitive to a user. Consequently, a simple backup and recovery scheme for identity keys is a better solution.

In Section 4.4.3, we discuss how identity escrow as well as identity attribute synchronization can be realized as part of our implementation.

4.4. Reference implementation

We implemented our re:claimID prototype as part of the GNUnet peer-to-peer framework[2]. The client software [141] and core backends[3] are available online as Free Software. This licensing model plays a crucial role in order to achieve our goal of an open, free identity service which is available to the general public. It is important to mitigate the threat of commercial re-licensing and ensuring that the source code is available. While

[2] `https://gnunet.org`, accessed 19/12/2018

[3] `https://git.gnunet.org/gnunet.git/tree/src/reclaim?id=61625d4834bc7a599446486c9d16f2451527f989`, accessed 04/08/2020

our high-level design could be implemented in the form of proprietary software, the resulting product would inevitable not withstand our threat model.

In this section, we initially discuss the architecture and implementation details of our re:claimID prototype. We present our choice of name system and ABE scheme in order to elaborate on how they affect the security properties of the prototype. The component architecture of our implementation is illustrated in Figure 4.5.

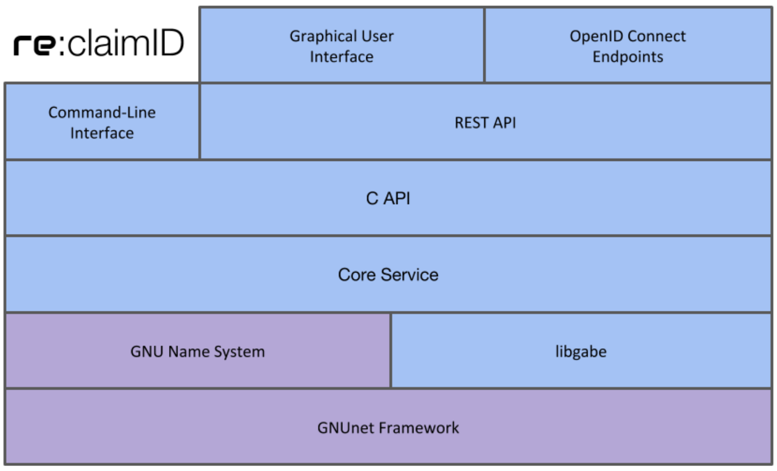


Figure 4.5.: The architecture of our re:claimID prototype. The purple base components are provided by the GNUnet framework.

The architecture consists of various application programming interfaces (APIs) which can be used to integrate our prototype into applications and systems. The re:claimID API exposes C and REST APIs which are used by other components. Details on the C API can be found in the GNUnet source code [120]. On top of the C API, we built a command-line interface (CLI) for use on the console and headless deployments, a REST API, an OpenID Connect 1.0 component and the graphical user interface (GUI) which is also used to guide the user through OIDC protocol flows.

In addition to details on the usage of the above interfaces, we present out-of-band authorization protocols. This includes implementation details on the OIDC layer which facilitates standards-compliant integration for RPs in Web use cases and an Android application for mobile use cases.

Further, we conduct both performance and usability evaluations on our re:claimID prototype. In our performance evaluation, we investigate the resolution times of re:claimID attributes in our GNS-based directory service while taking expected usage patterns into account. In this context, we also measured the performance impact of the used ABE scheme.

Finally, we carry out usability tests for each of the two use cases and the respective authorization protocols.

4.4.1. Overview

We build the decentralized directory service of our prototype on the GNS, a distributed hash table (DHT)-based name system which is part of GNUnet. In the following, we detail the inner workings of this name system and our ABE access control layer. Additionally, we discuss how both choices impact the security properties of our implementation.

GNS directory service

Storage of resource records as well as request and response routing in GNS is realized through the use of a DHT. The DHT used in GNS is called R^5N [43] and is designed to perform particularly well in restricted-route environments with malicious participants. Through the use of a DHT, GNS circumvents the need of centralized storage and infrastructure providers which are required in other name systems such as DNS.

According to Zooko's triangle theory for name systems [132], GNS name-value pairs are "self-authoritative": Names are linked together similar to how names are delegated in the Simple Distributed Security Infrastructure (SDSI) [108] which we have examined as part of our related work in Section 3.3.1. Like SDSI, GNS implicitly defines a trust model on top of a public-key infrastructure through the use of attribute delegation. Trust in GNS must be established out-of-band through a key exchange. Name-record pairs that are not trusted locally are resolved by traversing a trust path from the user to the name using SDSI-like delegation chains. Consequently, names in GNS are not globally valid. Each name is relative to the user that queries it. This is a major deviation from traditional name systems which provide globally unique names such as DNS and is essentially what makes GNS a "self-authoritative" naming scheme.

Figure 4.6 illustrates an example of GNS namespaces. Alice refers to Bob's namespace as "bob". To Alice, this is a top-level domain. She can resolve the IP of Bob's server under the name "www.bob". At the same time, Alice refers to Carol as "niece". She is able to resolve Carol's server under the name "website.nice". Through Carol, Alice is also able to resolve Bob's server using the name "www.bobby.nice" which yields the same result as "www.bob" but follows a different delegation path. This example illustrates how the names which point to the IP address of Bob's server are not the same for Alice and Carol and the names in general are not unique. This is an important differentiator between GNS and most other name systems which have a strictly hierarchical namespace organization.

Due to the lack of a strict hierarchy, GNS offers an open registration mechanism. Creation of a namespace is a simple matter of defining a public-private key pair (P, x). A namespace is identified by the public key P and records are signed using the private key x.

In GNS, requests for a name are routed through the network over a number of peers according to the routing protocol determined by the DHT. In that process, no peer that observes the query learns what name is requested in which namespace. Similarly, no peer – not even the peer that is chosen by the DHT replication mechanisms to persist the

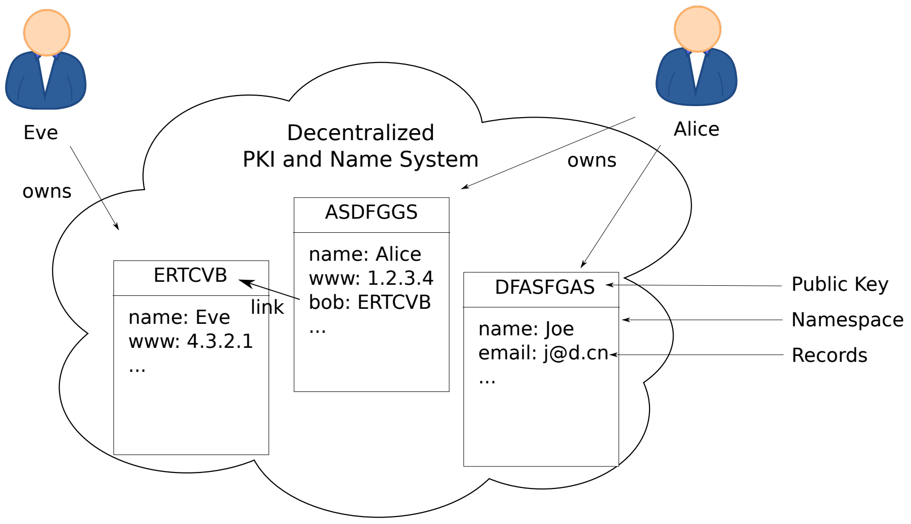


Figure 4.6.: Example scenario of namespace delegations in GNS.

respective records – learns the contents of the record data. The record contents can be read if the peer knows both the name and the namespace that is queried. This property is referred to as *query privacy* in GNS. It is realized through the derivation of a symmetric encryption key from the name and namespace information:

A record $\mathcal{R}$ is encrypted and stored in the DHT as a block $\mathcal{B}_{P,l}$ under the DHT key $q_{P,l}$. Here, l refers to the name and P to an ECC public key that is bound to a namespace. A user that owns the namespace generates the block $\mathcal{B}_{P,l}$ and the DHT key $q_{P,l}$ as follows: Let G be the generator in the ECC curve used by GNS where n is the size of the ECC group, a prime, and $n \leftarrow |G|$. $x \in \mathbb{Z}_n$ is the private key corresponding to the public key $P \leftarrow xG$. Initially, the user derives a private key d from the private key x and a hash h. The label l of the record and the public key P are hashed into h using a one-way cryptographic hash function H. The DHT key $q_{P,l}$ is a hash of the public key dG. The user then encrypts the record data $\mathcal{R}$ with a symmetric key derived from P and l using a hash-based key derivation function $HKDF$. The resulting ciphertext $\mathcal{C}_\mathcal{R}$ is signed using the private key d and a cryptographic signing function. $\mathcal{B}_{P,l}$ is a tuple consisting of the encrypted record data $\mathcal{C}_\mathcal{R}$, a signature $S_\mathcal{R}$ and the derived public key dG. The user publishes the block $\mathcal{B}_{P,l}$ in the DHT under $q_{P,l}$.

Equation 4.12 details the creation of a block $\mathcal{B}_{P,l}$.

$$\begin{aligned}
h &\leftarrow H(l,P)\\
d &\leftarrow hx \bmod n\\
q_{P,l} &\leftarrow H(dG)\\
\mathcal{C}_\mathcal{R} &\leftarrow \mathbf{enc}(HKDF(P,l),\mathcal{R}_{P,l})\\
\mathcal{S}_\mathcal{R} &\leftarrow \mathbf{sign}(d,\mathcal{C}_\mathcal{R})\\
\mathcal{B}_{P,l} &\leftarrow (\mathcal{S}_\mathcal{R},\mathcal{C}_\mathcal{R},dG)
\end{aligned} \tag{4.12}$$

In order to retrieve $\mathcal{R}_{p,l}$, the user must first calculate $q_{P,l}$. Unlike the namespace owner, he cannot calculate $q_{P,l}$ using d, as it is a private key. However, $q_{P,l}$ can alternatively be calculated using P and l:

$$\begin{aligned} h &\leftarrow H(l,P) \\ q_{P,l} &\leftarrow H(dG) = H(hxG) = H(hP) \end{aligned} \tag{4.13}$$

The user can decrypt the record data by deriving the symmetric decryption key $HKDF(P,l)$. Any peer unaware of P and l that encounters the block $B_{P,l}$ is able to verify the signature using $dG \in B_{P,l}$. Respectively, this enables peers to discard invalid or corrupt data. Had the signature been directly generated using the private key x, peers could easily identify the originating namespace as it is bound to the corresponding public key P. In GNS, the derived key dG does not enable peers to trivially identify the namespace corresponding to the record data. At the same time, the possibility of the above mentioned integrity checks is retained.

Similarly, an attacker in the network has a hard time extracting the label l from a query $q_{P,l}$ without also knowing P. If the attacker observes a query he only has two options: Either choose a fixed namespace P to attack and try to guess l or choose a fixed name l and guess the namespace P. In the first case, confirmation attacks against guessable names such as "www" or "email" are possible. Hence, the difficulty to guess l depends on the entropy and length of the chosen name. Another way to look at this property is that given a shared secret label l, a passive attacker is unable to decrypt or track record queries or record data. In the second case an attack is unfeasible as it would mean to brute-force the public key. Not the the term "public key" here is misleading. GNS does not expose private or public keys in the observable traffic of the protocol. Hence, if a zone public key is not known a priori or discovered through delegation, it remains confidential.

In combination the above properties are an integral feature of GNS which prevents leaking namespace contents. Further, it ensure query privacy and the confidentiality of the records persisted in the DHT.

ABE scheme

We realize the ABE-based cryptographic access control layer using the ABE library *libgabe* [119]. It is a fork of the original implementation by Bethencourt et al. [13]. We chose this implementation because of its general availability. Further, the use of *libgabe* allows us to establish an upper bound regarding our performance evaluation results.

Regarding our choice of ABE scheme, one might argue that our re:claimID prototype would benefit from recent advancements such as FAME [4]. FAME is superior to the original ABE scheme BSW by Bethencourt et al. [12] in terms of performance. However, FAME's advantage only fully takes effect for large, complex policies. This is because FAME always requires a fixed number of pairing operations. With other approaches, in particular the BSW CP-ABE scheme, the number of pairings increases significantly with the complexity of the policy.

Additionally, there is a difference with respect to key and ciphertext size: The FAME scheme requires three group elements per attribute. BSW only uses two group elements per attribute. This results in larger key and ciphertext sizes in the FAME scheme. On the other hand, FAME uses type-3 pairings, which are more efficient than the type-1 pairings used in the original [48]. The authors of FAME present an evaluation in [4] which shows that CP-ABE FAME outperforms BSW. In all three relevant aspects – key generation, encryption and decryption – FAME is faster. Most notably, decryption times are almost constant and independent of the attribute count due to the fact that a constant number of pairings is required for this operation in FAME.

In conclusion, we considered the original ABE scheme to be quite sufficient due to its maturity and simplicity. If the need for more complex policies including significantly higher number of attributes arises in re:claimID, schemes such as FAME that exhibit better performance in such cases can be integrated on demand. This is even possible without breaking backwards compatibility by introducing a new record type.

Security Properties

With re:claimID, we aim to satisfy the security goals of authenticity, integrity, privacy and confidentiality of identity data even in the face of our threat and attacker model. In the following, we explain how the goals are met within our prototype implementation.

Availability: We aim for high availability of the re:claimID service under the defined threat model for both the user and any RP. re:claimID inherits the security properties of GNS. In particular, the underlying DHT provides high resilience and censorship-resistance [43] against attackers which include our adversary. Records are replicated and stored redundantly in the DHT. No third party can prevent users from creating identities or managing attributes as long as cryptographic primitives such as record signatures cannot be broken.

Of course, availability can only be guaranteed by our implementation within the capabilities and resilience of the DHT. Especially if the peer-to-peer network only consists of a small number of nodes, a high churn rate can diminish data availability. Consequently, a stable user base is an important factor in order to ensure GNS record availability.

Integrity: Integrity of user attributes in our prototype is ensured using the data origin authentication property of GNS. All records mapped to the same name are aggregated and signed using the derived private key as detailed above. Blocks published in the DHT include the corresponding derived public key and signature. Further, the routing protocol of the DHT features a built-in signature verification in order to avoid replication and propagation of corrupt data. Each peer that encounters a block forwards and caches[4] it only if the provided signature can be verified using the provided public key. While

[4]Or persists if applicable. Persisting peers are selected through the replication model.

this allows attackers to store and persist any records he signs himself, it prevents an attacker from storing a block under a query key that is derived from the public key of another namespace not under his control. The attacker would need access to the derived private key of the respective namespace owner or else the provided public key in the block does not match the key used to derive the query key.

Privacy and Confidentiality: In our prototype, we achieve confidentiality of attribute data primarily through the use of ABE encryption which is part of the re:claimID design. Attribute data is exclusively stored in encrypted form.

However, our prototype is implemented on top of the GNS. As elaborated above, records in GNS are encrypted using a key derived from the query label and namespace public key. Further, GNS protects attribute enumeration from a passive and active attacker through its use of query privacy and the resulting mitigation of zone enumeration.

Consequently, in our prototype attribute records are effectively encrypted twice. The second layer of encryption which is provided by GNS is largely ineffective: As user attribute names presumably are easy to guess, e.g. "email", deriving the respective symmetric key is trivial. In Section 4.5, we provide a modification to the re:claimID design which leverages this property of GNS and allow us to replace to ABE access control layer with this built-in encryption. The resulting design, however, is no longer applicable to other name systems as to the best of our knowledge, query and response privacy is only found in GNS.

4.4.2. Attribute provisioning and sharing

In the following, we present how the CLI is used to perform basic re:claimID operations in our prototype.

Further, we present two authorization processes we have implemented for our prototype: The first is the OIDC protocol for web use cases. The second is an app based approach that is particularly suitable in Internet-of-Things (IoT) use cases.

In order for the user to manage identities, identity attributes and authorizations, we implemented an API, a CLI and a GUI. The GUI is also used in order to facilitate OIDC protocol flows. Consequently, we present it as part of our sections regarding attribute sharing.

Managing identities and attributes

In order to add an identity to re:claimID, a CLI which is part of GNUnet is used:

```
gnunet-identity --create=Alice
```

This will instantiate a new namespace in the GNS and generate the identity key set for *Alice* as defined in the re:claimID procedure **register**.

After creating an namespace, the user can add attributes to this identity using the `gnunet-reclaim` CLI:

```
gnunet-reclaim --ego=Alice --add email --value=alice@example.com\
                --type=STRING
```

In this example, Alice adds a new attribute "email", effectively executing the re:claimID procedure **add**. The attribute type is set to "STRING" which indicates that the attribute value is plain text. As elaborated in the re:claimID design, an attribute value may contain any kind of structured data including third party attestations. In this example, the value itself is simply a string representing Alice's email address. This CLI supports the other re:claimID procedures required to perform basic operations on attributes such as enumeration, deletion and update.

Managing authorizations

To authorize a RP to access a set of attributes, the user invokes the `gnunet-reclaim` CLI to generate a ticket. For ticket generation, the user must know the RP public key `rpkey` and the requested set of attributes. We assume that Alice wants to authorize a RP to access her email address and full name. She executes the following command which triggers the re:claimID **authorize** procedure of our prototype:

```
gnunet-reclaim --issue=email,full_name --ego=Alice \
               --rp=$rpkey
```

This will yield a re:claimID ticket in a string format. Alice manually transfers the resulting ticket $\mathcal{T}$ to the RP which can in turn process it to retrieve the attribute values. Let us assume that the identity behind `$rpkey` is Bob. He executes the following command which triggers the re:claimID **retrieve** procedure:

```
gnunet-reclaim --ego=Bob --consume=T
```

Bob's re:claimID client will initiate the re:claimID **retrieve** procedure including resolution and decryption of Alice's attributes.

If Alice decides to revoke Bob's access to the respective attributes, she simply executes:

```
gnunet-reclaim --ego=Alice --revoke=T
```

As mentioned in the design in Section 4.3.5, for a user to authorize a RP a ticket $\mathcal{T}$ must be transferred out-of-band. This authorization process must include the obtaining of user consent. We address available authorization processes in our prototype next.

Authorization over the Web

Every authorization process must include obtaining of user consent. A prime example of a protocol that facilitates such a process is OpenID Connect 1.0. Especially smaller websites and services commonly rely on this standardized protocol to outsource identity management. This includes user authentication and identity attribute management.

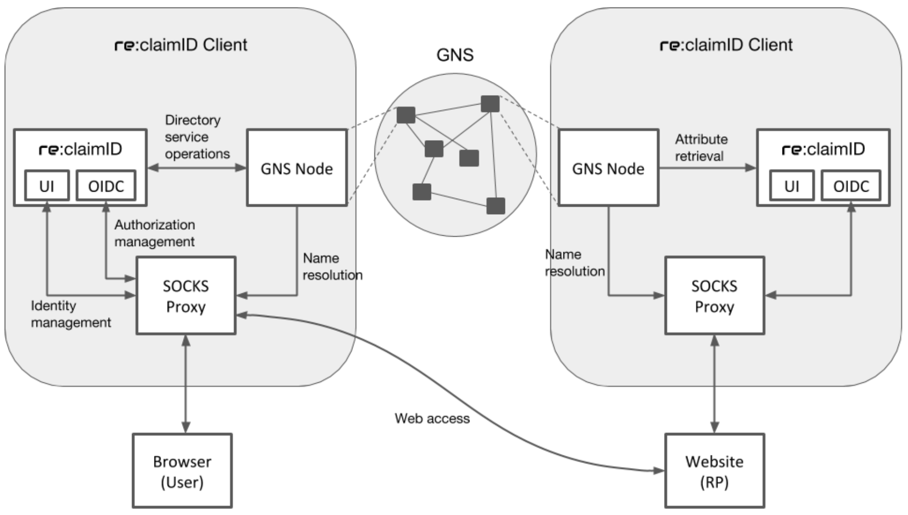


Figure 4.7.: An overview over the re:claimID client as used by end users and RPs.

Our prototype includes a drop-in replacement for "social login" systems built using OIDC using the feature set provided by our re:claimID implementation. Our aim is to offer a standards compliant layer on top of re:claimID. The result of this effort is that our prototype is compliant with the OIDC standard specification which facilitates integration into websites and services following in a similar fashion as existing IdPs.

In the following, we elaborate on how artifacts and protocol flows in OIDC can be used to abstract procedures and object of the re:claimID design. Figure 4.7 illustrates the design of our client implementations and the respective information flows.

Endpoints OIDC is an OAuth 2.0 profile which means it inherits the respective endpoints. OAuth 2.0 defines two REST endpoints: the *authorization endpoint* and the *token endpoint* which are normally served in OIDC by the IdP service. The OIDC specification additionally defines a third endpoint called *userinfo*. The user is primarily interacting with authorization endpoint. On the other hand, the RP is primarily interacting with the token and userinfo endpoint. In our design, both the user and the RP run their own instance of our prototype implementation. Each instance serves the above endpoints. Both user and RP only communicate with their local endpoints, but the behavior is the same as if it were a single service instance. Because a user may act as an RP and vice versa, all endpoints are exposed on every local instance, but are used according to the role requirements as defined in the specifications. This approach of distinguishing between user and RP facing endpoints that represent a single service realized on top of the decentralized re:claimID service is illustrated in Figure 4.11.

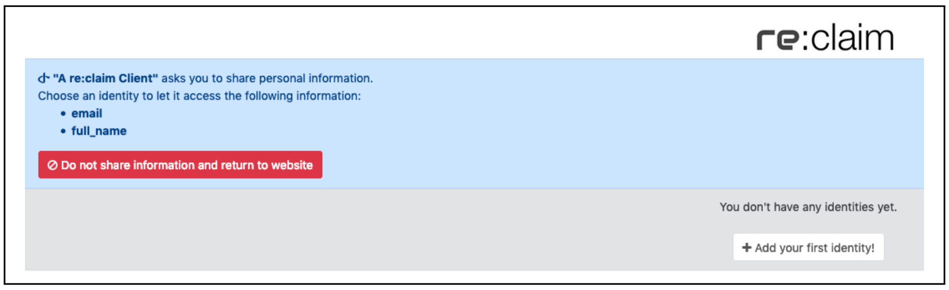
Grant types OIDC and OAuth 2.0 support a range of grant types. Grant types define ways in which RPs can prove that they obtained consent from the user. In our prototype, we exclusively support the "authorization code" grant type which is the recommended default according to the specification. While theoretically other grant types are feasible, this grant type is coincidentally the most suitable for integrating the re:claimID authorization procedure. In particular, this method facilitates the out-of-band exchange of the authorization ticket $\mathcal{T}$. An example of the authorization code flow is also illustrated as a swim lane chart in Figure 4.11.

Client registration The client in OIDC is the software component of the RP. A client is registered at the OIDC IdP. The core OIDC specification does not cover a registration protocol. In our implementation, registering a client entails registering a namespace for the RP in GNS and creating a respective public-private key pair. The public key serves as the client identifier `client_id`. Upon registration, the client is provided by the IdP with local credentials for authentication at the RP facing token endpoint. OIDC client authentication, which is also mandated by the specification, is only relevant for communication with the local re:claimID instance. The standards mandate the registration of one or more `redirect_uri` parameters which the client plans on using during authorization flows. The user must verify that a given client has registered this URI. This verification is an important security aspect also documented specifically in the OAuth 2.0 specification [61, Section 10.6]. Registration of this parameter is done in our design by adding a special resource record of type "RECLAIM_OIDC_REDIRECT" to the RP namespace under the empty label. It allows the user to resolve the parameter and verify that a given authorization request from a client matches the registered redirects. When the user's re:claimID OIDC instance receives an authorization request by the RP, it verifies that the provided redirect URI in the request matches the one published in GNS. Due to the GNS data origin authenticity guarantees, only the RP is able to publish the record under the respective namespace identified by the `client_id`. In addition to the redirect parameter, the client may also add a record of type "RECLAIM_OIDC_CLIENT". This record contains a human readable description of the client that can be displayed to the user during the authorization flow in order to facilitate consent retrieval. Table 4.3 illustrates the namespace of an RP including the respective OIDC specific entries.

Name	Type	Value
@	`RECLAIM_OIDC_CLIENT`	"Example Website"
@	`RECLAIM_OIDC_REDIRECT`	https://www.example.com/oidc_cb
www	`A`	1.2.3.4
www	`BOX`	TLSA data

Table 4.3.: An example namespace for a RP in re:claimID.

Figure 4.11 illustrates how a user – Alice – manages her attributes using our re:claimID prototype through a web frontend running on her local machine (1). The user interface as shown in Figure 4.8 guides the user through this process. Initially, Alice does not have any identities configured so the first step is to create a new identity. In our implementation, the user has the option to create any number of identities and edit them to their needs.


re:claim

"A re:claim Client" asks you to share personal information.
Choose an identity to let it access the following information:
- email
- full_name

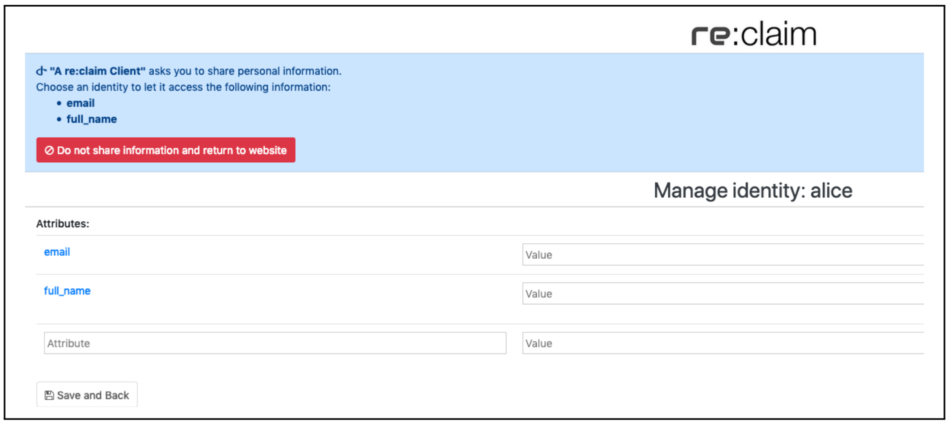
Do not share information and return to website

You don't have any identities yet.

Add your first identity!

Figure 4.8.: Initially, the user is asked to add their first identity.

Subsequently, Alice is given the option to provision the identity with attributes. In Figure 4.9, we can see that if a set of attributes $\mathcal{A}$ is requested by a website, the attribute keys are preallocated in the interface. Any attribute added by the user is encrypted and published in the directory (1) as defined in the re:claimID design. We assume that Alice wants to register at a service through its website (2). To initiate the OIDC authorization flow, the website contains a respective login button. When pressed, the website initiates an OIDC authorization code flow through the use of a browser redirect to the re:claimID OIDC endpoint (3a). In the redirect, the service requests access to the user attributes $\mathcal{A}$ which is according to the specification done using URI query parameters. She may at this point deny authorization and the process ends.


re:claim

"A re:claim Client" 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

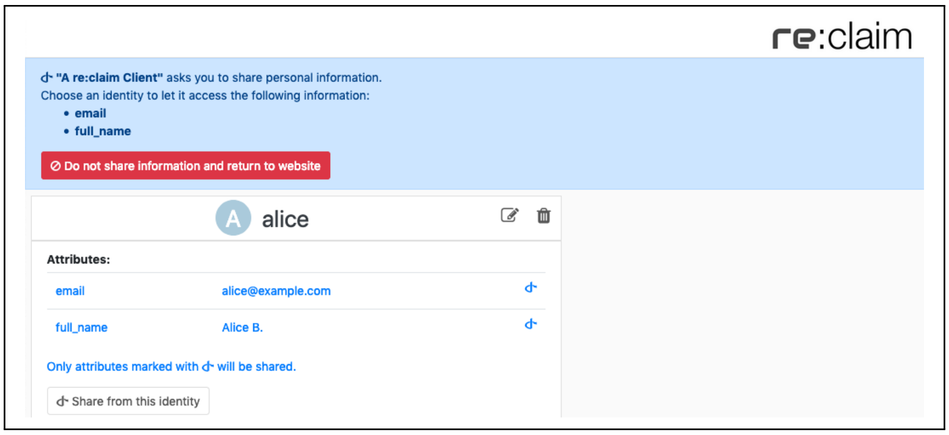
Attributes:

email	Value
full_name	Value
Attribute	Value

Save and Back

Figure 4.9.: The user is asked to add the requested attributes after adding a new identity.

If Alice consents to the authorization request by selecting an identity from the user interface (3b), the authorization procedure as defined in Section 4.3.5 is triggered (4). The result of the re:claimID authorization yields a ticket $\mathcal{T}$. The specification specifies, that the browser must then be redirected back to the website (5a,5b) along with an "authorization code" in a URI parameter. We piggyback the ticket $\mathcal{T}$ in the OIDC authorization code.


re:claim

"A re:claim Client" 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

alice

Attributes:

email	alice@example.com
full_name	Alice B.

Only attributes marked with will be shared.

Figure 4.10.: The main re:claimID user interface listing all available user identities.

The service then exchanges the code at the OIDC token endpoint according to the specification (6). Our re:claimID OIDC layer interprets the code and extracts the ticket $\mathcal{T}$. The ticket is subsequently used to retrieve Alice's attributes $\mathcal{A}$ as defined in Section 4.3.6 (7). The token endpoint returns an access token to the service that can be used to retrieve the attributes. In OIDC, user attributes are retrieved from the *userinfo* endpoint. The endpoint must be accessed using the access token. The resulting response contains the user attributes $\mathcal{A}$ (8). Our implementation caches the ticket $\mathcal{T}$. Using the access token the RP is able to retrieve up to date user attributes at any time which is the expected behavior in OIDC.

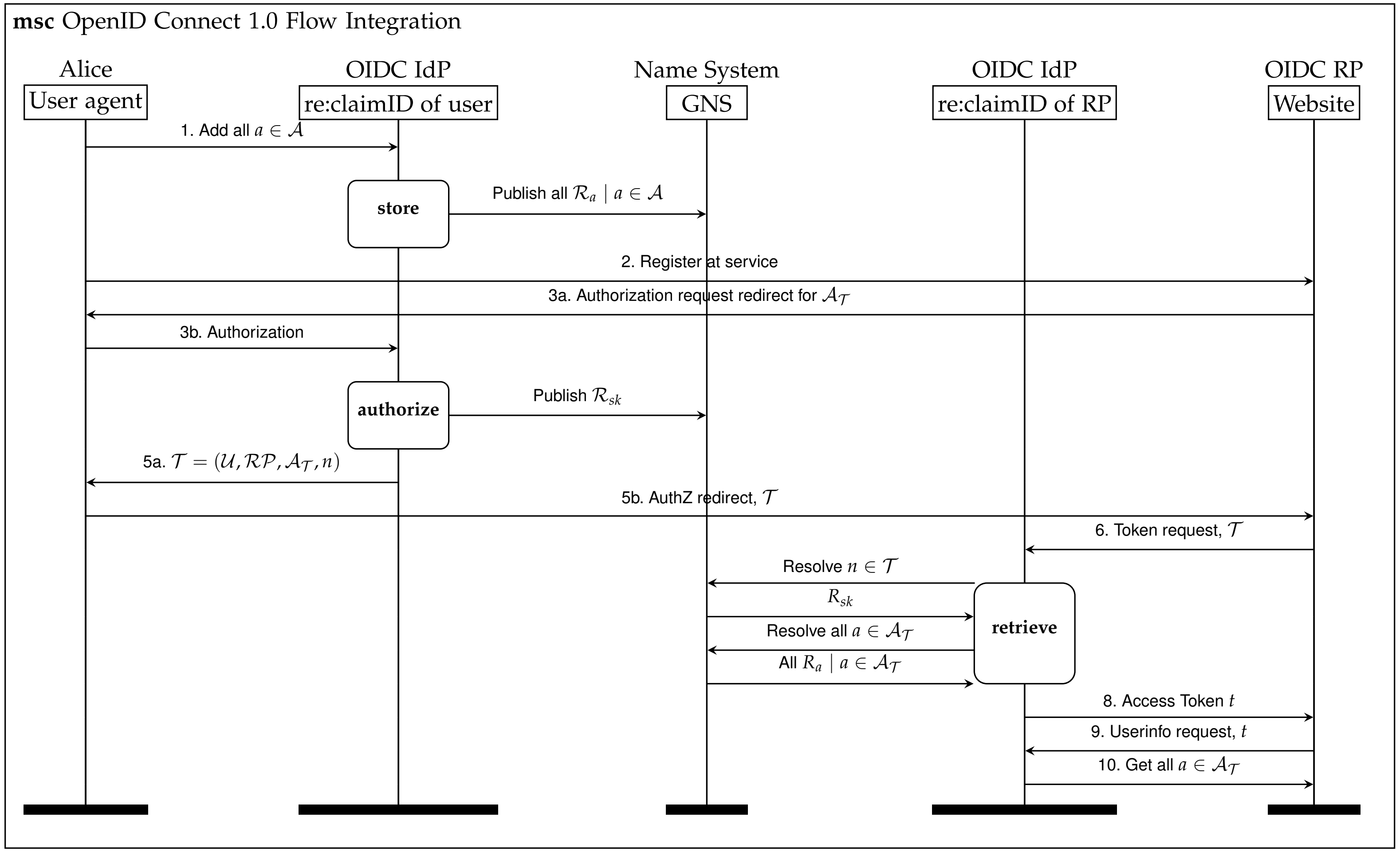

msc OpenID Connect 1.0 Flow Integration
Alice
User agent
OIDC IdP
re:claimID of user
Name System
GNS
OIDC IdP
re:claimID of RP
OIDC RP
Website
1. Add all $a \in \mathcal{A}$
store
Publish all $\mathcal{R}_a \mid a \in \mathcal{A}$
2. Register at service
3a. Authorization request redirect for $\mathcal{A}_\mathcal{T}$
3b. Authorization
authorize
Publish $\mathcal{R}_{sk}$
5a. $\mathcal{T} = (\mathcal{U}, \mathcal{RP}, \mathcal{A}_\mathcal{T}, n)$
5b. AuthZ redirect, $\mathcal{T}$
6. Token request, $\mathcal{T}$
Resolve $n \in \mathcal{T}$
R_{sk}
Resolve all $a \in \mathcal{A}_\mathcal{T}$
All $R_a \mid a \in \mathcal{A}_\mathcal{T}$
retrieve
8. Access Token t
9. Userinfo request, t
10. Get all $a \in \mathcal{A}_\mathcal{T}$

Figure 4.11.: An OpenID Connect authentication code flow in re:claimID.

Authorization using mobile applications

In the context of the Internet-of-Things (IoT) sharing authenticated data across different security domains is an important use case. We propose the use of re:claimID in order to enable domain-spanning sharing of device data. Traditional approaches to this challenge often follow the traditional pattern of adding trusted, centralized third parties such as device vendors or other intermediaries to enable the exchange of data. re:claimID allows us to propose a more elegant solution to this challenge. We use it to create an inter-domain infrastructure that allows users and device alike to establish trust relationships directly between entities and securely exchange information.

Let's consider the following example: A user acquires a device for his home such as a smart thermometer for heating. The vendor of this device might provide a value adding cloud service which allows the user to monitor the sensor output of the device and interconnect it with other devices. Traditionally, the link between device and vendor service is fundamental to the operation of the device. If the vendor goes under the device is more often than not no longer usable.

re:claimID allows us to mitigate those issues. While the vendor may provide an offering itself, if the device is able to publish and share its data through re:claimID, any third party service or even the user itself are able to access and process it. In order to do so, we have designed and implemented an authorization flow using a mobile application which authorizes an RP to access data published by a smart thing in re:claimID. Figure 4.12 illustrates the design and authorization flow. The protocol flow is illustrated in Figure 4.13.

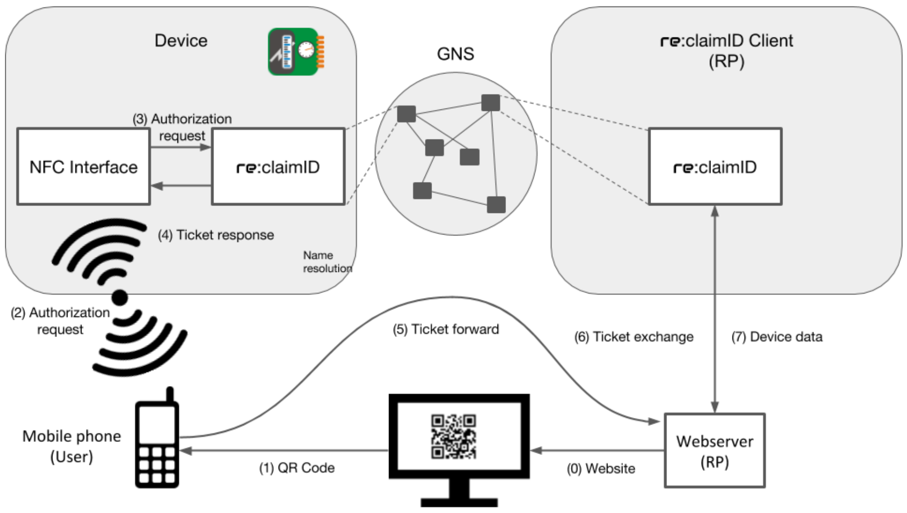


Figure 4.12.: Our re:claimID architecture in an IoT use case.

The smart device is running a re:claimID instance and publishes information, for example sensor data. A RP serves a website which displays a QR code. The user uses our re:claimID mobile application in order to scan the QR code (1). The QR code contains the information necessary to initiate a re:claimID authorization procedure at the device. The user implicitly consents to the authorization request by initiating an NFC exchange with the device in question (2). The device receives the authorization request, and starts the re:claimID **authorize** procedure (3). The resulting ticket $\mathcal{T}$ is transferred back to the mobile application (4). The mobile application in turn transfers the ticket to the RP (5). The RP is now able to retrieve the device data from re:claimID (6,7).

As a result, the user can authorize any re:claimID enabled RP to access the device data. This includes third party services or services created by the user.

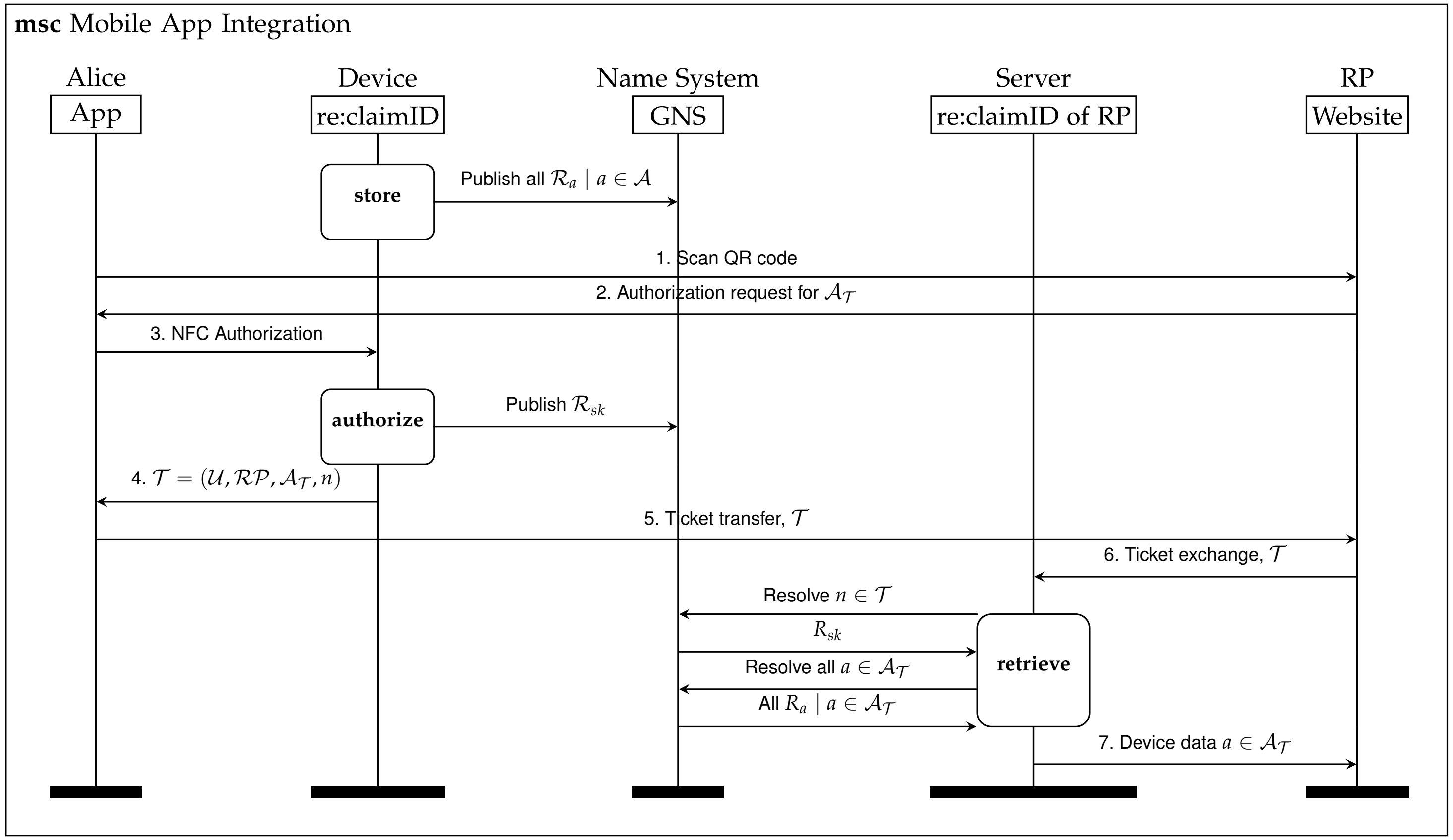

msc Mobile App Integration
Alice
App
Device
re:claimID
Name System
GNS
Server
re:claimID of RP
RP
Website
store
Publish all $\mathcal{R}_a \mid a \in \mathcal{A}$
1. Scan QR code
2. Authorization request for $\mathcal{A}_\mathcal{T}$
3. NFC Authorization
authorize
Publish $\mathcal{R}_{sk}$
4. $\mathcal{T} = (\mathcal{U}, \mathcal{RP}, \mathcal{A}_\mathcal{T}, n)$
5. Ticket transfer, $\mathcal{T}$
6. Ticket exchange, $\mathcal{T}$
Resolve $n \in \mathcal{T}$
R_{sk}
retrieve
Resolve all $a \in \mathcal{A}_\mathcal{T}$
All $R_a \mid a \in \mathcal{A}_\mathcal{T}$
7. Device data $a \in \mathcal{A}_\mathcal{T}$

Figure 4.13.: A QR Code/NFC proximity based authorization for re:claimID.

4.4.3. Identity escrow

GNS already provides us with a mechanism which allows us to revoke zone keys. However, it does not define a zone key backup or escrow system. In this section, we briefly discuss how we envision a user to manage his re:claimID identities with respect to cryptographic keys and attribute data.

Key revocation

Zone key revocation in GNS works by flooding a pre-computed revocation certificate through the network. The pre-computation of the certificate includes a signature using the original private zone key. Consequently, the revocation certificate must be generated and stored securely out of band before the private zone key is lost.

In order to prevent an adversary from maliciously flooding revocations through the network, the revocation certificate must include a proof of work. This proof of work is essentially a nonce *POW* which concatenated with the revocation certificate satisfies a specific difficulty constraint. In GNS, the hash over the certificate and the *POW* must exhibit a number D of leading zeros. This number D is specified by the revocation protocol and a valid proof can be verified by any node observing a flooded key revocation. Hence, malicious flooding of the network with revocation messages is prevented by having this proof of work requirement in place. Also, in order to effienctly handle network partitions, GNS implements an efficient set reconciliation protocol for distributed systems as proposed by Eppstein et al. [39].

We improved upon this revocation protocol of GNS zones as part of our implementation of identity key revocations in order to address two issues:

First, traditional proof of work schemes require to find a nonce such that a hash over the nonce is found which exhibits at least D leading zeros. Such calculations unfortunately have a undesirable statistical property when it comes to practical use of proofs with a high difficulty: High variance. High variance in proof of work calculations result in a situation where a proof on average can be calculated within a given time period on a specific hardware configuration. However, in unlucky cases this calculation may take significantly longer (or shorter) due to the high variance in the exponential distribution of results.

Second, the revocation message as implemented in GNS is not limited in terms of lifetime. A revocation message is valid *forever*. This has the effect that a proof calculation amortizes over time, making it basically just a minor hassle that can be stretched over a long period of time consequently making the selection of an appropriately hard proof target difficulty D impossible.

Our solution is to address those two issues by:

1. Reducing the statistical variance of the proof calculations by distributing the difficulty target across a number of proof calculations.

2. Incorporating timestamps and lifetime indicators in the revocation certificate.

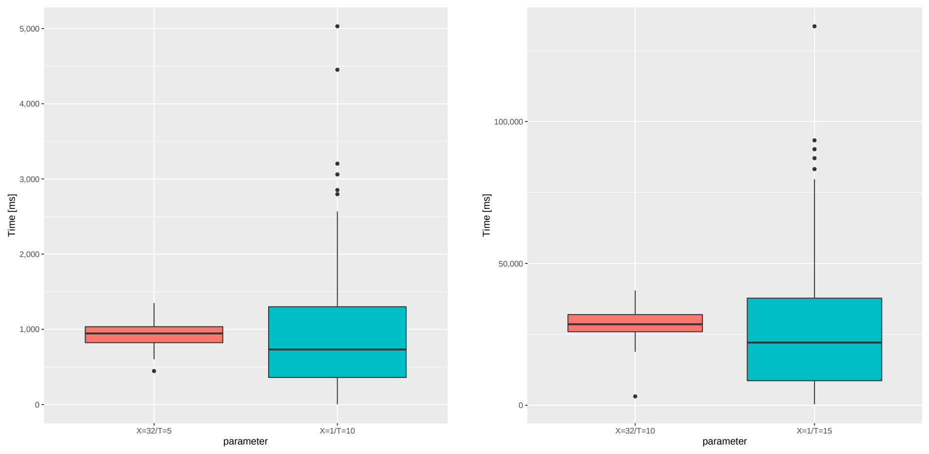


Figure 4.14.: Box plots of 100 proof calculations using different parameters.

We propose to reduce the variance in time it takes to calculate the proof of work by calculating a set of X statistically independent proofs where all proofs combined exhibit – on average – a difficulty of D.

$$\frac{\sum_{i=0}^{X-1}(D_i)}{X} \overset{!}{>} D \tag{4.14}$$

We experimentally confirmed that such an approach significantly reduces the variance of proof calculations. The box plots in Figure 4.14 illustrate our results. Our improved scheme with $X = 2^5$ independent proofs fares better than a naive proof of work implementation in various difficulty settings. The results show that in order to have proof calculations require the same amount of work, our scheme requires five less zeros in the average difficulty when compared to traditional calculations.

Regarding the lifetime of a revocation we define that a revocation certificate includes a timestamp, which denotes the date at which the certificate becomes valid. The certificate must not be considered valid before this date. The time to life of the revocation is determined by the average number of zeros which are exceeding the target difficulty D. For example, with our target difficulty 22 and the actual proof results in an average of 25, then this revocation is valid for $25 - 22 = 3$ *epochs*. Hence, the lifetime of a revocation certificate is implicitly defined through the difficulty of the provided proof and can be verified by each node as part of the proof validation.

In the current implementation, the target difficulty D is set to 22 leading zeros and a single epoch is defined as 365 days.

Escrow and data synchronization

Identities consist of both cryptographic keys as well as associated attributes. In order to facilitate the use of sets of identities between different devices, an escrow and synchronization mechanism is required. Our current design requires the user to manually synchronize key material as well as attribute data between devices.

However, in future work we plan to integrate modern key escrow and synchronization systems such as Taler's Anastasis [133] and the corresponding data synchronization service [134]. Both services are still under development and require further research for use in decentralized use cases. Specifically a suitable conflict-free replicated data type (CRDT) suitable for re:claimID identities is needed.

4.4.4. Performance evaluations

In order to verify the practicality of our prototype implementation and to identify possible scalability issues, we carry out performance testing. However, we limit our performance tests to the attribute retrieval aspect of the re:claimID design. This is due to the following reasons: First, authorization and deletion of attributes is simply a matter of cache expiration in GNS. Hence, an evaluation on those aspects is essentially only a trivial cache expiration verification. Second, we expect attribute retrieval to always be the most relevant process in performance measurements. An evaluation of an authorization includes ABE decryption and corresponding computationally intensive operations. Further, it also allows us to measure retrieval times of attributes across time and network sizes.

In the following, we test our implementation with regards to the following aspects related to the authorization process:

- Median time of key retrieval
- Median time of attribute retrieval
- Caching behavior on attribute retrieval
- Performance impact with respect to network size

Test setup

We conduct our evaluations on a virtual host with 32 vCPUs at 2,3 GHz and 32GB of RAM. In order to determine the time it takes to resolve keys and attributes, we bootstrap a GNUnet network $\mathcal{N}$. We define a *test run* to consist of multiple iterations of a single test. In each run, we repeat one test 10 times. We execute 1000 test runs to increase the reliability of our data set for a single test run. Further, we want to investigate the impact of the size of the network. Let n be the network size of $\mathcal{N}$. We run our experiment for $n = 50$, $n = 100$, $n = 150$ and $n = 200$, respectively. Before every test run, we bootstrap $\mathcal{N}$ again to ensure that any caches are purged.

A single test consists of the following steps:

1. We randomly choose a user node $A \in \mathcal{N}$ and a RP node $B \in \mathcal{N}$.
2. A and B create identities and exchange the respective public keys.

3. A adds an attribute a and authorizes B's access.
4. We simulate an out-of-band transfer of the ticket $\mathcal{T}$ from A to B
5. B uses the ticket to…
   - …retrieve the ABE user key $sk_{\mathcal{A}}$
   - …retrieve the attribute a and to decrypt the attribute value.

Each time we repeat the test, we randomly choose a different node $B \in \mathcal{N}$. The test is executed 10 times in every test run. However, A and a stay fixed after the initial test across a test run. As we do not tear down or re-bootstrap the network between tests, this allows us to investigate the impact of GNS caching on the authorization process. We measure the time it takes each node B to resolve the user key and the attribute. Hence, performance should mainly depend on the topological distance between A and B in the network. As the routing in the R^5N DHT used in GNS in part uses randomized routing strategies, this introduces some variability that is difficult to control for. Our experimental setup consists of a clique topology.

Results

In Figure 4.15, we show an overlay of the median attribute retrieval times with respect to network sizes. The data shows that attribute retrieval performance decreases with the size of the network. Retrieval times within a test run appear to converge. This effect also decreases with increasing network size.

In Figure 4.16, the data shows that the time it takes to resolve a user key varies heavily with a median of around 200 milliseconds. The variance is high throughout all 10 successive tests within a single test run and also across runs. If we take into account that we cannot expect any cache hits due to the individuality of this query, the observed data is plausible. The fact that the initial resolution exhibits a particularly high variance and median retrieval time supports this as well.

Regarding attribute retrieval, we can see in Figure 4.17 that the times also initially exhibit a high variance. In contrary to the ABE keys, the retrieval times of attributes within a run decrease. In fact, the retrieval times converge across network sizes down to less than 100 milliseconds. Similarly, variance decreases within a single test run and across network sizes. The data set shows the effects of caching in the name system and how it influences our implementation. We can assume that after the first initial RPs are authorized by a user to access attributes, resolution times improve greatly. That is, if the RPs retrieve attributes.

Discussion

The key and attribute retrieval times within a single test run initially exhibit a high variance. Attribute retrievals that follow the initial test within a single test run are faster and show increasingly less variance due to GNS caches kicking in. As expected,

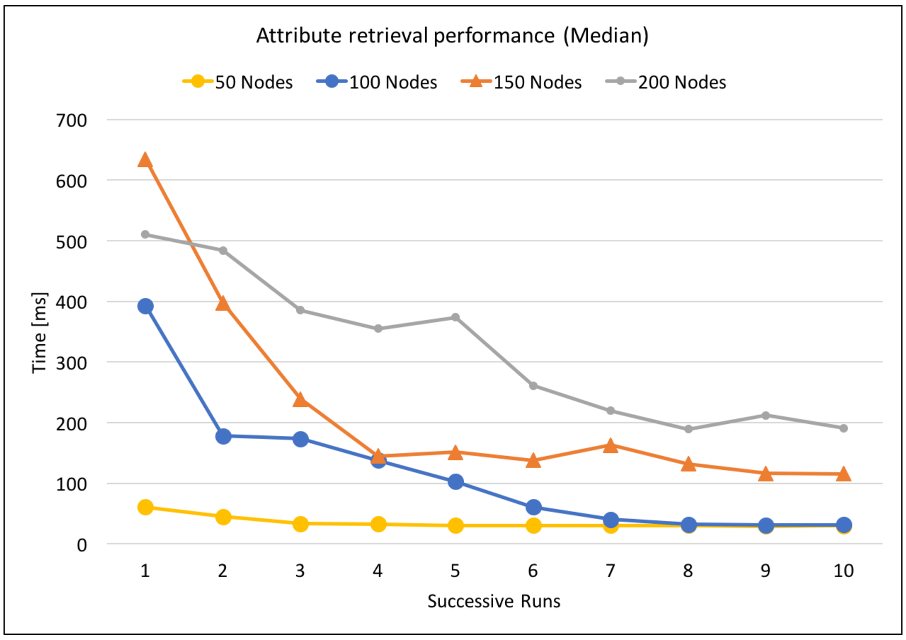


Figure 4.15.: Median attribute retrieval times across all test runs for network sizes of 50, 100, 150, and 200 nodes. [114]

we did not observe this behavior with respect to key retrieval. We further observed that increasing network size has a negative effect on retrieval times and the respective variance.

In summary, our evaluation results show that the implementation converges into a reasonably well performing system when used. Resolution times of around 100 milliseconds are acceptable in cases where user data is retrieved by requesting parties without user interaction after an initial authorization flow. We assume that an increase in the number of retrieved attributes does not significantly impact performance as attributes lookups can be performed in parallel.

We note that key resolution within the attribute retrieval process does not benefit from caching in our implementation using GNS. This impact is particularly noticeable in an interactive authorization protocol such as OIDC where the RP is expected to immediately require the attributes. In this case, we recommend an initial out-of-band transfer of the key as part of the authorization protocol. For example, by including the key in the code of an OIDC authorization redirect in addition to the ticket $\mathcal{T}$. This is also true for other implementations on top of name systems that make heavy use of caching strategies such as DNS or other delays in record publishing such as consensus algorithms in Namecoin.

Another case would be key rollovers for example those triggered by user initiated access revocations. We cannot predict the points in time of the revocation and the next occasion an RP requires the attributes to align. Suppose the attributes are required for processing by the RP but the key rollover happened just recently. Consequently, the

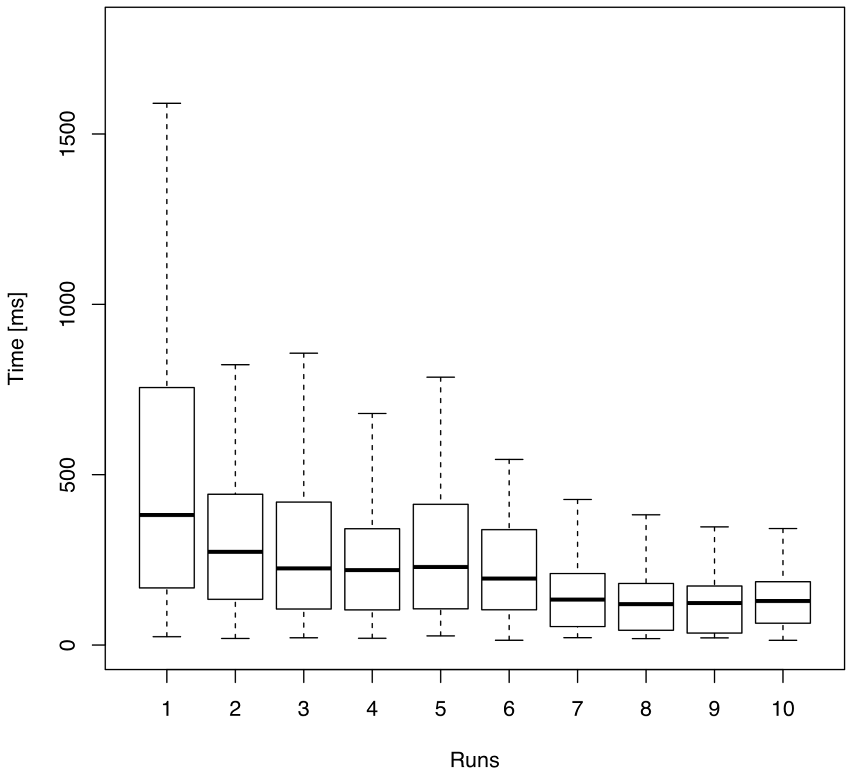


Figure 4.16.: Key retrieval performance of user keys for a network size of 100 nodes. [114]

attribute can no longer be decrypted. To minimize latency in such a scenario, we suggest that RPs periodically update the ABE key by retrieving it from the name system instead of only doing so when the next decryption fails.

4.4.5. Usability studies

In the following, we present the experimental designs and results of two usability studies of our prototype implementation. The first use case is a generic Web-based user data authorization scenario using our OIDC implementation. The second use case revolves around IoT device attribute sharing with a Web service and our NFC-based authorization protocol. The users are surveyed through a standardized System Usability Scale (SUS) and further questionnaires in order to evaluate key usability indicators. The wide-spread nature of the SUS score enables us to interpret its results in a way that facilitates a generalized judgment on the usability of our implementations.

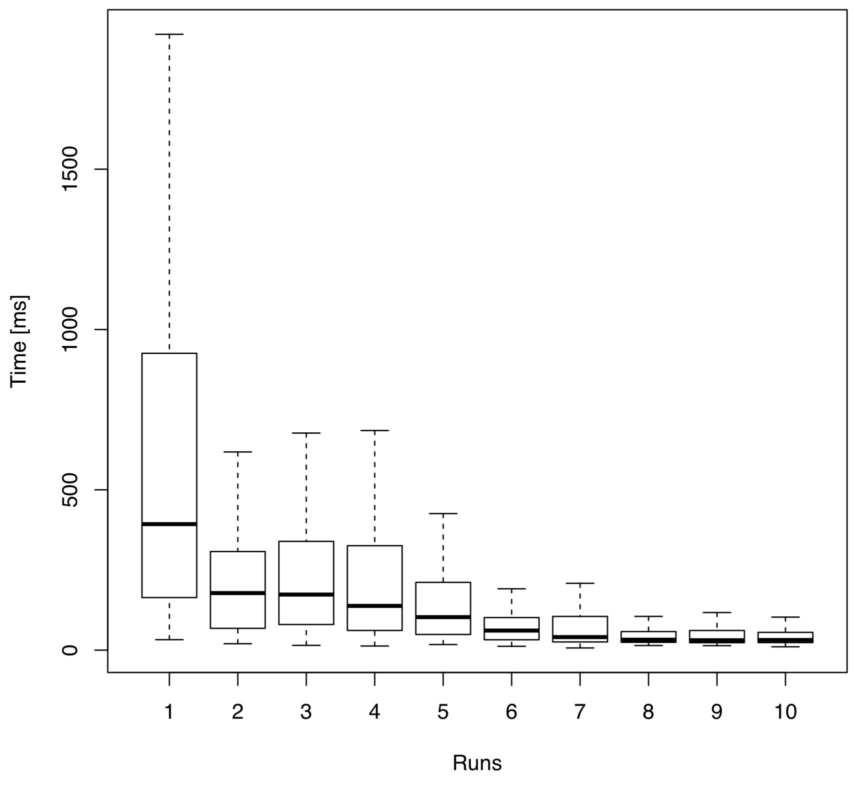


Figure 4.17.: Attribute resolution performance for a network size of 100 nodes. [114]

Use case 1: Web

For our web based use case, we evaluate our OIDC implementation on top of re:claimID. The specification of OIDC only covers the technical details of the protocol flow. User interface aspects are intentionally out of scope in the specification. Hence, user experience across different OIDC implementations may vary greatly. Our implementation features a website for identity attribute and authorization management which is the primary target of evaluation.

To evaluate the user experience, we create a RP website that requests access to user attributes by initiating an OIDC flow. The test is organized as follows: We prepare a workstation for the user that is running the RP web page as well as our re:claimID implementation. The user is provided with a written guidance in order to ensure that the instructions given to each user do not differ. The contents of the instruction sheets can be found in Appendix B.3. According to the instructions, the user is pointed to the RP web page in the browser. Then, we instruct the user to login using the re:claimID option.

The re:claimID button initiates the OIDC flow and redirects the user to our re:claimID OIDC authorization website. Here, the user is notified regarding the attributes which are requested by the website. Initially, there are no identities configured. The user is prompted by the user interface to create a new identity and then to add the requested attributes. In our evaluation, the request is for a full name and email address.

Once the user finished authorizing the RP, the OIDC "code" which represents the user consent and contains the ticket is transferred to the website. The website uses the code to retrieve the user attribute and display the success of the exchange with their local re:claimID OIDC instance. The user is eventually presented by a welcome web page displaying the personal data that the user has shared with the website.

We aggregated the collected SUS scores of 38 participants that completed the task into a box plot in Figure 4.18. The collected SUS scores can be found in Appendix B.3. The data shows, that the median SUS score is above 90 points. This indicates that the usability of our re:claimID OIDC user interface is above average when compared to other systems [92]. It is is well above 80,3%, which groups it in the top 10% of all scores collected by [92].

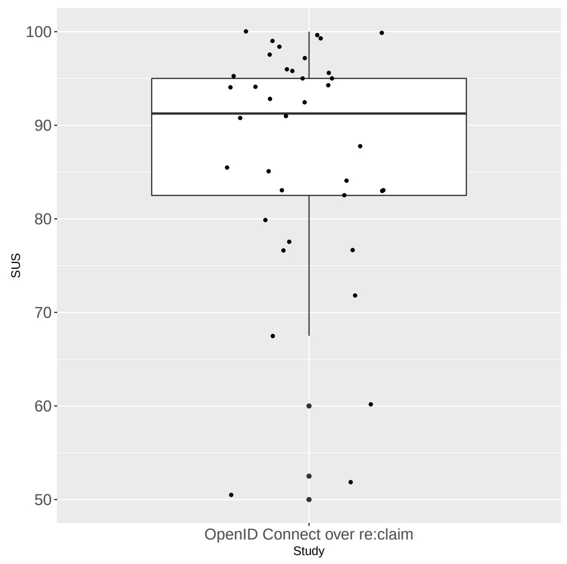


Figure 4.18.: SUS scores for the re:claimID web use case. [57]

Out of 42 participants that entered the survey form, we had 27 men, 6 women and 7 who refused to answer. The self-reported age groups of the participants were 7 in 18–24, 19 in 25–34, 7 in 35–44, 6 in 45–54, 1 in 55–64 and none older. In terms of education, 1 reported no degree, 4 reported a high school graduate degree, 3 professional training, 9 Bachelor degrees, 17 Master degrees, and 6 Doctorates.

Use case 2: IoT

In addition to our web use case, we demonstrate re:claimID as a suitable authentication and authorization system for the IoT. To do so, we conduct a usability experiment that involves a user acting as a technician, a device and a web service. The device is a hardware component which is equipped with a variety of sensors. The web service is capable of consuming sensor data and process it. In the real world, this service could be used in predictive maintenance or other data analytics use cases. The data origin in this use case are the sensors attached to the device. The user is acting as the owner of the device and is in charge of authorizing RPs to access the data.

In our design, the device is collecting sensor data and publishes it in re:claimID. Hence, the device is setup with a re:claimID identity. The device is further equipped with a near field communication (NFC) interface. We use the NFC interface in our authorization flow as discussed in section 4.4.2. The technician uses a mobile phone which is running the re:claimID mobile application. For our survey, we created a prototype device as depicted in figure 4.19. The device is based on a Raspberry Pi 3B. We attached an NFC shield used for the NFC authorization flow and a BME280 sensor. The sensor is able to measure temperature and atmospheric pressure.

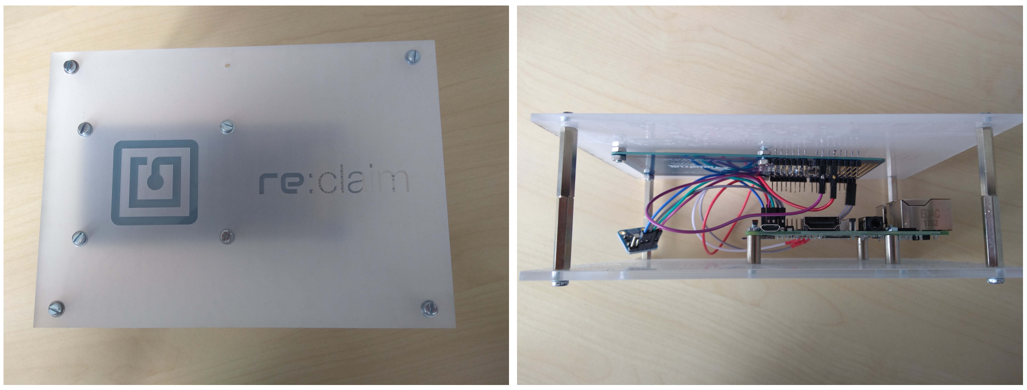


Figure 4.19.: The prototype re:claimID device used in the usability study. [57]

In our study, the technician is tasked to authorize the web service to access the sensor data. To do so, the technician first accesses the website of the web service. The website displays a QR code which contains the authorization request including requested attributes. The technician is instructed to scan the QR code using the re:claimID app. Using the re:claimID app, the technician must scan the QR code:

Upon scanning the QR code, the user is presented with information regarding which attributes the web service requests. In our specific setup, the requested attributes are temperature, pressure as well as altitude. The user is instructed to consent to the request by bringing the phone in close proximity to the device.

The mobile application relays the authorization request of the website to the device and triggers a re:claimID authorization procedure. The device prepares the authorization ticket and includes it in the response to the mobile application. The mobile application

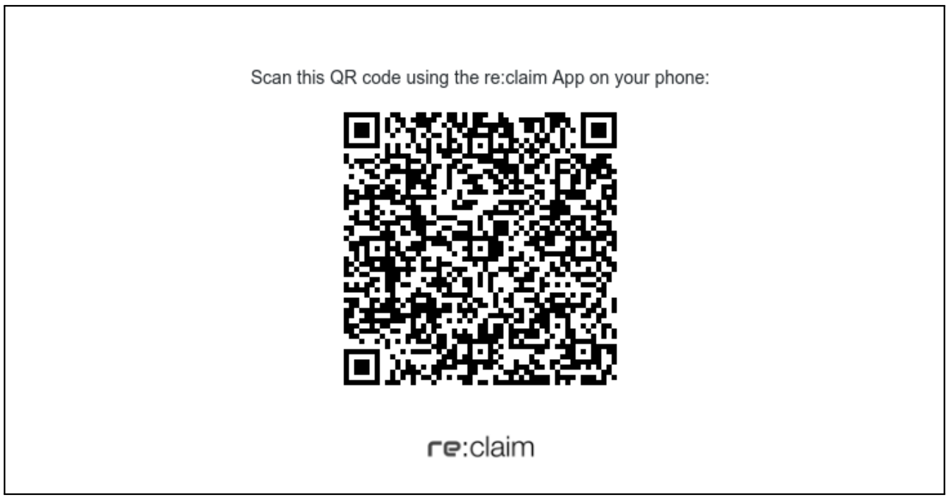


Figure 4.20.: The user scans the QR code on the website using the reclaim app.[57]

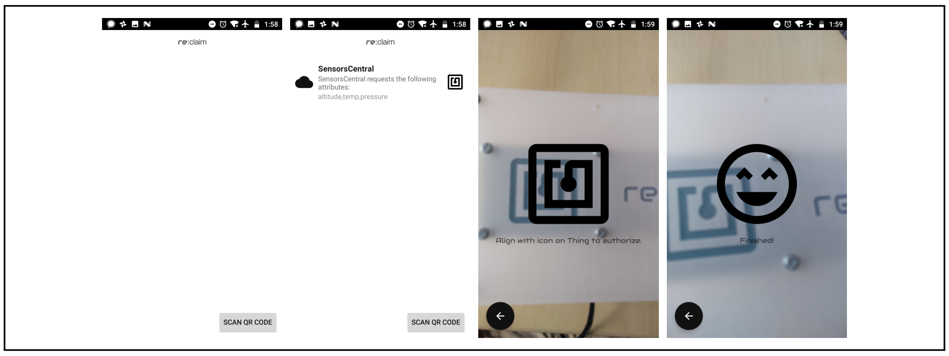
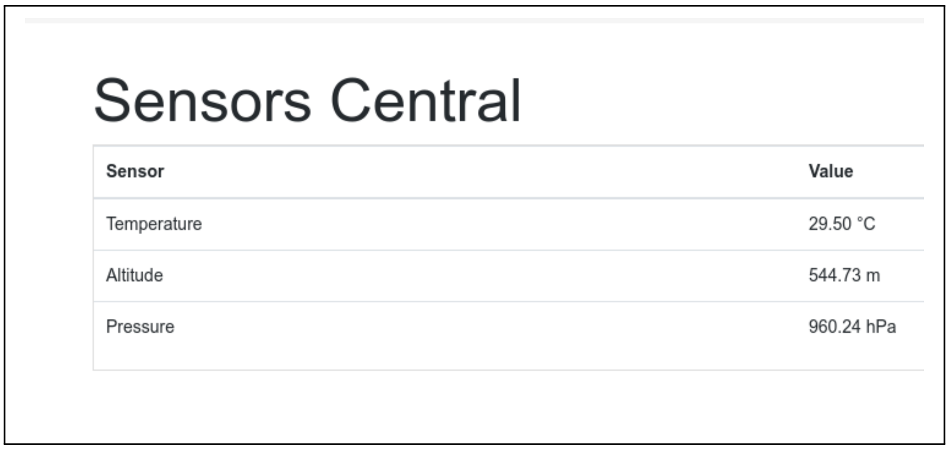

re:claim
re:claim
SensorsCentral
SensorsCentral requests the following attributes:
altitude,temp,pressure
SCAN QR CODE
SCAN QR CODE
Align with icon on Thing to authorize
Finished!

Figure 4.21.: The user initiates the NFC authorization flow with the device. [57]

relays the ticket to the web service which triggers the web service to process the ticket and retrieve the attributes from re:claimID. To visualize the result, the website displays the retrieved sensor data:

To evaluate the usability of the authorization procedure, we asked 35 users of which 32 completed the experiment to rate their experience through a standardized questionnaire including a SUS. The resulting SUS scores can be found in Appendix B.3. Figure 4.23 shows a box plot of all SUS scores. In this survey, the median SUS score is above 80 points. While this is a lower score than our web based survey, it still indicates that the usability of our system compared to other systems [92] is well above average. As the median score is above 80,3% it is positioned in the top 10% of all scores collected in [92].

Out of 34 participants that filled in the survey form, we had 22 men, 6 women and 7 who refused to answer. The self-reported age groups of the participants were 5 in 18–24, 18 in 25–34, 8 in 35–44, 2 in 45–54, 1 in 55–64 and none older. In terms of education, 1 reported a secondary school degree, 4 reported a high school graduate degree, 8 Bachelor degrees, 18 Master degrees, and 3 Doctorates.



Sensors Central

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

Figure 4.22.: After a successful authorization, the website is triggered to retrieve and display the sensor data. [57]

4.4.6. Integration into federated infrastructures

Our prototype can be integrated into existing federated identity ecosystems through the OIDC protocol. There are two approaches in order to realize such an integration: One approach is to use existing IdPs as sources of attested identity attributes which are then shared over re:claimID by the user. In this case, our re:claimID implementation acts as an OIDC RP of external IdPs.

The other approach is for the user to use re:claimID solely as a management service for his pseudonymous identities. Here, our prototype functions as the OIDC IdP and external IdPs are the respective OIDC RPs.

While we have not implemented identity federation in our prototype, we show how it is theoretically possible to setup above use cases and discuss the implications of each respective architecture.

re:claimID as downstream Identity Provider

If we configure our implementation as RP of OIDC IdPs in order to realize identity federation, websites would still interact directly with re:claimID as elaborated in our original design above. The user's client then continually retrieves current, attested identity information from one or more IdPs. However, the assertions are not necessarily forwarded to the RP and hence it cannot verify that user attributes are actually sourced at a specific IdP. In general, the OIDC protocol itself does not specify how such assertions are requested by an RP. Consequently, in order to realize this approach the OIDC specification needs to be extended or modified. However, presentation of claims retrieved from another party is supported through the "Aggregated" and "Distributed" claims features found in the OpenID Connect Core Specification [112]. Alternatively, a completely new authorization protocol which accommodates this requirement must be designed. Figure 4.24 illustrates this approach.

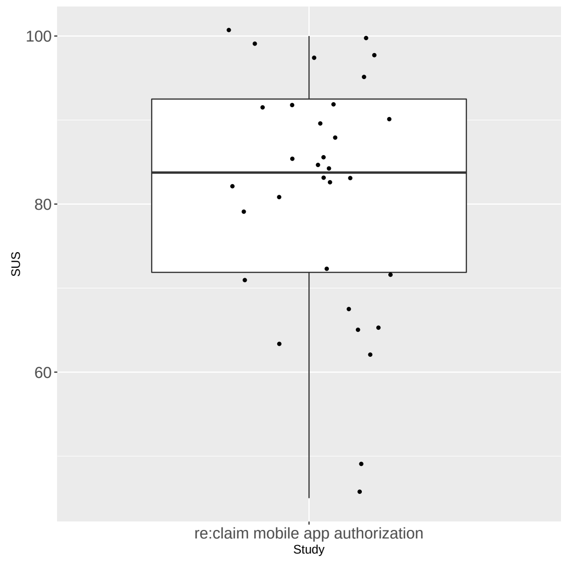


Figure 4.23.: SUS scores for the re:claimID IoT authorization study.

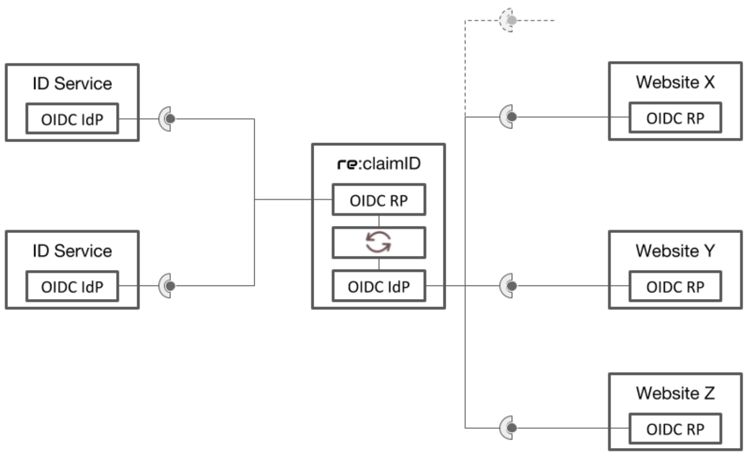


Figure 4.24.: re:claimID integrated into an existing OIDC ecosystem as RP.

The more pressing question in this scenario is how the upstream IdP of the user can be discovered. Presently, two off-the-shelf options exist for this: The OpenID Connect Discovery Protocol [111] and the id4me system [11]. The OpenID Connect Discovery relies on so-called "web finger" servers which serve the relevant metadata required to interact with the IdP. The location of a respective web finger service is derived from the user identifier, assumed to be an email address or URI.

id4me on the other hand exclusively relies on DNS for IdP discovery. It expects from the user to own a domain name which can be used as part of the user identifier and as a directory for IdP metadata. Management if this metadata and domain name is left to the end user.

Both OpenID Connect Discovery and id4me make heavy use of OpenID Dynamic Client Registration [110]. This registers the intermediary IdP as a "public" client. Public clients in OIDC are clients which can be registered without prior authorization. The trust model between the upstream IdP and the RP is out of scope for both approaches. Hence trust between a website and the third party which is asserting attributes of the user must be established out of band.

re:claimID as upstream Identity Provider

In this case, the re:claimID implementation acts as an upstream IdP for another OIDC IdP. This downstream OIDC IdP is used by RPs instead of the re:claimID service. The downstream IdP may authenticate the user and assert attributes as usual, for example through the use of email verification procedures. The advantage for RPs is that they are able to choose an IdP in their local trust domain to connect to. The users are required to interact with different IdP for each website and service, but they are able to use their re:claimID identities across all services and no data is redundantly stored across a variety of IdPs. Figure 4.24 illustrates this approach.

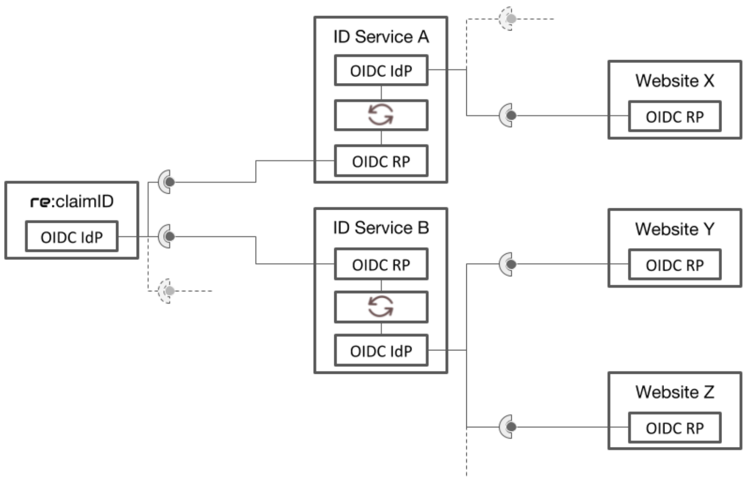


Figure 4.25.: re:claimID integrated into an existing OIDC ecosystem as IdP.

Assessment

We consider only the first approach to be compatible with the design goals and security model of re:claimID. Unfortunately, the OIDC specification does not really accommodate for federated attribute assertions which is a significant issue in the first use case. Unless the origin of an attribute is evident to the RP, this federation does not benefit the user or RP beyond what re:claimID already provides with self-issued attributes.

In the second approach, the privacy advantages of re:claimID are partially lost: IdPs are still able to track user movements across services which are registered to them. If we assume that RPs are using IdPs in their local infrastructures and trust domains, this is not an issue. However, if external service providers such as Google and Facebook are used, user tracking is still a privacy concern and aggregation of user data and the resulting implications remain unchanged. It is a subtle threat, that such an architecture may degenerate into something that is strikingly similar to the current situation which we initially tried to improve upon.

In conclusion, while both approaches are theoretically possible, we expect that any benefits are far outweighed by the elaborated practical limitations.

4.5. Alternative approaches

Our design of re:claimID presented above is implicitly tailored to be standards-compliant and name system agnostic. We achieve the standard compliance by having an OIDC layer. Through the use of ABE and an abstract and simple name system definition, re:claimID is even name system agnostic.

In this section, we present an alternative approach to re:claimID which culminated in the final design presented in this chapter. We present a modification to our re:claimID design which could improve efficiency through a tighter integration with GNS. Both approaches hinge on two specific security properties of GNS: Query privacy and non-traversable namespaces.

First, we discuss our original design in which we focused on strong security guarantees and theoretical soundness at the expense of standards-compliance and generalizability. We published this work as “Managing and Presenting User Attributes over a Decentralized Secure Name System” in the “Proceedings of the 11th International Workshop on Data Privacy Management and Security Assurance (DPM)” [123]. In this work, we rely on properties of GNS in a way that would make it difficult to implement the design on other name systems. In the following, we discuss this initial design which revolves around the idea of *identity token records* and how we improved on this idea in re:claimID.

Second, we present how re:claimID could benefit from a tighter integration with the GNS and its built-in features. Primarily, our proposed modifications would allow us to drop the requirement of the ABE encryption layer by replacing it with the less resource intensive symmetric record encryption scheme of GNS.

4.5.1. Identity token records

Like re:claimID, this approach revolves around the idea of decentralizing identity and attribute management through the use of a name system directory. It aims to provide stronger security properties with respect to authorization, availability, and freshness than existing traditional, centralized IdPs.

Individual attributes are not published in the name system as resource records. Instead, when a user decides to share attributes with a RP, the attributes are assembled into a single record. The sole audience for this record is the respective authorized RP. The contents of such a record consist of *identity tokens*, a data structure which holds the shared attributes.

We exploit the fact that a label in GNS may be treated as a shared secret as discussed in Section 4.4.1. This allows us to securely exchange information between two entities via the name system. In our original design, we call this shared secret name a *grant*. The grant is an expression of consent by the user and authorization of a RP. It exclusively allows the RP to retrieve specific records from the name system. The user authorizes a client through the generation of a *ticket* which contains the grant. As can be deduced from its name, a ticket serves the same purpose as the ticket in the design of re:claimID. However, instead of containing a reference to an ABE key, it contains the grant. The goal is that only the authorized party is able to retrieve a respective attribute set which is contained in the identity token.

In addition to the attribute sharing mechanism, we propose an authorization protocol which allows to authenticate the user. Unlike the standardized OIDC protocol, our proposal includes an implicit public key authentication of the user. This eliminates the need of external authentication mechanisms such as a password-based login at an IdP.

In the following, we elaborate on the details of the name system records and the authorization protocol.

User attributes and identity tokens

The local storage of attributes can be realized through any kind of database. In our implementation, we use GNS as a local attribute data storage. We define a special record type **ID_ATTR** that holds unencrypted identity attribute data. This approach is similar to re:claimID where we also have attribute records, but in this case records are not unencrypted using ABE and instead flagged as private in GNS. This prevents the records from being published into the DHT and consequently they cannot be resolved.

As mentioned above, users create identity tokens in order to share attributes with RPs. A user issues an identity token for each authorization of a RP. The identity token contains all the attributes which are requested by the RP. To facilitate the sharing of identity tokens via GNS, we define a record type **ID_TOKEN**. The name of an identity token record under which it is stored is the *grant*. RPs retrieve identity tokens by querying the respective grant in the user's GNS namespace. By design, and analog to ABE decryption keys in re:claimID, grants must be kept confidential by the user and the RP.

Tickets

We use a ticket $\mathcal{T}$ as a container for the grant. In the following, we refer to an RP using its public key e_{RP}. The user is identified by a public key e_{user}. The user encrypts the grant contained in the ticket in addition to the built-in record encryption of GNS. This ensures that only the owner of the respective private key d_{RP} is able to read the data.

The encryption is done using static ephemeral elliptic curve Diffie-Hellman (ECDHE) [20]. This allows the user and the RP to establish a shared symmetric encryption key $k_{\mathcal{T}}$ which is derived from the public key e_{RP} and a temporary ephemeral private key d_{ECDHE}. We define the ticket as $\mathcal{T} \leftarrow (p, e_{ECDHE}, s_{\mathcal{T}})$. It consists of the encrypted ticket payload p, the ephemeral ECDHE public key e_{ECDHE} and the cryptographic signature $s_{\mathcal{T}}$. The signature is generated over p and k using d_{user}, the private key corresponding to e_{user}. The encrypted payload itself contains an encrypted tuple (l, n, e_{user}). The tuple contains the grant l, a nonce n and the user public key e_{user}. The nonce is included to prevent replay attacks as part of the authorization protocol we discuss later. The RP must verify the authenticity of the ticket by verifying $s_{\mathcal{T}}$. The user generates the ticket $\mathcal{T}$ as follows:

$$\begin{aligned} k_{\mathcal{T}} &\leftarrow HKDF(ECDH(d_{ECDHE}, e_{RP})) \\ p &\leftarrow \mathbf{enc}(k_{\mathcal{T}}, (l, n, e_{user})) \\ s_{\mathcal{T}} &\leftarrow \mathbf{sign}(d_{user}, (payload, e_{ECDHE})) \\ \mathcal{T} &\leftarrow (p, e_{ECDHE}, s_{\mathcal{T}}) \end{aligned} \tag{4.15}$$

We use the functions and definition from Section 4.3.1. "ECDH" denotes the Diffie-Hellman key derivation function. In this respect, we note that the following holds:

$$HKDF(ECDH(d_{ECDHE}, e_{RP})) = HKDF(ECDH(e_{ECDHE}, d_{client}))$$

This allows the RP to use e_{ECDHE} which is contained in $\mathcal{T}$ in combination with its private key to decrypt the payload. Should the RP private key d_{ECDHE} become compromised at any point in time in the future, the payload can only be decrypted if the corresponding ticket containing e_{ECDHE} is also compromised.

Authorization

In the following, we outline an authorization given a RP that requests an attribute set $\mathcal{A}_{\mathcal{T}}$. We define an attribute as $a \leftarrow (key, value)$ where key is the attribute name and $value$ the attribute value. Initially, the RP issues an authorization request to access an attribute set $\mathcal{A}_{\mathcal{T} \subseteq \mathcal{A}}$ to the user. The request consists of the following parameters: The requested attribute names $\{key | a = (key, value) \in \mathcal{A}_{\mathcal{T}}\}$, a nonce n and the public key e_{RP}.

In order to approve the authorization request, the user creates an identity token including the respective attributes. The token includes a representation of the user's public key e_{user} as well as expiration information. The user creates a signature over the token with the private key d_{user}. The token and signature are encrypted using a symmetric key k_{token}. This key is derived using ECDHE from the RP public key e_{RP} and a new ECDHE private key e_{ECDHE}. As with the ticket above, only the authorized RP is able to decrypt the token. The user stores the ECDHE public key e_{ECDHE} along with the encrypted token data and signature in a resource record $\mathcal{R}_{grant}$ and publishes it under the grant:

$$\begin{aligned} k_{token} &\leftarrow HKDF(ECDH(d_{ECDHE}, e_{RP})) \\ \mathcal{X} &\leftarrow \mathbf{enc}(k_{token}, \mathcal{A}_{\mathcal{T}}) \\ \mathcal{R}_{grant} &\leftarrow (e_{ECDHE}, \mathcal{X}) \end{aligned} \tag{4.16}$$

The user responds to the RP request with a ticket $\mathcal{T}$. Similar to ABE key records in re:claimID, we reuse the grant in order to allow the RP to asynchronously retrieve updated tokens at a later time. For performance reasons, as we also discussed in Section 4.4.4 with respect to the re:claimID design, it is sensible to send the requested attribute set along with the ticket in the initial request.

As soon as the RP receives the ticket $\mathcal{T} = (p, e_{ECDHE}, s_{\mathcal{T}})$, it must verify the signature. If the signature is valid, the RP decrypts the ticket payload by using the symmetric key $k_{\mathcal{T}}$. To do so, the RP uses the key e_{ECDHE} provided in $\mathcal{T}$ and his private key d_{RP}:

$$\begin{aligned} k_{\mathcal{T}} &\leftarrow HKDF(ECDH(e_{ECDHE}, d_{RP})) \\ (l, n, e_{user}) &\leftarrow \mathbf{dec}(k_{\mathcal{T}}, p) \end{aligned} \tag{4.17}$$

Subsequently, the RP must verify that the nonce n matches the nonce from its authorization request. Then, the RP resolves the token record $\mathcal{R}_l = (e_{ECDHE}, \mathcal{X})$ from GNS. To decrypt the token record payload $\mathcal{X}$, the RP calculates the symmetric key k_{token} using the private key d_{RP} and the public ECDHE key e_{ECDHE}:

$$\begin{aligned} k_{token} &\leftarrow HKDF(ECDH(e_{ECDHE}, d_{RP})) \\ \mathcal{A}_{\mathcal{T}} &\leftarrow \mathbf{dec}(k_{token}, \mathcal{X}) \end{aligned} \tag{4.18}$$

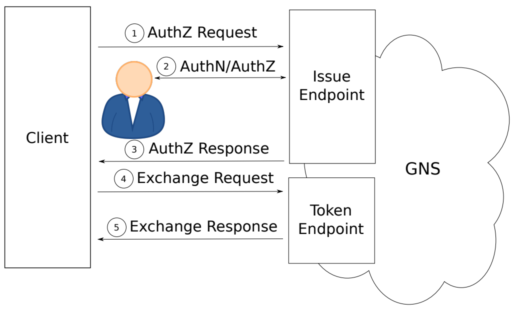
From now on, anytime the RP requires up-to-date attributes it reuses the grant to retrieve a fresh token from GNS. This assumes that the user regularly updates identity token records accordingly. On the other hand, if the user revokes access to the token record by deleting it, resolution will fail and the grant cannot be used any longer.

We implemented the above authorization protocol for a Web use case. Similar to OIDC, we utilize REST endpoints for token issuance and retrieval. As in re:claimID, this implementation consists of a user-side issue endpoint and a RP-side token endpoint. Both endpoints are connected through GNS to exchange tokens.

A user interface guides the user through authorization process. Figure 4.26 gives an overview over the authorization process. The sources of interface and an example client are available online [122, 118].

Protocol proof

As we do not use a standardized protocol such as OIDC, we cannot implicitly rely on its correctness. Hence, we use Casper [88] to verify our authorization protocol. Our model



1. A client requests authorization to access identity attributes and redirects the user agent to the user interface.
2. The user authorizes the client to access user attributes by instructing the issue endpoint to issue a ticket and an identity token.
3. The client receives an authorization response containing a ticket.
4. The client issues an exchange request and passes the ticket from the authorization response to his own token endpoint.
5. The client endpoint retrieves the token from GNS and passes it to the client.

Figure 4.26.: The authorization protocol. The client represents a RP.[123]

consists of three actors: the RP client which is the `INITIATOR` of the protocol, the `USER`, and `GNS`. The source code of the proof can be found in Appendix B.4.

In the proof we assume that the grant in the ticket and the records stored in GNS are expected to be secrets shared between the user and the RP. We assume that the user is able to establish trust in the RP public key e_{RP}. The signature s over the encrypted ticket payload includes the nonce provided by the RP in the attribute request. Hence, the nonce is essentially a challenge for the user and the authorization response with signature functions as a proof-of-possession of d_{user}.

We note that the RP is not explicitly authenticated. However, only the owner of d_{RP} is able to use the ticket and retrieve the payload contents including the grant.

Discussion

When compared to re:claimID, this approach suffers from three issues: The first and smaller issue arises from the fact that our proposed authorization protocol is not standards-compliant. This allowed us to formally prove the security of our authorization protocol without relying on additional security from, for example, the OIDC protocol. However, it also significantly and arguably detrimentally impacts the integration efforts for RPs.

The second issue stems from the tight binding to GNS. Unlike re:claimID, this approach generally not applicable when using other name systems than GNS. Names in GNS can be used as shared secrets and it ensures that they do not leak to any passive or active attackers. This is why we can use a shared secret name as the grant.

The third issue results from the randomized labels we use to store identity tokens. As the target audience for this record is exclusively the RP, any performance benefits which are usually associated with caching do not come into effect. GNS is a name system that makes heavy use of caching and relies on it to mitigate performance bottlenecks in the

DHT. We did not conduct performance evaluations of our prototype implementation, but we are confident that the system exhibits poor caching behavior.

As a consequence, we designed re:claimID with a standards-compliant OIDC interface and a name system agnostic, cryptographic authorization layer. Further, in our re:claimID prototype we took into account the caching properties of GNS and demonstrated its practicality in our performance evaluations.

4.5.2. Integrated approach

We designed re:claimID in a manner which – in theory – allows it to be implemented on top of almost any name system with a basic set of security properties. In this section, we show how re:claimID can be modified by tailoring the concept to a specific name system: GNS. On this basis, we outline how it is possible to alter the design of re:claimID to avoid the requirement of an additional cryptographic authorization layer.

Attributes

In this modified approach, the user stores each attribute $a \in A$ under randomized names n_a. This prohibits attackers from opportunistically resolving the attribute key in the user namespace $\mathcal{N}_{\mathcal{I}}$. The records are encrypted using a key derived from n_a and the namespace public key e as discussed earlier in Section 4.4.1.

Authorization

Upon receiving and consenting to an authorization request of a RP, the user creates a new namespace $\mathcal{N}_{\mathcal{I},\mathcal{RP}}$. The user further adds a delegation from his namespace $\mathcal{N}_{\mathcal{I}}$ to $\mathcal{N}_{\mathcal{I},\mathcal{RP}}$ via another randomized name n_{RP}. Inside this namespace, the user adds mappings from the requested attribute keys to the attribute records in $\mathcal{N}_{\mathcal{I}}$. Both the public key of this new namespace and n_{RP} are defined as shared secrets between the RP and the user. The name n_{RP} is transferred to the RP via the ticket $\mathcal{T}$. It replaces the nonce $n \in \mathcal{T}$ of the original re:claimID design.

The RP uses n_{RP} to retrieve its dedicated authorization namespace $\mathcal{N}_{\mathcal{I},\mathcal{RP}}$. Then, the RP is able to retrieve the $n_a \mid a \in \mathcal{A}_{\mathcal{T}}$ from this namespace. This allows the RP to retrieve the attribute values from the user namespace $\mathcal{N}_{\mathcal{I}}$.

Figure 4.27 illustrates an example where two RPs, A and B, are authorized to access user attributes.

Revocation

The revocation process is similar to the re-encryption of attributes in the original re:claimID design. In order to revoke access of a RP to a set of attributes, the user must change the randomized names n_a in the namespace $\mathcal{N}_{\mathcal{I}}$. This renders the references in the respective authorization namespace of the affected RP invalid. All affected RPs that also have access to any of the affected attributes need updated information in their

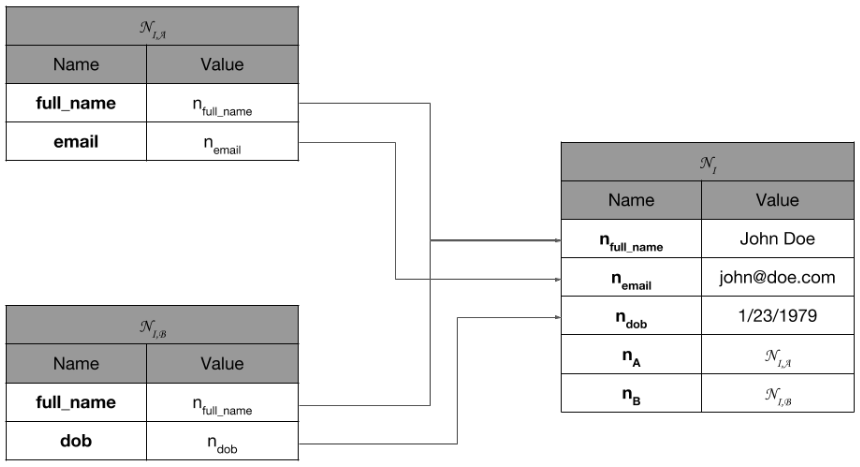

$\mathcal{N}_{I,A}$
Name | Value
full_name | n_{full_name}
email | n_{email}
$\mathcal{N}_{I}$
Name | Value
n_{full_name} | John Doe
n_{email} | john@doe.com
n_{dob} | 1/23/1979
n_A | $\mathcal{N}_{I,A}$
n_B | $\mathcal{N}_{I,B}$
$\mathcal{N}_{I,B}$
Name | Value
full_name | n_{full_name}
dob | n_{dob}

Figure 4.27.: Alternative re:claimID design using authorization namespaces in GNS.

respective authorization namespaces. Figure 4.28 shows the namespaces from above after the access of RP B was revoked.

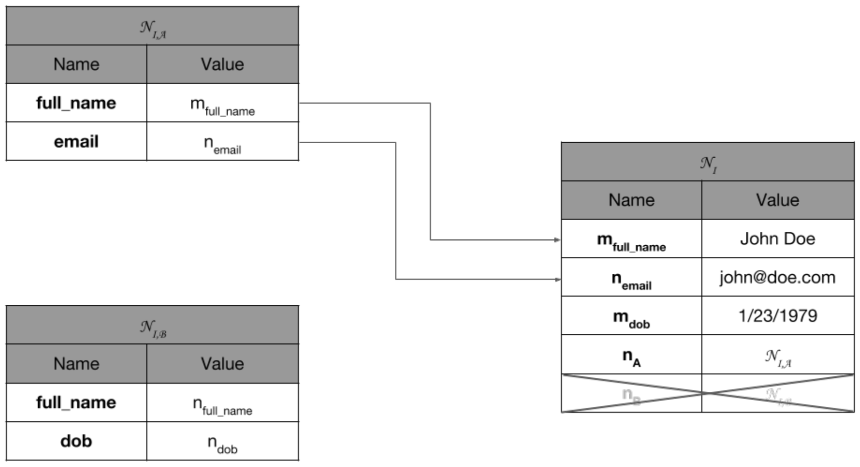

$\mathcal{N}_{I,A}$
Name | Value
full_name | m_{full_name}
email | n_{email}
$\mathcal{N}_{I}$
Name | Value
m_{full_name} | John Doe
n_{email} | john@doe.com
m_{dob} | 1/23/1979
n_A | $\mathcal{N}_{I,A}$
n_B | $\mathcal{N}_{I,B}$
$\mathcal{N}_{I,B}$
Name | Value
full_name | n_{full_name}
dob | n_{dob}

Figure 4.28.: Access revocation in the alternative re:claimID design.

Discussion

The biggest gain this approach yields is that we no longer need the ABE encryption for our access control needs. The elliptic curve (EC)-based encryption which is used in GNS is much less costly than the pairing-based CP-ABE scheme we use in re:claimID. Further, we no longer use the attribute keys as names for our attribute records. This increases privacy in so far as it is no longer possible to confirm that an namespace contains a certain attribute name. At the same time – unlike our initial approach above – we can still take advantage of caching in GNS for attribute data.

On the other hand, at the moment this approach limits the choice of name systems to GNS. The downside is that this integrated approach is neither generic nor universal.

4.6. Summary

In this chapter we addressed research questions 1 and 2 by introducing re:claimID, a decentralized service for self-sovereign identity management. In order to enable and empower the user to exercise his right of informational self-determination without having to compromise this control through the introduction of a trusted third party, we propose user-managed attributes in name systems. We advocate for the complete dissolution of a central attribute provisioning and sharing service into a decentralized query protocol and add privacy-preserving features including a cryptographic access control mechanism for sensitive attribute data. The above allows us to separate the IdP responsibilities regarding identity verification and assertion from the management and sharing of identities. Consequently, the IdP is unburdened from any liability risks which arise from the responsibility of providing a service which allows provisioning and sharing of potentially sensitive, private user information.

In order to ensure the practicality of our approach, we present a prototypical implementation of re:claimID based on the name system GNS. Despite the disruptive underpinnings of the re:claimID design, we show how it can be integrated using standard protocols. In a series of experiments we evaluated the performance and scalability of this reference implementation. We extract from the measurements results that our implementation of re:claimID is able to serve small to medium applications with up to a few hundreds of participants in production. There is room for performance optimizations such as alternative access control techniques presented in Section 4.5.2. Alternatively, boosting performance through the use of alternative cryptographic schemes, like in the lightweight ABE scheme by Yao et al. [147] is another direction of future work.

Another area of future research is the aspect of accumulated contextual metadata which may be used to link unlinkable pseudonymous identities to a single person. In order to mitigate such attacks, a holistic, full-stack approach must be taken which takes into account metadata that is incurred at physical and logical addressing on the Internet today as well as routing and service discovery. The combination of privacy-focused technologies such as peer-to-peer overlays and Tor-like routing protocols as well as secure name systems like GNS are a viable direction to take here.

In addition to further performance improvements, we aim to address the extension of re:claimID to support privacy-preserving attribute-based credentials. In general, approaches which allow RPs to establish trust in attributes shared by a user are not directly addresses by re:claimID. We present our work on this aspect including the use of succinct non-interactive arguments of knowledge in order to support privacy-preserving ABCs in the next chapter.

CHAPTER 5

Establishing trust in self-sovereign identities

In this chapter, we present our work on trust establishment in decentralized systems. Our research is directly motivated from the requirements our re:claimID design has with respect to trusted assertions on user managed identities and attribute data.

In the following, we present two approaches: The first approach revolves around the decentralized modeling and processing of attribute-based access policies. We use methods from attribute-based delegation (ABD) in combination with secure name systems to assert attributes across trust domains.

Further, we present a design of privacy-preserving credentials suitable for decentralized storages such as – but not limited to – re:claimID. We base our approach on Zero-knowledge Succinct Non-interactive Arguments of Knowledge (zkSNARKs). They allow users to present privacy-preserving credentials issued by trusted third parties without having to engage in any interactive presentation protocol with the relying party (RP). The latter is particularly common for other zero-knowledge privacy credential systems such as Idemix [22]. Through the use of non-interactive zero-knowledge (NIZK), RPs are able to retrieve and verify credentials from directory storages, as opposed to directly from users.

This chapter is partially based on a publication with the title "Practical Decentralized Attribute-Based Delegation Using Secure Name Systems" published 2018 in the 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications [113]. The sections regarding NIZK credential system is based on a publication with the title "ZKlaims: Privacy-preserving Attribute-based Credentials using Non-interactive Zero-knowledge Techniques" published 2019 in the 16th International Conference on Security and Cryptography [124].

5.1. Delegation of attributes using name systems

Our goal is to realize the existing concept of ABD into a practically usable, decentralized authorization system. In this section, we show how name systems inherently provide the capabilities and mechanisms required for ABD schemes. We base our design on an existing ABD scheme and show that it can be implemented efficiently on top of an existing secure name system.

At a glance, we present the following contributions:

- We show how secure name systems are suitable vehicles in order to implement ABD.
- We design a practical, decentralized ABD system on top of the secure name system.
- We present a prototypical implementation on top of the GNU Name System (GNS).

As we have already established in Section 2.1.2, certain name systems allow us to create secure mappings from attributes to resources. The mappings can be publicly queried by any entity. However, name registration – i.e. the creation of mappings – is only possible by the respective namespace owner. Further, the fact that name systems are agnostic towards the interpretation of record values is also relevant in the context of our approach as it allows us to integrate any delegation expression.

In the following, we argue that name systems are a suitable mechanism for ABD. Let us consider a delegation in the form $A.a \leftarrow e$: In this example, the delegating namespace owner is A and the delegated attribute is a. The delegation target is defined by an expression e. An inherent mechanism found in most name systems and namespace hierarchies is authority delegation. It allows a namespace owner to delegate the authority over names to other parties. For any regular Domain Name System (DNS) namespace delegation, e would specify the domain name of another name server, for example "ns1.example.com". This name server represents the entity that is given authority over a.

However, in ABD the delegation expression can be more complex than a single entity. As elaborated in Section 3.3.1, e can consist of various different types of delegations. In the following, we use the formalization by Li et al. regarding delegation types:

$$A.a \leftarrow B \tag{Type 1}$$
$$A.a \leftarrow B.b \tag{Type 2}$$
$$A.a \leftarrow B.b.a \tag{Type 3}$$
$$A.a \leftarrow \bigcap_{i=1}^{n} f_i \tag{Type 4}$$

A DNS namespace delegation is semantically equivalent to a *Type 1* delegation, where $e = B = \text{“}ns1.example.com\text{''}$. Most name systems inherently provide a storage, resolution and delegation mechanism for *type 1* delegations through dedicated record types such as "NS" (DNS) or "PKEY" (GNS). Other delegation expressions are uncommon in existing name systems.

In the remainder of this section, we define a record format for *attribute delegation records* which allow us to model any delegation type as a name system record. Further, we show how to resolve attribute-based encryption (ABE) delegation chains using a *delegation chain discovery* algorithm.

In the following, we first establish a set of security properties our system must provide before we detail our record types and algorithms. In addition to our ABD scheme, we outline an access control policy evaluation protocol for our design which allows us to use ABD in an actual authorization use case. Finally, in order to demonstrate the viability of our contribution, we present a reference implementation in the context of a real world scenario.

5.1.1. Security properties

As our approach above makes only general assumptions, basically any name system can be a suitable vehicle for ABD. However, in practice only those name systems with strong security guarantees are reasonable choices for an implementation. An insufficiently resilient name system is subject to a variety of attacks that fatally impair the ABD system. Before we delve into our design, we discuss the security properties that we aim for:

Authenticity and integrity The primary goal of our system is to ensure that any given attribute delegation chain is authentic. This implies that we must be able to verify each delegation $A.a \leftarrow e$ in a delegation chain. We already established above that we plan on using a name system in order to store and retrieve individual delegations. Further, a delegation in the form $A.a \leftarrow e$ can be found in the namespace of an entity A. A name system which features *data origin authenticity*, would inherently provide us with authentic and correct delegations. Most secure name systems such as DNS Security Extensions (DNSSEC), Namecoin and GNS fit this bill and realize this property through the use of cryptographic record signatures.

Availability In order to use ABD and the resulting delegation chains to evaluate policies and make authorization decisions, delegations must be available. If unavailable, authorization decisions are possibly based on false negatives and access is unjustifiably denied. On the other hand we can rule out false positives as long as our mechanism provides authenticity and integrity guarantees of delegations as discussed above. Ideally, a name system provides availability of records in the face of an attacker as defined in the re:claimID threat model. Blockchain-based name systems address this by having all records replicated by all participants. Unfortunately, this mechanic does not necessarily provide strong availability guarantees: In practice, blockchains are notoriously prone to even archaic attacks on peer-to-peer networks [89, 51]. DNS-based designs suffer from the already discussed issues, most importantly the inherent susceptibility to censorship. We abide by our assessment on record availability in the context of re:claimID where we settled on GNS. We consider it to be a well-rounded system for an implementation in order to ensure delegation availability as well due to its resilient underlying distributed hash table (DHT).

Confidentiality and privacy Bulk collection and enumeration of attribute delegations is unwanted, as it is prone to expose organizational trust relationships. Consequently, this disqualifies name systems that do not consider use cases where mappings and record values are confidential such as Namecoin or DNS. Neither protect the contents of resource records or namespaces. The issue with Namecoin can be generalized to any blockchain-based design that does not have any additional technical mechanisms in place to provide respective protections. At the same time, DNSSEC suffers from a design weakness that results in a privacy issue regarding namespace enumeration[10]. While mitigations exist, they are not yet widely deployed [54]. GNS on the other hand provides us with namespace privacy and even delegation confidentiality given that the delegate attribute is a shared secret. This allows us to design an ABD scheme which facilitates what we call *hidden delegations*. We give details on the use case and mechanism of hidden attributes in Section 5.1.4.

5.1.2. Design

In response to our considerations on security properties above, we design and implement our ABD system on top of GNS. It provides us with the strongest security models under our threat model. By limiting our design to a name system which exhibits the same properties as GNS, we take a different route than for re:claimID which we designed to be name system agnostic. On the other hand, through this choice we gain additional leeway which allows us to define additional features such as hidden delegation records. Further, we plan to use the system to support re:claimID attribute assertions. Since we already settled for GNS in our prototype implementation, introducing yet another name system would only increase complexity.

We first design a delegation record type which allows us to resolve ABD asserted attributes. Further, we present a way to combine ABD-asserted attribute-based credentials which are suitable to be provisioned using re:claimID. By means of a delegation chain discovery algorithm we facilitate access control and policy evaluation through the use of ABD.

In the following, we use the name system function definitions from Section 4.3.1. Further, we use Li's notation for attribute delegations from Section 3.3.1 which we reiterated in the beginning of this chapter.

Attribute delegation records

We use the operator $\leftarrow$ to denote attribute delegations. We define that the *issuer* and the *delegated attribute* are found on the left side of the operator. Accordingly, the *delegation subject expression* is found on the right side. In the original design by Li arbitrarily long delegations on the right-hand side of the expression are not allowed. This limitation is only imposed to simplify the proposed resolution procedures. We lift this restriction by imposing *issuer-side storage* of delegations in our design. Delegations in the form $A.a \leftarrow B.b_1.[...].b_n$ are valid in our design. Issuer-side storage implies that attribute delegations are managed and provisioned by the issuer entity as opposed to the subject. This is unlike *subject-side storage*, where the subject is managing and provisioning the attribute it is delegated. As a result, our design mandates that delegations are always stored with the namespace owner.

In order to support delegation resolution, we first define the resource record type "ATTR". We use this record type to specify attribute delegations in the format as presented in Section 3.3.1. Given a delegation in the form $A.a \leftarrow e$, we map it into a namespace as follows: We define an entity as

$$A := (\mathcal{I}_A, \mathcal{N}_A) \tag{5.1}$$

where $\mathcal{I}_A$ and $\mathcal{N}_A$ are an identity and a namespace associated with A as used in Section 4.3.1, respectively. The delegated attribute a is the name of a record in the namespace $\mathcal{N}_A \in A$. The value of the record contains e, a delegation expression that has the semantics of ABD delegation types.

In order to support all four types of attribute delegations, we define an appropriate wire format which allows to model any delegation expression e. Specifically, the value of an "ATTR" record contains a *delegation set* data structure. The record wire format for both the record and a delegation set can be found in Appendix B.4. A delegation set consists of a number of *delegation set entries*. A single delegation set entry represents exactly one delegation of type $A.a \leftarrow f$. The expression f represents delegation types 1 through 3. Hence, in order to model those types of delegations, a delegation set contains a single entry.

To model the type 4 delegation, a delegation set contains n entries. Combined, the n set entries represent a delegation in the form $A.a \leftarrow \bigcap_{i=1}^{n} f_i$. A type 4 delegation constitutes a logical "AND". Hence, all expressions f_i must be resolvable in order to complete a delegation and establish its correctness. It is not possible to model a logical "OR" using a single delegation set. Instead, to model this case we simply specify multiple delegation set records under the same attribute a.

Algorithm 1 details how a single attribute delegation is published:

```
Algorithm 1: add/update
    input : Attribute $a$
            User identity $\mathcal{U} = (\mathcal{I}_\mathcal{U}, \mathcal{N}_\mathcal{U})$
            Delegation expression $e$
  1 $\mathcal{R} \leftarrow (type = 'ATTR', e, ttl = 1h)$;
  2 publish($\mathcal{N}_\mathcal{I}, a, \mathcal{R}$);
```

To revoke attribute delegations, the issuer simply removes the mapping from his namespace. The delegations will remain resolvable until the respective records of the attribute delegation expire in the name system.

Credential records

In order to establish delegation chains from delegated attributes to a specific attribute-based credential (ABC), we differentiate between attribute delegation records and a credential record. Credential records are backward pointing delegations $A.a \leftarrow B$ of type 1 which typically come at the end of a delegation chain. Unlike attribute delegation records which we always store in the issuer namespace $\mathcal{N}_{\mathcal{A}} \in A$, credential records are stored in the namespace of the subject B.

Subjects are provided with the credential record value in the form of a third party attested ABC. Such ABCs are issued by authorities such as employers, email providers, or nation states. We refer to the set of ABCs issued to B as $\mathcal{C}_B$. In our design, the user manages and shares attribute credentials in the form $A.a \leftarrow B$ as private resource records in re:claimID. We define the attribute format in Appendix B.5. Revocation of attribute credentials is performed using traditional means such as having the issuer provide a signed expiration date along with it. While more sophisticated revocation mechanisms are conceivable, this is out of scope of our ABE system.

We note that the ABCs could alternatively be represented as type 1 attribute delegations. However, this is disadvantageous as such an approach allows enumeration of all entities that were issued credentials which match a certain delegated attribute. From a privacy perspective this is not wanted. Further, we can leverage re:claimID as a mechanism which allows users to manage and share attributes in a self-sovereign fashion.

Delegation chain discovery

In order to facilitate the resolution and verification of attribute delegations, we design a custom resolution algorithm. We assume that any resolution starts with an initial attribute and a set of target credentials. The resolution algorithm then follows the delegation chain in accordance with the delegation logic of Role-based Trust Management (RT) as defined in Section 3.3.1.

The resolution is successful when a delegation chain $\mathcal{D}_{A.a,B}$ is found from the issuer namespace A, over the attribute a to a subject credential subset $\mathcal{C}_{A.a} \subseteq \mathcal{C}_B$. However, resolution can only succeed if all attribute delegations $d \in \mathcal{D}_{A.a,B}$ are resolvable *and* B is able to provide the respective set of credentials $C_{A.a}$. We define the delegation chain resolution algorithm in the recursive evaluate procedure. We note that an actual implementation of this procedure should define a discrete recursion depth limit in order to ensure termination.

Initially, the procedure is called with the start of the delegation chain $\mathcal{D}_{A.a,B}$ and a credential set $\mathcal{C}_B$. In lines 1-4. the procedure checks if the given attribute delegation is already a match against any provided credential in $\mathcal{C}_B$. If no match is found, the attribute delegation record set $\mathcal{S}_{\mathcal{R}}$ is resolved from the name system in line 5. If the resolved resource record contains a delegation set with a single entry, the expression e is of type 1-3, otherwise type 4. In any case, we proceed to evaluate all delegation sets

Procedure evaluate

```
input :Issuer $A = (\mathcal{I}_A, \mathcal{N}_A)$
       Attribute $a$
       Remainder $r = xs \vee \emptyset$
       Subject $B = (\mathcal{I}_B, \mathcal{N}_B)$
       Credential set $\mathcal{C}_{A.a} \subseteq \mathcal{C}_B$
foreach $E.e \in \mathcal{C}_{A.a}$ do
    if $E = B \wedge e = a$ then return $TRUE$;          /* Delegation found */
    ;
end
$\mathcal{S}_\mathcal{R} \leftarrow$ resolve($\mathcal{N}_A$, $a$);
result $\leftarrow$ FALSE;
foreach $\mathcal{R} \in \mathcal{S}_\mathcal{R}$ do
    result $\leftarrow$ FALSE;
    foreach all $e \in \mathcal{R}$ do                          /* Type 4 */
        switch $e$ do
            case $E$ do                                         /* Type 1 */
                if $\emptyset = r$ then result $\leftarrow$ FALSE;        /* Invalid path */
                else
                    result $\leftarrow$ result $\wedge$ evaluate($E, x, s, B, \mathcal{C}_{A.a}$);
                end
                break;
            end
            case $E.e$ do                                       /* Type 2 */
                result $\leftarrow$ result $\wedge$ evaluate($E, e, r, B, \mathcal{C}_{A.a}$);
                break;
            end
            case $E.e.f$ do                                     /* Type 3 */
                result $\leftarrow$ result $\wedge$ evaluate($E, e, fr, B, \mathcal{C}_{A.a}$);
                break;
            end
        end
    end
    if $TRUE = result$ then break;              /* Found valid delegation */
end
return $result$;
```

$\mathcal{R} \in \mathcal{S}_{\mathcal{R}}$ starting in line 7. If any delegation set can be followed to a credential of B, the delegation is valid and we can terminate the algorithm. In order to ascertain this case, for each expression e represented by a delegation set $\mathcal{R}$, we perform SDSI-style rewriting [25] of the original delegation $A.a$. We combine SDSI-style rewriting rules and the backward resolution of a delegation graph in Li's approach for RT_0 as discussed in Section 3.3.1. As mentioned previously, we enforce the issuer-side storage of attribute delegations by storing them in the issuer namespace. We can see here that this allows us to mitigate the otherwise necessary and complex unified approach by Li which is able to resolve both backward *and* forward search of the delegation graph. Our procedure follows the rewrite-resolve-check pattern described in the following text until we can match a credential against a delegated attribute.

In lines 11-17, we handle the case of a type 1 delegation and must distinguish between two cases: If the delegation remainder r is not empty, it consists of an attribute chain xs where x is the first attribute in the chain and s the rest. In this case, we rewrite the initial delegation attribute $A.a$ to $E.x$ and shorten the remainder to $r' = s$ by recursively calling the evaluate procedure. This results in a reduction of the remainder and delegation expression. If the delegation remainder is empty, we have reached an invalid chain. We already verified that no credential in $C_{A.a}$ matches $A.a \leftarrow B$. A delegation in the form $A.a \leftarrow E$ without a remainder r leaves us with no further options to continue the delegation chain at this point.

In lines 18-20, we handle type 2 delegations of the form $A.a \leftarrow E.e$. We simply rewrite the delegation attribute by recursively calling the evaluate procedure with $E.e$ as the delegation attribute and an unmodified remainder r. Hence, the type 2 delegation does not change the remainder.

When encountering a type 3 delegation $A.a \leftarrow E.e.f$, we must add f to the existing remainder r in lines 22-25. Then, we rewrite the delegation attribute $A.a$ with $E.e$ by recursively calling the evaluate procedure with a new remainder $r' = fr$. The type 3 delegation results in an expansion of the remainder and delegation expression.

In our proposed procedure, type 4 delegations in the form $e := \bigcap_{i=1}^{n} f_i$ are processed individually as discussed above. The procedure at that point branches out for every $f_i \in e$. The branches are eventually reduced to a single evaluation result when all paths have been evaluated.

5.1.3. Access control and policies

Above, we established how attribute delegation chains are resolved and verified given a set of user credentials. Now, we discuss how the delegation chain discovery algorithm is used to perform authorization and access control using policies. In order to authorize subjects to access a resource, we define policies that contain delegated attributes: We define that an entity $\mathcal{V}$ creates an access policy $\mathcal{P}_{\mathcal{V},r}$ that specifies a set of attributes $(a_n \mid a \in \mathcal{A}_{\mathcal{V}})$ needed to access the resource r. Any subject $\mathcal{S}$ is granted access to r if it is able to present a set of credentials $\mathcal{C}_{\mathcal{V}.x} \mid x \in \mathcal{P}_{\mathcal{V},r}$ that satisfied the policy $\mathcal{P}_{\mathcal{V},r}$.

$\mathcal{V}$ validates that for each $x \in \mathcal{P}_{\mathcal{V},r}$, there exists a delegation chain $\mathcal{D}_{\mathcal{V}.x}$ from $\mathcal{V}.x$ to $\mathcal{C}_{\mathcal{V}.x}$. We define that the verifier $\mathcal{V}$ is always the issuer for all attributes $x \in \mathcal{P}_{\mathcal{V},r}$. In order to achieve this functionality, we define the verify procedure.

```
Procedure verify
    input : Issuer $V = (\mathcal{I}_V, \mathcal{N}_V)$
            Policy $\mathcal{P}_{\mathcal{V},r}$
            Subject $S = (\mathcal{I}_S, \mathcal{N}_S)$
            Credential set $\mathcal{C}_\mathcal{P}$
1   result $\leftarrow$ TRUE;
2   foreach $x \in \mathcal{P}_{\mathcal{V},r}$ do
3       result $\leftarrow$ result $\wedge$ evaluate($V, x, \emptyset, S, \mathcal{C}_\mathcal{P}$);
4   end
5   return result;
```

However, the verifier cannot know the credentials $\mathcal{C}_S$ of the subject or which subset $\mathcal{C}_\mathcal{P} = \{\bigcup_{x \in \mathcal{P}} \mathcal{C}_{V.x} \mid \mathcal{C}_{V.x} \in \mathcal{C}_S\}$ it must provide to satisfy $\mathcal{P}$ a priori. Similarly, the subject may initially not be aware of the contents of $\mathcal{P}$. Even if the contents of $\mathcal{P}$ are known to S, the credential subset $\mathcal{C}_\mathcal{P}$ that matches $\mathcal{P}$ must first be identified. To enable this, we define another procedure: collect. We note that this procedure includes the creation of a superset of the credentials of a user. This is a highly inefficient but necessary part of the procedure.

We use the above procedures and integrate them through the following authorization protocol. Figure 5.1 illustrates the protocol flow. The steps in the protocol are as follows:

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

(2) V denies the access request and responds with a policy $\mathcal{P}_{\mathcal{V},r}$.

(3) S uses the procedure **collect** which yields the credential subset $\mathcal{C}_\mathcal{P} \subseteq \mathcal{C}_S$ that satisfies $\mathcal{P}_{\mathcal{V},r}$.

(4) S accesses the resource r again and adds $\mathcal{C}_\mathcal{P}$ to the request.

(5) V processes the credential set $\mathcal{C}_\mathcal{P}$ using the **verify** procedure which performs delegation chain discovery to verify that a delegation chain $\mathcal{D}_{\mathcal{V}.x} \mid x \in \mathcal{P}_{\mathcal{V},r}$ exist.

(6) V grants S access to r only if all delegation chains can be found.

This process is simple enough to be integrated into standard protocols such as REST request and the use of respective authorization headers. At the same time it is generalized in a way which theoretically also allows integration into sophisticated authorization flows and trust negotiations, for example in the context of User-Managed Access (UMA).

```
Procedure collect
    input : Issuer $V = (\mathcal{I}_V, \mathcal{N}_V)$
            Policy $\mathcal{P}_{\mathcal{V},r}$
            Subject $S = (\mathcal{I}_S, \mathcal{N}_S)$
            Credential set $\mathcal{C}_S$
1  $\mathcal{C}_\mathcal{P} \leftarrow \emptyset$;
2  foreach $x \in \mathcal{P}_{\mathcal{V},r}$ do
3      $\mathcal{C}_{min} \leftarrow \mathcal{C}_S$;
4      found ← FALSE;
5      foreach subset $\mathcal{C}_i \subseteq \mathcal{C}_S$ do
6          if evaluate($V, x, \emptyset, S, \mathcal{C}_i$) then
7              found ← TRUE;
8              if $|\mathcal{C}_{min}| > |\mathcal{C}_i|$ then $\mathcal{C}_{min} \leftarrow \mathcal{C}_i$;
9          end
10     end
11     if found then $\mathcal{C}_\mathcal{P} \leftarrow \mathcal{C}_\mathcal{P} \cup \mathcal{C}_{min}$;
12 end
13 if found then return $\mathcal{C}_\mathcal{P}$;
14 else return $\emptyset$;
```

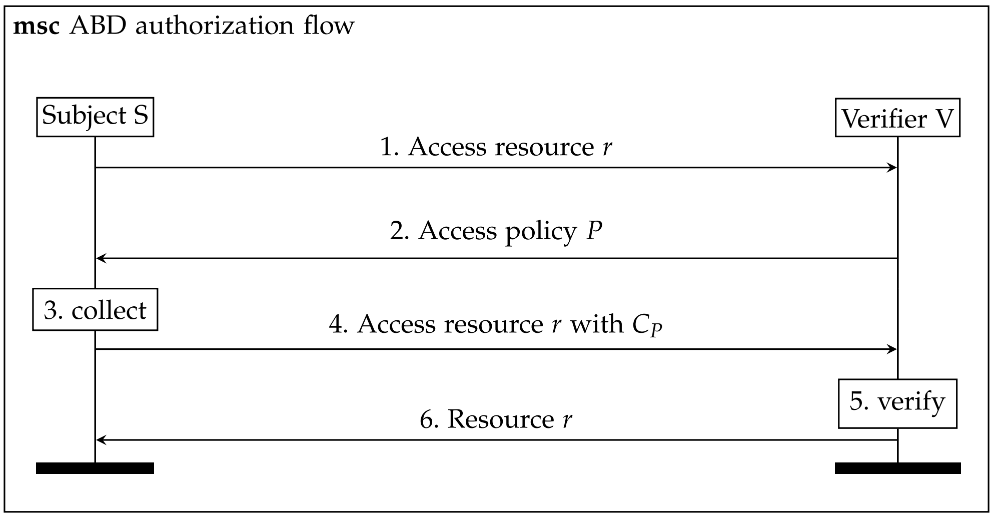


Figure 5.1.: ABD authorization protocol.

5.1.4. Hidden delegations

Since we use GNS as our underlying directory service for attribute delegations, it is possible to hide attribute delegation records. This is done by leveraging the query privacy and namespace enumeration protection mechanisms of GNS which we discussed in Section 4.4.1.

In the following, we discuss how we can use the fact that a name l can be treated as a shared secret between two parties that want to exchange information via the name system. Specifically, we use this property in order to allow issuers to hide attribute delegations. For example, consider a type 2 delegation $A.a \leftarrow B.b$. In GNS, the resolution returning this hidden delegation is only possible if the shared secret attribute a issued by A is known a priori. Consider the following use case as an example how such a delegation is useful:

A whistleblower D has co-conspirators X_i. Each is issued a credential C_i of the form:

$$D.s_i \leftarrow X_i$$

The attributes s_i are randomly generated, shared secrets between D and X_i, respectively. Further, the group uses a whistleblowing service S which allows them to access and upload files. To this end, D delegates the secret attribute p to the co-conspirator attributes:

$$D.p \leftarrow D.s_1, D.s_2, ...$$

The attribute $D.p$ is used by S in the access policy $P = D.p$. When S evaluates the policy, any attribute query performed in the delegation chain discovery algorithm is protected. No observer is able to efficiently deduce any party X_i or delegation attribute by monitoring the network. Through this approach, we facilitate private policy enforcement using hidden attributes.

5.1.5. Example scenario

Before we go into details on our implementation, we give an example scenario which is in need of a flexible and decentralized ABD system. Specifically, we integrate our ABD design in the context of our research in the area of privacy friendly anti-doping control systems as conceptualized in [125]. The use case allows us to investigate how an authorization policy and relevant attribute delegations look like in a real world use case. In short, we aim to improve the anti-doping process by providing a secure and privacy-preserving service S to athletes and anti-doping organizations.

Motivation

The premise is that doping control officers (DCOs) need to use a service to retrieve the current location of an athlete in the field in order to conduct unannounced doping controls. The peculiarity of this use case is found in the organizational authorization requirements and trust structures. At a glance, the following entities play a role in this scenario:

- World Anti-Doping Agency (WADA)
- National anti-doping organizations (NADOs)
- Doping control officers (DCOs)
- Doping control subcontractors
- Doping control system service provider
- Athletes

The organizational relationship between NADOs and WADA is not a strict hierarchy as one might initially assume. National anti-doping organizations (NADOs) merely incorporate and adhere to the world anti-doping code which is defined by the World Anti-Doping Agency (WADA). Every single NADO is responsible for organizing and executing doping tests for athletes in their respective regional domain.

At the same time, this task is commonly delegated to subcontractors. This loose organizational structure makes a central management of credentials difficult. We assume that WADA is neither interested in nor authorized to manage attributes of DCOs from subcontractors. The same is true for the doping control system service provider. But, WADA is able to assert that a NADO does adhere to its code.

At this point, one might be tempted to resort to a traditional public key infrastructure (PKI). WADA could acts as CA NADOs as sub-CAs. NADOs could in turn delegate attributes to DCOs. By design PKIs such as X.509 are often limited to bind a key to a subject where the subject is uniquely identified by a globally unique name. Any participating entity, from WADA to subcontractor and service provider to Athlete, needs an identity asserted from this PKI. Further, X.509 does not directly address attribute delegation and resolution. In this construct, issuance and revocation of attributes at runtime is tedious.

Solution

In order to address the scenario as defined above, we propose the use of ABD-based access control models. We note, however, that this problem and our proposed solution are not unique to the above scenario. Any use case in which the organizational structures are similar to the ones elaborated above, our proposed ABD delegations are viable options in order to model trust relationships.

In the following, we present a respective setup of delegation chains that reflect the complex entity relationships:

$$S.user \leftarrow WADA.nado.dco \tag{5.1}$$
$$WADA.nado \leftarrow NADA \tag{5.2}$$
$$WADA.nado \leftarrow USADA \tag{5.3}$$
$$NADA.dco \leftarrow C_1.dco \tag{5.4}$$
$$USADA.dco \leftarrow USADA.contractor.dco \tag{5.5}$$
$$USADA.contractor \leftarrow C_2 \tag{5.6}$$
$$C_2.dco \leftarrow C_2.employee \cap C_2.controller \tag{5.7}$$
$$C_1.dco \leftarrow Alice \tag{5.8}$$
$$C_2.employee \leftarrow Bob \tag{5.9}$$
$$C_2.controller \leftarrow Bob \tag{5.10}$$

The service S delegates the attribute *user* to *WADA.nado.dco*. We assume that S uses this attribute to authorize its users. Essentially, S uses a type 3 delegation to grant any entity access that is considered a *dco* by a WADA NADO. For this, S assumes or knows that *WADA* delegates the attribute *nado* to all NADOs.

In this case, example NADOs are the German "Nationale Anti Doping Agentur" *NADA* and the "U.S. Anti-Doping Agency" *USADA*. The NADOs are further assumed to delegate the attribute *dco* to their subcontractors using a type 2 delegation. In this case, $NADA$ delegates the *dco* attribute to a single contractor C_1.

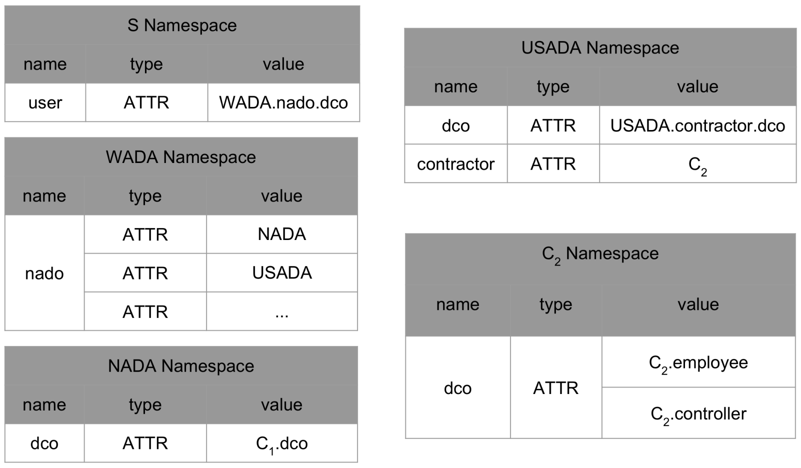
$USADA$ uses a more complex delegation: It uses the attribute *contractor* to define all subcontractors currently in use. This allows $USADA$ to dynamically add or remove contractors without having to modify the delegation logic. For both NADOs, the subcontractors may choose to either delegate the *dco* attribute to an attribute expression that is more meaningful to the contractor or directly assign it to a user.

What is important is that *WADA* is not necessarily aware of the attribute delegation in use by S. Trust relations in this design are purely on a need to know basis. In traditional PKI approaches, the service S would have to delegate authority over the *user* attribute to *WADA*. In our approach, this is not required as the delegations are always defined with the issuer and never with the subject.

Figure 5.2 illustrates how namespaces could be organized in order to reflect the above setup. However, in the figure we omit the namespace of contractor C_1 as it is empty.

Figure 5.3 further allows us to illustrate the delegation chain discovery for our scenario:

1. *S.user* is resolved to a single record with the delegation set entry *WADA.nado.dco*, a type 3 delegation.
2. *WADA.nado* resolves to two records containing one delegation set entry each: *NADA* and *USADA*. The rewritten expressions *NADA.dco* and *USADA.dco* are added to the graph.
3. *NADA.dco* resolves to a record containing the delegation set entry $C_1.dco$. Bob does not have a credential to satisfy this delegation.
4. *USADA.dco* resolves to $USADA.contractor.dco$. The attribute $USADA.contractor$ resolves to C_2 leading to $C_2.dco$.



S Namespace		
name	type	value
user	ATTR	WADA.nado.dco

WADA Namespace		
name	type	value
nado	ATTR	NADA
	ATTR	USADA
	ATTR	...

NADA Namespace		
name	type	value
dco	ATTR	C_1.dco

USADA Namespace		
name	type	value
dco	ATTR	USADA.contractor.dco
contractor	ATTR	C_2

C_2 Namespace		
name	type	value
dco	ATTR	C_2.employee
		C_2.controller

Figure 5.2.: Namespaces in our reference scenario.

5. $C_2.dco$ resolves to a single record containing two delegation set entries representing the type 4 expression $C_2.employee \cap C_2.controller$.

6. Bob has the credential $C_2.employee$ that matches the delegation set entry $C_2.employee$. The graph is backtracked but the delegation set containing $C_2.controller$ is not yet fulfilled. Bob's credentials are checked against $C_2.controller$ and the credential $C_2.controller \leftarrow Bob$ satisfies the delegation set in (5).

7. The delegation graph is backtracked further until $S.user$ is reached and the delegation chain is successfully discovered in (8).

5.1.6. Reference implementation

In the following, we present details on our implementation of the ABE design above. This reference prototype [116] integrates with re:claimID in order to enable users to manage ABCs. We created a proof-of-concept application [117, 121] which supports the ABD authorization protocol as specified in our design. In the next sections, we discuss the details of our implementation.

We define the record types for attribute credentials and attribute delegations for GNS and we present an application programming interface (API) specification which can be found in the GNUnet source code [115].

Credential management

In order to enable the management of credentials and delegations, we implement a record type for attribute credentials "CRED" and for attribute delegations "ATTR" according

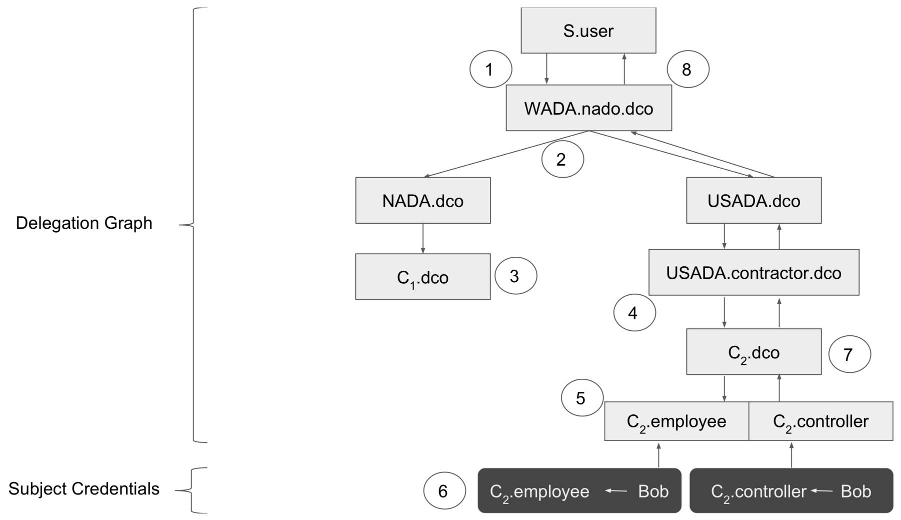


Figure 5.3.: Delegation graph in the reference scenario from *S.user* to *Bob*.

to the wire format found in Appendix B.1. To facilitate the creation of user credentials, we implement a credential component and a respective API as part of GNUnet.

For example, if the issuer is a company A that decides to issue an attribute credential *employee* to a subject B, the following command is executed using the gnunet-credential command-line interface (CLI) tool:

```
gnunet-credential --issue --ego=A --attribute=employee --subject=B \
                  --ttl=1y
```

The command returns a credential encoded in a Base32 string which is valid for one year. This allows the issuer to transfer the credential to the subject, which in turn stores it in its local credential wallet. In order to store the credential, the subject uses our CLI tool gnunet-namestore which is used to manage local namespaces:

```
gnunet-namestore --add --zone=B --name=test \
                 --type=CRED --value=DATA
```

The primary function of the gnunet-namestore CLI tool is to manage local GNS namespaces. We define our credential wallet as a set of private records of type “CRED”. Private records are never published in the name system and are consequently not disclosed to other parties.

Delegation management

In order to create an attribute delegation, we define the use of “ATTR” records. Let us consider the following example: A web service S needs to define a policy which allows a group of users to access the service’s resources. Let us further assume, that the service calls any authorized user “user”. This means that in order for any entity to gain access, there must be a delegation path from *S.user* to a respective credential.

Let us assume there is a company A of which all employees must be able to access the resources. First, S adds a type 1 attribute delegation $S.company \leftarrow A$ using the `gnunet-namestore` CLI, effectively delegating the name "company" to A:

```
gnunet-namestore --add --zone=S --name=company \
                 --type=ATTR --value=A
```

S now defines that any entity which is delegated the name "employee" by A is considered a valid user. Consequently, S uses a type 2 delegation $S.user \leftarrow A.employee$ to delegate the name "user" to any entity A calls "employee":

```
gnunet-namestore --add --zone=S --name=user \
                 --type=ATTR --value="A employee"
```

Now, the company A consists of various branches. Hence, A delegates the attribute "employee" to all of its branches using a type 3 delegation $A.employee \leftarrow A.branch.employee$:

```
gnunet-namestore --add --zone=A --name=employee \
                 --type=ATTR --value="A branch.employee"
```

This mitigates the need for a central employee repository for A and delegates the responsibility of employee definitions to the branches.

Finally, S might decide that in addition to a credential which proves that the user is an employee of A, a service-specific credential "authorized" is needed. Consequently, S modifies the delegation of "user" using a type 4 delegation $S.user \leftarrow S.authorized \cap A.employee$:

```
gnunet-namestore --add --zone=A --name=division \
                 --type=ATTR --value="S authorized, A employee"
```

As a result, the namespaces of S and A are populated with the respective ABD resource records as illustrated in Figure 5.1 and Figure 5.2. We note that delegations of the name "authorized" are not stored as delegations in the namespace because we can assume that they are credentials as issued above. We omit the definition of $A.branch$ delegations and respective employee credential creation for the sake of brevity.

Name	Type	Value
user	`ATTR`	S authorized, A employee

Table 5.1.: An example namespace for a service using attribute delegation records.

Attribute collection and verification

In order to collect credentials which satisfy an attribute as part of an authorization flow, the CLI utility `gnunet-credential` is used. Consider the above scenario and in addition a user B that needs to collect credentials that satisfy the attribute *user* as defined by S. In this case, B executes:

Name	Type	Value
employee	ATTR	A branch.employee
branch	ATTR	B

Table 5.2.: An example namespace for a company using attribute delegation records.

```
gnunet-credential --collect --ego=B \
                  --issuer=S --attribute=user
```

If the delegation chain is resolvable, this call returns a set of credentials CS which B can present to another party in order to prove that $S.user \leftarrow B$ holds. Using this credential set CS, the other party can verify that claim by executing:

```
gnunet-credential --verify --issuer=S --attribute=user \
                  --subject=B --credential=CS
```

This command returns the delegation chain if it could be found and an error otherwise.

5.1.7. Caching considerations

In this section, we discuss performance implications of the GNS for our ABD system. Distributed name systems like DNS and GNS make use of query response caching to improve lookup performance. Responses are cached in the network by servers or nodes that are not authoritative, i.e. they are not the authorities over the particular records contained in the response. Caching in name systems is particularly effective for two reasons: First, many participants query the same, popular names. Second, namespaces are organized in a delegation hierarchy. This hierarchy is reflecting various cache-levels.

In GNS, the time it takes to resolve a query that is not locally cached depends on the network topology and replication setting of the DHT. All nodes in the network cache observed responses. Additionally, client resolvers have local response caches. As the lookup of a locally cached delegation does not require any network queries its performance impact is negligible. Hence, the time it takes to query a delegation chain in our GNS-based ABD system is primarily determined by the performance of the underlying peer-to-peer overlay; in case of GNS this is the DHT.

Given this observation, we can make use of the response caching mechanisms in our ABD system: Previously queried delegations are cached locally by the requester and on intermediate participants in the network which consequently improves resolution times of all delegation chains that contain a particular cached delegation. As a rule of thumb, cache hits are more likely for delegations which that are repeatedly resolved.

With respect to our example scenario from Section 5.1.5 consider the resolution of a delegation chain $S.user \leftarrow ... \leftarrow C_1.dco$ corresponding to Alice's credential $C_1.dco \leftarrow Alice$. In this case, a minimum of five namespace queries are required assuming the caches are empty. After an initial resolution, we can assume that at least the delegations in (1), (2), (3) and (4) of our example are cached.

If we now consider the delegation chain for Bob with credentials $C_2.employee \leftarrow Bob$ and $C_2.controller \leftarrow Bob$, the cached delegation chain from $S.user$ (1) to $USADA$ can be leveraged. Only the remaining delegations (5), (6) and (7) need resolution.

We can generalize this observation in the following. Given a delegation setup as follows:

$$A.a \leftarrow B.b.c \tag{5.1}$$

$$B.b \leftarrow C, D \tag{5.2}$$

$$C.c \leftarrow E.d \tag{5.3}$$

$$D.c \leftarrow F.d \tag{5.4}$$

After the resolution of a delegation chain $A.a \leftarrow ... \leftarrow E.d$ corresponding to a credential $E.d \leftarrow Alice$ we can safely assume that (5.1), (5.2) as well as (5.3) are cached. If the caches were empty, each namespace query induces a network operation with an duration of $t_{uncached}$ where

$$t_{uncached} := \sum_{i=0}^{n_{delegation}} t_i^{query}$$

and $t_i^{uncached}$ is the time it takes to resolve a delegation i from a namespace by performing a query without caching. $n_{delegation}$ is the length of the final delegation chain.

If subsequently a delegation chain for another subject with a credential $E.d \leftarrow Bob$ is resolved, we expect resolution times to be faster because the delegation chain is already cached. The resolution time is estimated to be

$$t_{cached} := \sum_{i=0}^{n_{delegation}} t_i^{cached}$$

where t_i^{cached} is the time it takes to resolve a delegation from a namespace if the response is already cached.

Finally, if the delegation chain for a subject $F.d \leftarrow Eve$ is resolved, we can leverage cache hits for the first part of the chain (1), (2) and (3) but expect a cache miss for (4). Hence, the resolution is expected to take longer than for $E.d \leftarrow Bob$, but it is still expected to be faster than the initial resolution for $E.d \leftarrow Alice$.

By combining the above observations, we expect the time it takes to verify a delegation chain for any caching name system to be

$$t := \sum_{i=0}^{n_{cached}} t_i^{cached} + \sum_{i=0}^{n_{uncached}} t_i^{uncached}$$

where $n_{delegation} := n_{cached} + n_{uncached}$.

In the specific case of GNS, resolvers locally cache all responses. As lookup of a cached delegation does not require any network operations we define $t_i^{cached} \in O(1)$. Consequently, the time t it takes to query a delegation chain in a GNS-based ABD system is $t := \sum_{i=0}^{n_{uncached}} t_i^{uncached} + C$ where C is a constant factor.

In summary, the time complexity of a delegation chain lookup in GNS primarily depends on the number of uncached delegations. This implies that for users that have not previously interacted with a verifier, authorization time can be noticeable. However, once authorized all following authorizations are likely near instantaneous, only requiring network queries when delegations in the delegation chain expire. Further, we can expect network performance to increase with a in increase in users and use of the system. This is a reasonable assumption as DNS resolution of domain names also exhibits this property.

5.2. Non-interactive zero-knowledge credentials

Above, we presented an approach which allows users and RPs to establish trust in attributes through the concept of attribute delegations. However, resolution of trust chains may be unnecessarily complex depending on the actual scenario. For instance, if the user and RP have a trusted third party Identity Provider (IdP) in common, the resulting trust chain is trivially short. In such cases, presentation of classical ABCs is a logical approach which does not require the introduction of unnecessary complexity. However, the use of classical ABCs such as X.509 certificates reveals potentially sensitive attribute values such as name, age, or social relationships to the RP while the actual value might not even be required.

Precisely for this use case, privacy-preserving attribute-based credential (PP-ABC) systems have been proposed in the past [100, 99, 22]. PP-ABCs are built using mathematical protocols which allow users to prove statements over attributes. This mitigates the need for revealing the attribute value itself if the statement is sufficient for the RP. The most prominent example which is commonly given that applies to this use case is age verification: The specific date of birth is irrelevant if the RP is only interested in whether or not the user is above a certain age.

Traditional approaches such as those mentioned above often require the user and the RP to engage in an interactive proving protocol. This prevents RPs from retrieving and validating statements if the user is offline. This property is disadvantageous for identity systems such as re:claimID where centralized directories are replaced by decentralized peer-to-peer storages. One approach to remedy this issue is to require that proofs of statements are predefined by the issuer of the ABC. Whenever a RP requests a specific statement from a user, he must interact with the IdP in order to acquire it.

In our contribution, ZKlaims, we present a design for PP-ABCs which can be presented non-interactively and at the same time allow users to decide what statements are made as long as they do not violate the underlying attribute. We achieve this property through the use of non-interactive zero-knowledge proofs.

The non-interactive property allows us to integrate ZKlaims into re:claimID. Proofs can be stored and retrieved through its resilient, decentralized delivery mechanism. We build upon recent advancements in the area of zkSNARKs in order to design NIZK credentials which we present in the following sections. We further present an evaluation of our reference implementation and discuss its performance traits. Finally, we present our prototypical implementation as part of re:claimID.

5.2.1. Background

zkSNARKs are a theoretical class of proofs which satisfy a specific set of formal properties in order to realize a NIZK proof. Its origin lies in the research area of verifiable computation (VC) schemes [102, 56].

There are two actors in any VC scheme: The prover and the validator. In our case, the prover is the user and the validator is a RP. VC schemes allow users to prove the correct evaluation of a function given a set of inputs. This facilitates outsourcing of computations onto a third party, for example because of significant complexity and high performance requirements. At the same time, the result can be efficiently verified in order to make sure that the computation was actually performed. Because of this, VC schemes are optimized towards providing an efficient way to verify computation results. On the other hand efficiency with respect to the proving processes is commonly not prioritized.

zkSNARKs [8] were designed as an extension to a specific class of VC schemes in order to add zero-knowledge properties to the schemes. Popular verifiable computation schemes which allow to build zkSNARK proofs include Pinoccio and a scheme by Groth et al. [102, 56].

In zkSNARKs, the verifier of a computation must be able to verify the proof – i.e. the result of the computation – without a private input called "witness". This "witness" is only known to the prover. The verification of a result does not require the private function input. Hence, the verifier is able to verify a computation result without knowledge of what was actually the subject of the computation. We generalize the following high-level primitives of a zkSNARKs scheme:

$$Setup(\varphi) \rightarrow (pk, vk) \tag{5.5}$$

$$Prove(pk, \vec{a}, \vec{x}) \rightarrow \pi \tag{5.6}$$

$$Verify(vk, \pi, \vec{x}) \rightarrow \{0, 1\} \tag{5.7}$$

Initially, we must establish a constraint system φ. Our constraint system is a set of linear constraints which are internally translated into arithmetic circuits by the zkSNARKs scheme. The constraint system is a blueprint that allows us to define the shape of ground truths. It also allows us to derive the proving key pk and verification key vk through the use of a setup procedure. The constraint system – and consequently both pk and vk – are public. They are meant to be published and known by both prover and verifier, respectively.

For a constraint system to be useful in our use case, it must be constructed in a way that supports proofs on ABCs. In order to achieve this, we define the setup and construction of a ZKlaims constraint system construction in the design section. To generate a proof π, the prover must supply proof input vectors $\vec{x}$ and $\vec{a}$ as well as the proving key pk. $\vec{x}$ is the property to prove and $\vec{a}$ are private attributes only known to the prover. Hence, $\vec{a}$ is the "witness" to our zkSNARKs scheme. We discuss the contents of the input variables along with the ZKlaims ABCs later in this section. In order to validate a proof π, the verifier uses the verification key vk and the public input vector $\vec{x}$ as inputs to the validation procedure. The verification result is either valid or invalid.

5.2.2. Design

In the following, we present our design of ZKlaims which satisfies the following three requirements:

Statements on credentials: ZKlaims must allow users to generate proofs on third party issued credentials. The user must be allowed to freely choose the statement $\vec{x}$. The verifier must be able to verify that the statement is true with respect to the third party issued credential without the knowledge of the actual credential value $\vec{a}$.

Non-interactive presentation: ZKlaims must allow verifiers to prove the correctness of a statement non-interactively, e.g. without online interaction with the user or the credential issuer.

Selective disclosure: The user should have the option to selectively disclose credential values if necessary.

For ZKlaims, in addition to the prover and verifier we define a third actor: The credential issuer. The issuer is a trusted third party that is issuing ABCs to users which take the role of provers. The credential contents are private and represent the private input vector $\vec{a}$ of the proving procedure. The prover is able to make any statement regarding its issued credentials and create a proof π which asserts that the statement is valid. The verifier is an entity which requires the prover to prove the validity of a certain statement. In our use case, this statement is based on a specific ABC. The proof π is zero-knowledge in that the verifier only learns whether or not the statement on a credential is valid. The contents of the credential are not disclosed to the verifier. Figure 5.4 illustrates our scenario including the generation of the keys, a proof and its verification.

We use this illustration in the following sections to explain the design and usage of ZKlaims.

Attributes and credentials

First, we define a ZKlaims credential $C := (\vec{a}, \vec{y}, S)$. We define the input vectors $\vec{a}$ and $\vec{y}$ as bit vectors:

$$\vec{a} := \vec{a_0} \mid \dots \mid \vec{a_n} \textit{ where } \vec{a_i} \in \{0,1\}^* \tag{5.8}$$

$$\vec{y} := \vec{h_0} \mid \dots \mid \vec{h_n} \textit{ where } \vec{h_i} = hash(a_i) \tag{5.9}$$

$\vec{y}$ represents the private attributes of the statement $\vec{x}$. $\vec{p}$ and $\vec{r}$ are variable and may be chosen by the prover as part of the proving process. S is a signature over $\vec{y}$ and is created

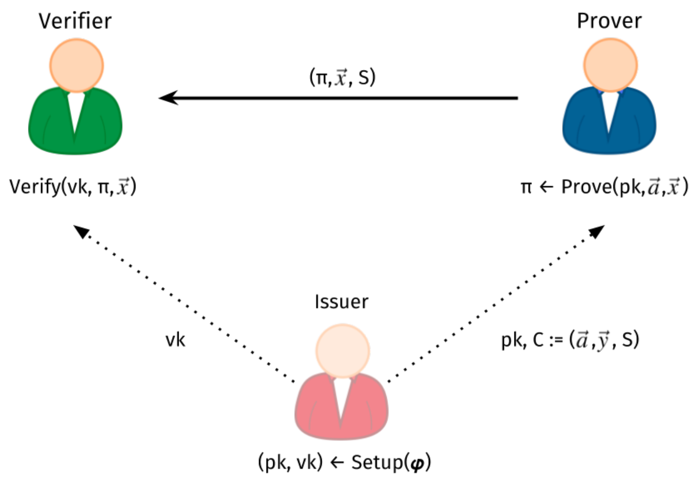


Figure 5.4.: Actors and protocol primitives in a ZKlaims use case.

by the issuer. As part of the vector $\vec{y}$, the issuer must include a salt in order to prevent trivial brute-force attacks against the hash preimage. The signature is created through traditional public-key cryptography. This allows a verifier to establish trust from his set of trusted, third party credential issuers to the credential $\mathcal{C}$ and verify its authenticity.

We note that $\vec{a}$ contains $n+1$ elements but there are only n attributes while the last element is reserved. This is a design choice for the following reason: We define the last element $\vec{a_n}$ to be a unique identifier of the credential. It is a random nonce generated by the credential issuer when the credential $\vec{y}$ is issued. The nonce ensures that the credential is unique across subjects even if their attributes $\vec{a}$ are the same.

We expect an issuer to provide a mechanism that allows the user to retrieve credentials $\mathcal{C}$ through a secure communication channel. This transfer is out of scope of this work, but for web-based use cases it can be realized through a traditional TLS channel in combination with client authentication. Then, the transfer of the credential from the issuer to the user can be performed through a simple download procedure. The user then stores the credential in a wallet on a local storage under his control.

Constraint system and keys

As discussed in the background section, we must define the constraint system φ. The entity responsible for creating the constraint system is the credential issuer because it is the entity which is authoritative over what kinds of attribute credentials it plans to issue to its users. A ZKlaims constraint system φ_{zklaim} must be setup so that it enables a prover to prove statements on credentials in the form $\mathcal{C}$. Figure 5.5 illustrates our circuit construction.

Constraint systems process input variables in an algebraic circuit and output a boolean return value. Hence, it is possible to combine multiple constraint systems into one new

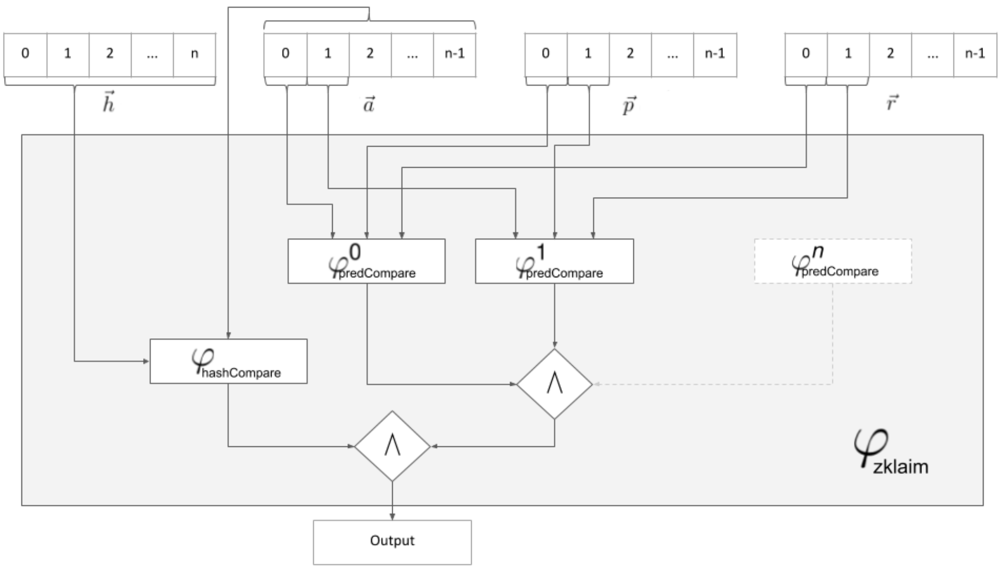


Figure 5.5.: ZKlaims constraint system φ_{zklaim}.

constraint system. In our design, we define the linear constraint system φ_{zklaim} as a combination of $n + 1$ sub constraint systems:

$$\varphi_{zklaim} := \varphi_{hashCompare} \wedge \left(\bigwedge_{i=0}^{n} \varphi^i_{predCompare} \right) \tag{5.10}$$

The *hashCompare* constraint allows the prover to verify that the user-provided private input vector $\vec{a}$ matches the credential $\mathcal{C}$ contents. The second class of constraint systems are used to model, prove and verify comparative statements on the private input $\vec{a}$. For this the issuer must pre-determine the number n of attributes that $\vec{a}$ may contain as it determines the upper bound of sub constraint systems of type $\varphi_{predCompare}$. As illustrated in Figure 5.5, each $\varphi^i_{predCompare}$ constraint takes exactly one $a \in \vec{a}$ as input whereas the $\varphi_{hashCompare}$ constraint system takes the whole input vector $\vec{a}$. As constraint systems are rigid in this regard, a change in the number of attributes requires a regeneration of the constraint system φ_{zklaim}.

Proving

Using φ_{zklaim} any entity is able to generate the public proving key *pk* and verification key *vk* using the respective $Setup(\varphi_{zklaim})$ procedure of the zkSNARKs scheme. The key *pk* is used by the user in order to prove the validity of statements on his attribute credentials. Each $\varphi^i_{predCompare}$ may be used by the prover to impose a predicate $\vec{p}_i$ with respect to a reference value $\vec{r_i}$. The hashed attribute references in $\vec{y}$ are combined with the above into the public proof input vector $\vec{x}$:

$$\vec{x} := \vec{y} \mid \vec{p} \mid \vec{r} \tag{5.11}$$

By default, each $\vec{p_i}$ is initialized as a no-op dummy operation which always evaluates to true. In order to create a statement on an attribute $\vec{a_i}$, the user sets the predicate $\vec{p_i}$ to any combination of $<$, $=$ and $>$ or their respective complements $\not<, \neq, \not>$. This predicate is used in combination with a reference value $\vec{r_i}$ which contains a value that the corresponding attribute $\vec{a_i}$ is to be checked against with the predicate $\vec{p_i}$. As an example, to create a proof input which is supposed to verify that a user is born before a certain data, the reference value $\vec{r_i}$ for the "data of birth" attribute would be set to a certain timestamp in the past which reflects the age barrier. The position n of the reference value is defined by the issuer through the constraint system. The predicate is set to $\not<$. Such a proof input $\vec{p}$ allows a user to prove that he is over a certain age.

In general, $\vec{p}$ can be chosen arbitrarily by a prover. However, $\varphi_{hashCompare}$ is used to import the requirement that any prover must be able to provide a witness in the form of a pre-image to $\vec{y}$, namely $\vec{a}$. As already mentioned above, $\vec{a}$ – and in particular $a \in \vec{a}$ – serves as a secret that the prover must present in the proving process as part of a witness to the $\varphi_{hashCompare}$ constraint. Hence, only the subject which is in possession of a credential $\mathcal{C}$ from the issuer is able to satisfy the constraint system.

In order to validate a proof π, the prover must apply the public proof input $\vec{x}$, the proving key pk as well as the private input vector $\vec{a}$ to satisfy the constraint system φ_{zklaim} and generate a proof π. The user generates a proof as follows:

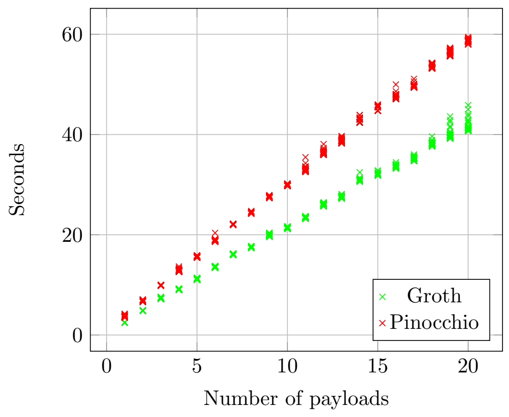
$$\pi \leftarrow Prove(pk, \vec{a}, \vec{x}) \tag{5.12}$$

The user is able to provide the hashes in $\vec{y}$ and the pre-image $\vec{a}$ from the credential $\mathcal{C}$ to satisfy the $\varphi_{hashCompare}$ constraint system. The public input vector $\vec{x}$ is built using $\vec{y}$, the predicate inputs vector $\vec{p}$ and the reference value vector $\vec{r}$ which represent the statements made by the user on the attributes in $\vec{a}$. We expect that the predicates are defined a priori, for example, through a negotiation between the verifier and the user. The verifier can request from the user to provide a specific proof including certain predicates and thus defines the respective predicate variables $\vec{p}$ and $\vec{r}$.

After the user calculates the proof π, it can be presented to a verifier along with $\vec{x}$. We define a ZKlaims context $(\pi, \vec{x}, S)$ which – due to the non-interactive nature of the proof – can be persisted by the user and non-interactively retrieved and verified by a verifier.

Verification

It is not necessary for a verifier to directly interact with the prover to verify a proof π. However, upon retrieving the ZKlaims context $(\pi, \vec{x}, S)$ and before the verification of π, the verifier must verify the signature S over $\vec{y} \mid y_i \in \vec{x}$. Using this information, the verifier proceeds to retrieve the correct proving key vk from the trusted issuer and uses it to verify the proof π:



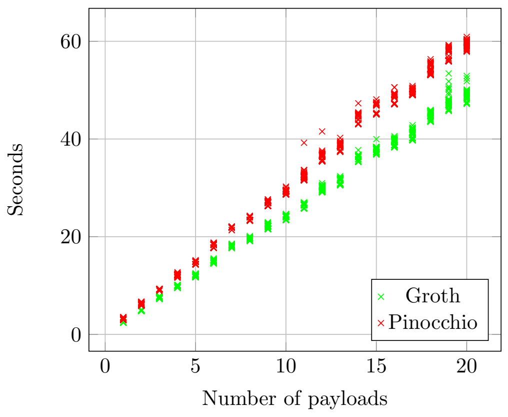


Figure 5.6.: Left: time required to derive verification key vk & proving key pk from the constraint system φ_{zklaim}.
Right: time required to create proof π depending on the number of attribute payloads.

$$result \in \{FALSE, TRUE\} \leftarrow Verify(vk, \pi, \vec{x}) \tag{5.13}$$

The verification function yields $TRUE$ if the user was able to provide inputs to φ_{zklaim} that satisfy the underlying constraint systems. It is essential that verifiers check that the predicate inputs and reference vectors $\vec{p}$ and $\vec{r}$, which are provided by the user as part of the ZKlaims context, are semantically what they expect them to be. Especially if the verifier specifically requested a predicate to be proven, such as "$age \geq 18$", the respective predicate (greater or equal) as well as input variable to check against must be correctly set.

5.2.3. Example use

In order to illustrate the use of a ZKlaims credential, let us assume that a user must provide a proof π_{dob} to be of legal age in order to access a website. The website accepts credentials issued by a local government. Let us further assume that this government issues credentials C_{dob} which are prefined to contain a single attribute $\vec{a} := \vec{a}_{dob}$, containing the date of birth of a subject. Issued credentials further include the $\vec{y} := \vec{h}_{dob}$ and corresponding signature S in order to provide a trusted assertion of the attribute value to be used in the verification process.

The website requests a proof from its users with a reference value $\vec{r}_{dob}$ that contains the closest date which would pass as a legal age given the current date. Additionally, the requested $\vec{p}_{dob}$ then would consist of a $\geq$ predicate. Any user in possession of a C_{dob} may calculate π_{dob} as follows:

$$\pi_{dob} \leftarrow Prove(pk, \vec{a}_{dob}, \vec{x}_{dob} := \vec{y}_{dob} \mid \vec{p}_{dob} \mid \vec{r}_{dob}) \tag{5.14}$$

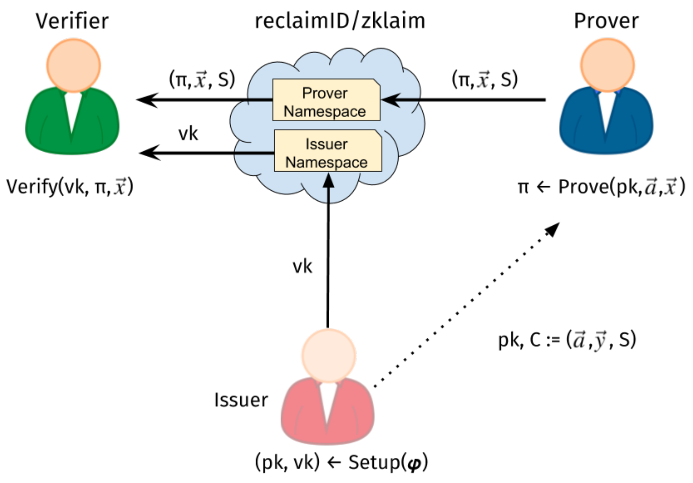


Figure 5.7.: ZKlaims integration with the reclaimID identity provider.

The user may then provide the proof to the website. The website can verify the proof and authorize access to users if the verification is successful. It does not learn the actual date of birth of the user.

5.2.4. Reference implementation

Our reference implementation[1] is built on top of the *libsnark*[2] library. zkSNARKs are based on verifiable computation schemes. While libsnark supports a variety of different schemes including Pinocchio [102], we use the scheme of Groth [56] which is also readily available. We settled on Groth because it exhibits better performance than the other schemes available in libsnark.

Performance and evaluation

We evaluated the performance of ZKlaims for issuing, proving and verification with respect to the number of attributes in a single credential. Due to technical limitations imposed by the underlying constraint system we must use fixed size inputs to the constraint system. We define a collection of attributes that fits into a single input as a *payload*. In our instantiation, we use a payload size of five attributes. In our implementation, a single payload holds up to five attributes. Our test setup consists of an Intel Core i7 7500U 3.2 GHz with 16 GB of RAM.

In Figure 5.6 we can see that the time it takes to construct the issuer constraint system increases linearly with the number of attribute payloads in our credential. It takes roughly 2.5 seconds to build a constraint system that supports five attributes in a single payload and increases by the same amount for every additional set.

[1]`https://gitlab.com/kiliant/zklaim`, accessed 2019/01/08
[2]`https://github.com/scipr-lab/libsnark`, accessed 2019/01/08

The time it takes a prover to construct a proof depending on the amount of payloads defined by the issuer constraint system can be found in Figure 5.6. We can see that just like the initial construction of the constraint system, proof construction time also increases linearly with the number of payloads. Creating a proof in the case of a single payload takes roughly 2.4 seconds and increases by the same amount for every additional set.

While we evaluated the time it takes the verifier to validate proofs, the results suggest that the impact is negligible: In ZKlaims, proof verification is simply a matter of evaluating a polynomial function and measured times range below 10 milliseconds. The issuer constraint system needs to be created only once initially. Proofs need to be constructed every time a verifier has a new request regarding the predicates it needs proven. Once a proof for a combination of predicates exists, it can be stored and presented non-interactively to any concerned verifier.

In addition to the evaluation of performance, we also take a look at the size of proving and verification keys as well as proof size dependent on the number of payloads. In Table 5.3, we find that the minimum size of a proving key is roughly 9 MB. With every payload – i.e. every set of five attributes – supported by the issuer constraint system, the proving key increases in size by 40 to 50 MB. At 20 payloads, this results in a 182 MB proving key. Due to this size constraint, issuers should either limit the number of supported attributes or bootstrap dedicated constraint systems so that a prover is only required to handle proving keys for attributes actually relevant to them.

At the same time, the verification key takes a minimum of 784 bytes and its size increases by 700 to 1000 bytes for each additional payload. The space burden thus clearly lies with the prover. Proofs itself are of constant size at 137 bytes. This is good news with respect to storage capabilities provided by our name system-based directory service in re:claimID since 137 bytes are manageable.

Payloads	***pk* in MB**	***vk* in bytes**	**Proof in bytes**
1	8.65	784	137
5	43.15	1543	
10	86.94	2493	
15	133.37	3443	
20	174.51	4436	

Table 5.3.: Key and proof sizes depending on the number of payloads.

Integration

We designed ZKlaims specifically for decentralized identity provider services that require or support non-interactive presentation of identity attributes. To publish and propagate ZKlaims objects such as the issuer credential system, verification key and proofs, we use re:claimID.

Given the strict size constraints of proving and verification keys, and due to technical constraints of name systems, we assume that verification keys must be addressed out-of-band. However, since the authority over the issuer constraint system – and with it the keys – is the issuer itself and keys can be presumed to rarely change, out-of-band distribution using traditional means such as web servers is feasible.

On the other hand, distributing credentials and, more importantly, proofs using any of the above name system-based delivery systems is certainly possible. Users create proofs and authorize verifiers to retrieve and verify them from the name system in an efficient, completely decentralized fashion.

In our implementation, the issuer publishes the ZKlaims constraint system ϕ, the verification key vk and the proving key pk in re:claimID. This record is published in a namespace which is owned by the issuer. This allows any prover to retrieve the issuer's constraint system and proving key and to verify its integrity and use it as inputs in proving and verification procedures. It should be noted here that proving keys over 63 kilobyte would not fit into a single record of the GNS directory due to the record size limit and would have to be split into multiple records.

Figure 5.7 illustrates the integration of ZKlaims with the re:claimID identity provider. The prover shares the proving context including the proof π, the proof input $\vec{x}$ and the credential signature S with the verifier over re:claimID. This is done by having the prover store the proving context as an attribute record in reclaimID. This attribute is shared with a verifier through an out-of-band authorization protocol such as OpenID Connect. Our reference implementation can be found online as part of the GNUnet peer-to-peer framework[3].

We note that integration in the OpenID Connect 1.0 (OIDC) flow as supported by re:claimID is theoretically possible. However, the standard does not cover a way to request credentials from specific third party identity providers and much less the request of specific statements or proofs on ABCs. Consequently, while our implementation within re:claimID supports ZKlaims, the standardized authorization protocol does not. We consider the integration and possible extension of the OIDC to be out of scope of our research and merely a necessary standardization effort.

5.3. Summary

In this chapter, we primarily addressed our third research question which revolves around the challenge of establishing trust in identities which are shared over self-sovereign identity systems such as re:claimID.

In the first part of this chapter, we show how the general idea of ABDs-based trust delegation can be used to establish trust in self-sovereign identities. While ABD systems have been proposed by other authors in the past, practical implementations which retained its decentralized and distributed nature were missing. Our contribution is

[3]`https://gnunet.org/git/gnunet.git/tree/src/zklaim?h=zklaim`, accessed 2019/02/13

the continuation of their work and the transfer into a real-world use case by means of designing and implementing a practical ABD trust management system.

Our design combines the role-based policy language RT_0 for distributed trust establishment with the decentralized infrastructure of name systems. The system enables the creation and specification of trust relationships without the need for centralized trust anchors and policy databases.

We propose a trust chain resolution strategy for name systems which, according to the theoretical groundwork provided by Li et al. [84], ensures that chains are consistently resolvable. In a reference scenario, we demonstrate that there is a practical benefit of a decentralized ABD in real world applications. Further, we show how secure name systems such as GNS provide a suitable basis for a prototypical implementation.

In future work, we consider enhancing our system with distributed trust negotiation instead of our basic authorization protocol, such as the one proposed by Li et al [84]. Discovery and selection of trust chains could be delegated to an authorization framework such as UMA in order to allow RPs to use a standardized protocol for standardized authorization management. Further, we did not discuss usability aspects of delegated attribute management systems: Creating policies based on decentralized attributes requires a priori knowledge or out-of-band discovery of possible delegations. Also, it would be valuable to understand to what depth attributes are realistically delegated in practice.

The other contribution to research question three in this chapter is ZKlaims. Given a trusted IdP between user and RP, we present the design for non-interactive privacy-preserving credentials based on a non-interactive zero-knowledge protocol. It enables users to present statements on third party asserted ABCs to RPs without having to disclose the potentially sensitive information contained in the credentials. We have shown how zkSNARKs can be leveraged for decentralized identity sharing systems such as re:claimID. Our performance evaluations of ZKlaims show that its application is practical and where integrators must account for additional resources.

Regarding the integration of ZKlaims in a protocol such as OIDC we face open issues regarding spotty standardization. Current specifications do not accommodate authorization requests by RPs which allow to indicate the type of attribute which is requested. While we do not see any fundamental problems with designing such a scheme on top of OIDC, this aspect is out of scope of this work.

In summary, we have presented two approaches to privacy-preserving trust establishment for decentralized directory services and self-sovereign identity systems. We achieved our goal to complement re:claimID with respect to third party attribute assertions using approaches which do not degrade the decentralized character of re:claimID.

CHAPTER 6

Conclusions

In this thesis we approached the complex topic of decentralized identity management in order to mitigate the increasing privacy concerns which arose over the past years. We demonstrated how self-sovereign management and sharing of identities and data *by the user* can be separated from authoritative identity assertions issued by Identity Provider (IdP)s *for the user*.

Our contributions include an abstract design for a decentralized directory service architecture on top of secure name systems. In order to allow users to manage and control access of third parties to identity information stored in the decentralized directory we proposed the use of attribute-based encryption (ABE). In addition to this novel approach to identity management, we presented a standard-compliant protocol layer to ease integration efforts by relying parties (RPs) and limit changes in the user experience. We implemented a prototype of our design on top of the GNU Name System (GNS) and the use of off-the-shelf ABE implementations. Our performance and usability evaluations of our reference implementations show that the system is suitable for deployment in practice. We demonstrated that the system can be used not only for end user identity management and attribute sharing, but also for device identities and sensor data in the Internet-of-Things (IoT). In order to complement our self-sovereign identity system, we propose a decentralized, attribute-based delegation (ABD) system built on the same building blocks. It facilitates the implementation of flexible trust models across trust domains. Additionally, we present a privacy credential system compatible with decentralized directory services.

In the following, we discuss how and to what degree our contributions address our initial research questions. Further, we conclude this thesis by giving an overview over open research questions. Those constitute aspects which we identified in the course of our research, but did not investigate as they do not directly fall into the scope of this thesis.

6.1. Contributions to research questions

Our contributions were motivated by the research questions we defined in Chapter 1 of this thesis. We note that while our contributions address the research questions as elaborated in the following, we do not consider our approaches to be the only possible solutions the those challenges. We presented the current state of the art in Chapter 3 and outlined our contributions beyond that. Future research may find alternative answers to our research questions.

Research Question 1: How can we ensure the users right to informational self-determination regarding his digital identities? In order to support the user in exercising his right for informational self-determination, we propose the decentralization of identity provider services and the separation of identity verification and personal data sharing. We designed re:claimID in order to address this challenge and to realize a decentralized, open service for self-sovereign identity management and attribute sharing. We propose the use of a secure name system as basis for a directory service. This service is open to all users within the confines of name registration rules imposed by the underlying name system. Users are further enabled to switch between or dispose of any of their digital identities. In order to maximize the openness of the system, our re:claimID reference implementation is built on top of the GNS, which does not impose constraints on name registration. We put authorization decisions and enforcement thereof on identity data completely in the user's hands through the use of a cryptographic access control layer built using ABE.

Consequently, we maximize the ability of users to exercise their right to informational self-determination by facilitating self-management of digital identities and personal data without the requirement of any third party services. This separates the digital identities of users from commercial interests such as targeted advertisement and reduces opportunities of data abuse.

Research Question 2: How can we mitigate the liability concerns that arise with the management of identity data? By putting the user in charge of identity and attribute management trough a decentralized, self-sovereign identity architecture, we enable IdPs to mitigate liability threats in connection with data protection laws. IdPs no longer have to worry about providing services which allow RPs to retrieve and users to manage identity and attribute data. Instead, they can focus on their main purpose: Being authorities with respect to specific aspects of digital identities.

In use cases where third party asserted credentials are required by RPs, IdPs fill the important role of verification and attestation of user attributes. Through our contributions, such assertions can be handed over by the IdP to the user. No longer must IdPs store all user data permanently in order to provide it through an online service to RPs.

We admit that our approach is prone to destroy current business models of some IdPs which revolve around data analytics of social movements and monetization of personal user data. But, it also mitigates the liability risks IdPs are subjected to with respect to privacy laws and regulations. IdPs are enabled to follow the spirit of data reduction and data economy. This reduces risks which are commonly associated with unintended user data disclosure, for example through hackers and data leaks.

Research Question 3: How can trust in identity attributes be established in decentralized, self-sovereign identity systems? Self-sovereign identity systems, especially those that support storage and retrieval of identity data asynchronously with RPs, face a challenge: What if the RP requires external, third party verification of the identity data? We elaborated above that this is the primary purpose of IdPs. However, in traditional systems trust establishment was achieved implicitly, by having the IdP itself serve the asserted data. In self-sovereign identity system, such as the one proposed by us, this is no longer an option. While existing public key infrastructures (PKIs) can be used to complement self-sovereign identity systems, their imposed rigid trust models often still require centralized infrastructure or impose centralized trust hierarchies. We investigated mechanisms for decentralized trust establishment in order to find efficient and flexible approaches to this challenge.

We proposed the use of ABD policies which are resolvable via the same decentralized infrastructure which is used by self-sovereign identity systems. This enables IdPs to assert not only user identities, but delegate assertions and build assertion chains in order to build a trust ecosystem which can hypothetically span across all IdPs. At the same time, RPs are enabled to build access control policies using local authoritative trust domains and evaluate trust chains to their users.

In addition to ABD trust establishment, we also propose a privacy-preserving credential system compatible with decentralized directory services. While approaches like ABD allow us to decentralize trust establishment itself, we also investigated the protection of personal information through the use of privacy credentials. Privacy credentials make use of zero-knowledge protocols to enable users to provide credentials without disclosing personal information. The challenge we were faced with is that current protocols require interaction between user and RP for credential presentation. This is not practical as the RP may require the credential information at times when the user is offline. As a result we created attribute-based credentials which can be presented non-interactively retaining zero-knowledge properties.

6.2. Outlook and future work

In the process of our research, we have identified additional future work which we did not address as part of this thesis. We do not consider research in the area of self-sovereign identity systems to be exhausted. The following aspects require further investigation and respective results would also particularly benefit and complement the contributions we presented in this work:

Usage control over shared identity data: The first topic is *usage control over shared identity data*. While our contributions address questions of access control and authorization, we left aside any concept of data usage control.

When a user authorizes a RP to access personal information, it is free to do as it pleases with data including, but not limited to, disclosure, modification and storage. While we assume the RPs act in their self-interest by not storing this data in order to mitigate liability risks, we cannot ensure that they actually behave in this way.

This issue could be addressed through non-technical means such as certification and audits of RPs. Systems for continuous assurance of respective properties facilitate such approaches through technical means. However, certifications offer no tangible guarantees for the user. While current privacy laws and regulations also dictate the limits of personal data processing, technical means in order to enforce them are still in their infancy. Finding efficient technical solutions to this problem which are applicable in decentralized architectures such as re:claimID is a relevant and quite challenging area of research. One direction would be to implement protocols revolving around trusted computing and remote attestation with trust anchors rooted in computing hardware.

Identity and key recovery mechanisms: The second aspect falls into the area of *identity and key recovery mechanisms*. Given the significance of digital identities in all aspects of life today, the loss of access to them is fatal from the point of view of the user. Self-sovereign identity systems such as re:claimID or blockchain-based approaches make heavy use of asymmetric cryptography. The users only wield control over their identities as long as they are in possession of the respective associated private keys.

Since self-sovereign identity systems cannot rely on trusted cloud storages in order to implement self-service recovery mechanisms, alternative approaches must be found. One conceivable approach is to split up key material and persist it in the decentralized storages in a way which allows only the user to reassemble it. Another approach is using the social graph of the user and distributing necessary information for identity recovery on friends' devices [137]. If the user trusts a number of peers then threshold encryption schemes as another option. Threshold cryptography itself is already well-researched [32, 76, 55] and is slowly finding its way into key management for decentralized applications in order to increase resilience and usability [50]. However, usable integration in general and integration into identity systems in particular are promising areas of future research.

In any case, no optimal solution exists at the moment but is required in order to ensure uninterrupted, smooth operation of a self-sovereign identity system.

Device identity management and sensor data sharing: In Chapter 4, we already briefly discussed the applicability of re:claimID to the IoT. However, our contribution in this thesis regarding trust establishment through ABD especially in the context device enrollment is an interesting research opportunity. Existing efforts regarding device enrollment in local trust domains such as the IETF BRSKI [104] specifications do not directly address trust establishment of initial device identities. The applicability of ABD in order to build truly distributed IoT ecosystems which support the establishment of trust from device vendors to users is an open question.

Further, our approach to privacy credentials might also be applicable in the IoT context: In order to improve security in the IoT, there are two conflicting requirements. The first is that administrators want to know if devices in their networks are running software which is up to date and do not contain any known vulnerabilities. If they do, then they need to take precautions such as firmware updates or network quarantine. On the other hand, mechanisms which allow to ascertain the software versions help attackers to find applicable exploits easily. As such, device and software are often configured to not make this information publicly available.

The above requirements are more or less mutually exclusive. However, through the use of privacy credentials it might be possible to tackle this problem.

List of Figures

List of Tables

APPENDIX A

Publications in the context of this thesis

In the following we give an overview over and brief descriptions of relevant publications in the context of this thesis. The contributions are listed in chronological order.

1. Managing and Presenting User Attributes over a Decentralized Secure Name System ***Martin Schanzenbach and Christian Banse*****.** Published in: *Proceedings of the 11th International Workshop on Data Privacy Management and Security Assurance (DPM), Heraklion, Greece, 2016*

Relation to thesis In this paper, we investigated how a decentralized name system could be used to manage and present user attributes. In particular, our goal was to show how a secure, decentralized service can be realized for a web based authorization scenario. We discussed possible approaches to such a design and proposed a new web authorization protocol. We used formal methods to ensure that our proposed protocol satisfies our security goals and presesented a reference implementation. Lessons learned and research questions resulting from this work formed the basis of re:claimID which we presented in detail as part of this thesis.

2. A Brief History of Authorization in Distributed Systems: Information Storage, Data Retrieval and Trust Evaluation ***Ava Ahadipour and Martin Schanzenbach*****.** Published in: *Proceedings of the International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), Sydney, Australia, 2017*

Relation to thesis In the context of our work on decentralized identity services we achnowledged the need for trust establishment especially in the context of decentralized systems. As part of our research, we surveyed historical and recent research in this area and established a categorization scheme which allowed us to identify trust models which are compatible with decentralized use cases and those that are not. As a result of this research, we dove deeper into the concepts of ABD which we also propose as part of our trust establishment sections in this thesis.

3. Practical Decentralized Attribute-Based Delegation using Secure Name Systems *Martin Schanzenbach, Christian Banse and Julian Schütte.* Published in: *Proceedings of the International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), New York, USA, 2018*

Relation to thesis In this paper, we present our design and implementatiton of an ABD system built on top of a secure name system. Our goal was to find an approach to trust establishment which supports and extends our re:claimID design and the attributes shared therein. In this work, we propose a generalizable approach to facilitate ABD authorization and trust establishment throug the use of secure name systems as delegation storage and retrieval mechanisms. We formally define how a secure name system can be used to support such systems and what limitations need to be addressed. We present a reference implementation on top of the GNS and discuss its applicability in areal world use case in the context of anti-doping. While we did not give our design or implementation a specific name, this method of trust establishment is a core contribution in the context of this thesis and applicable in the context of re:claimID.

4. reclaimID: Secure, Self-Sovereign Identities using Name Systems and Attribute-Based Encryption *Martin Schanzenbach, Georg Bramm and Julian Schütte.* Published in: *Proceedings of the International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), New York, USA, 2018*

Relation to thesis This publication contains the core design and evaluation of re:claimID. Our research builds upon our original deliberations regarding decentralized identity services elaborated above. In this work, we address our discovered research questions and various shortcomings of the original approach. In the paper, we present an abstract design for a decentralized, self-sovereign identity service which is realized on top of a decentralized directory service. We apply the idea of a name system as decentralized directory service and enhance existing approaches, including our own initial aproach, in order to mitigate a variety of shortcomings.

For one, the design aims to be name system agnostic and we present a concrete implementation in top of the GNS. Further, we present how ABE can used as a cryptographic access control layer on open, decentralized directory services in order to protect the users' privacy and personal data. Along with our research in ABD systems, re:claimID constitutes another core contribution whithin this thesis and is presented as such in Chapter 4.

5. ZKlaims: ZKlaims: Privacy-preserving Attribute-based Credentials using Non-interactive Zero-knowledge Techniques *Martin Schanzenbach, Thomas Kilian, Christian Banse and Julian Schütte.* Published in: *Proceedings of the 16th International Conference on Security and Cryptography (SECRYPT 2019)*

Relation to thesis In this puplication we outline our design for ZKlaims, the non-interactive zero-knowledge (NIZK) credential system we discuss in Chapter 5. We proposed the use of Zero-knowledge Succinct Non-interactive Arguments of Knowledge (zkSNARKs) in order to realize privacy credentials which can be presented non-interactively. Further, we present key performance indicators of this credential system such as credential size and proving as well as verification speed. Our results in combination with the non-interactive property of the system allow us to integrate it into our proposed decentralized self-sovereign directory service re:claimID. Finally, we outline an integration paths into re:claimID in the paper, but the system is in theory applicable to any decentralized directory service.

APPENDIX B

re:claimID prototype

B.1. Wire formats

	0	8	16	24
0	32-Bit Integer (Attribute type)			
1	32-Bit Integer (Attribute version)			
2	32-Bit Integer (Name length$n$)			
3	32-Bit Integer (Value length)			
4	DATA (Name)			
...				
4+n-1				
4+n	DATA (Value)			
...				

Figure B.1.: Attribute record (Type ID_ATTR)

	0	8	16	24
0	DATA (Serialized ABE master key)			
...				

Figure B.2.: ABE master key record (type ABE_MASTER_KEY)

	0	8	16	24
0	0-terminated string (Attribute list)			
...				
n-1				
n	DATA (Serialized ABE key)			
...				

Figure B.3.: ABE decryption key record (Type ABE_KEY)

B.2. Key-Policy ABE Variant

An implementation using KP-ABE would differ just slightly from the CP-ABE version. We present drop-in replacements for the cryptographic functions as well as the add and authorize procedures in the following:

$$\begin{aligned}\textbf{setup}_{\mathsf{ABE}}() &\to (msk, pk)\\ \textbf{keygen}_{\mathsf{ABE}}(msk, \mathcal{P}) &\to sk_{\mathcal{P}}\\ \textbf{enc}_{\mathsf{ABE}}(pk, \mathcal{M}, \mathcal{A}) &\to \mathcal{C}_{\mathcal{A}}\\ \textbf{dec}_{\mathsf{ABE}}(sk_{\mathcal{P}}, \mathcal{C}_{\mathcal{A}}) &\to \mathcal{M}\end{aligned}$$

```
Procedure addKPABE
  input : User attribute $a = (key, value, version)$
          User identity $\mathcal{U} = (\mathcal{I}_{\mathcal{U}}, \mathcal{N}_{\mathcal{U}})$
1 $\mathcal{A} \leftarrow key \oplus version$;
2 $\mathcal{C}_{\mathcal{A}} \leftarrow \texttt{enc}_{\mathsf{ABE}}(pk \in \mathcal{I}_{\mathcal{U}},\ \mathcal{M} = value,\ \mathcal{A})$;
3 $\mathcal{R} \leftarrow (type =' ID\_ATTR', \mathcal{C}_{\mathcal{A}}, ttl = 1h)$;
4 $\texttt{publish}(\mathcal{N}_{\mathcal{I}},\ key,\ \mathcal{R})$;
```

```
Procedure authorizeKPABE
input : User identity $\mathcal{U} = (\mathcal{I}_{\mathcal{U}}, \mathcal{N}_{\mathcal{U}})$
        requesting party $\mathcal{RP} = (\mathcal{I}_{\mathcal{RP}}, \mathcal{N}_{\mathcal{RP}})$
        Set of attributes $\mathcal{A}$
output: A ticket $\mathcal{T}$
1 $\mathcal{P} \leftarrow \bigvee_{(key,value,version)\in\mathcal{A}} key \oplus version$;
2 $sk_{\mathcal{P}} \leftarrow \texttt{keygen}(msk \in \mathcal{I}_{\mathcal{U}},\ \mathcal{P})$;
3 $\mathcal{C} \leftarrow \texttt{enc}(e \in \mathcal{I}_{\mathcal{RP}},\ sk_{\mathcal{P}})$;
4 $n \leftarrow_R \mathbb{R}$;
5 $\mathcal{R} \leftarrow (type =' ABE\_KEY', \mathcal{C}, ttl =' 1h')$;
6 $\texttt{publish}(\mathcal{N}_{\mathcal{U}},\ n,\ \mathcal{R})$;
7 return $\mathcal{T} \leftarrow (\mathcal{U}, \mathcal{RP}, \mathcal{A}, n)$;
```

B.3. Usability studies

The following instructions were used in the usability studies conducted in the DASEIN project [57].

B.3.1. Web study

Results

Table B.1.: SUS results from web study.

Participant	q1	q2	q3	q4	q5	q6	q7	q8	q9	q10	SUS Score
p1	5	1	5	1	2	1	5	1	5	4	85.0
p2	5	2	4	1	3	2	5	1	3	1	82.5
p3	5	1	5	1	5	1	5	1	5	1	100.0
p4	5	1	4	1	5	2	5	1	5	1	95.0
p5	Not finished										
p6	3	2	5	1	3	1	4	2	5	1	82.5
p7	5	2	5	1	5	3	4	1	4	1	87.5
p8	5	1	5	1	5	1	4	1	5	1	97.5
p9	5	1	5	1	5	1	5	1	5	1	100.0
p10	3	2	4	1	4	3	4	2	4	2	72.5
p11	Not finished										
p12	3	4	2	1	4	3	3	3	2	2	52.5
p13	4	1	5	1	5	2	5	1	5	1	95.0
p14	4	2	4	2	4	2	4	2	4	1	77.5
p15	4	1	5	1	4	1	5	1	5	1	95.0
p16	4	1	5	1	4	1	5	1	4	2	90.0
p17	5	1	5	1	5	1	5	2	4	1	95.0
p18	4	1	5	1	4	1	4	1	4	1	90.0
p19	4	1	5	1	5	1	5	1	3	1	92.5
p20	5	1	5	1	5	1	5	1	5	1	100.0
p21	5	1	5	1	4	1	4	1	5	1	95.0
p22	4	2	5	1	4	2	4	1	3	1	82.5
p23	2	1	5	1	5	1	5	3	4	1	85.0
p24	Not finished										
p25	4	2	4	1	3	2	5	2	4	1	80.0
p26	4	1	5	1	4	1	5	1	5	1	95.0
p27	5	1	5	1	5	1	5	1	5	1	100.0
p28	5	1	5	1	5	1	5	1	5	1	100.0
p29	4	1	5	1	3	2	5	2	4	1	85.0
p30	5	1	4	1	5	1	4	1	5	1	95.0

Table B.1.: SUS results from web study (continued).

p31	2	3	3	3	4	2	3	3	2	3	50.0
p32	4	2	4	1	3	1	4	2	4	2	77.5
p33	3	3	4	1	4	2	3	4	2	2	60.0
p34	5	1	5	1	3	1	5	1	5	1	95.0
p35	4	2	5	1	4	1	5	5	5	1	82.5
p36	4	1	5	1	5	1	5	2	4	1	92.5
p37	4	2	5	1	5	1	5	1	5	1	95.0
p38	5	3	4	1	3	2	3	3	3	2	67.5
p39	5	1	4	1	5	1	5	1	5	1	97.5
p40	3	2	4	2	4	1	5	1	2	1	77.5
p41	5	1	5	1	5	1	4	1	5	1	97.5

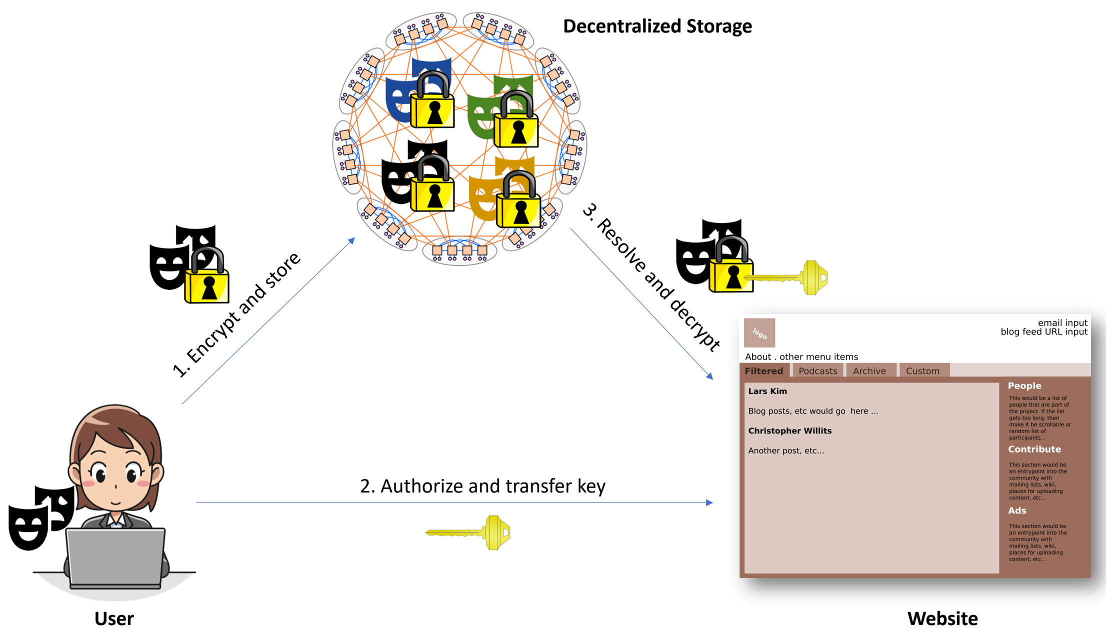
Instructions

User Authentication Study

Welcome to our User Authentication Study!

In this study you will use a new innovative decentralized authentication service "re:claimID". 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:claimID 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:claimID" 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. **A**dd a name for “full_name”, e.g. “John Doe”.
8. **C**lick on “Save”.
9. You may now create additional pseudonyms or attributes.
10. Finally, **c**lick “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!

B.3.2. IoT study

B.3.3. Results

Table B.2.: SUS results from IoT study.

Participant	q1	q2	q3	q4	q5	q6	q7	q8	q9	q10	SUS Score
p1	4	1	5	1	5	1	4	1	4	1	92.5
p2	4	3	3	1	4	1	4	3	4	3	70.0
p3	5	1	5	1	5	1	5	1	5	1	100.0
p4	4	3	4	4	5	1	5	1	2	2	72.5
p5	Not finished										
p6	4	1	4	1	4	2	4	1	4	1	85.0
p7	5	1	3	1	5	2	3	2	4	1	82.5
p8	3	3	5	2	4	1	4	2	2	3	67.5
p9	5	1	5	1	5	1	5	1	5	1	100.0
p10	3	1	2	1	3	1	3	2	2	2	65.0
p11	Not finished										
p12	Not finished										
p13	3	4	3	1	4	1	5	2	3	4	65.0
p14	4	1	5	1	5	1	5	1	3	1	92.5
p15	2	4	3	3	4	2	3	3	2	4	45.0
p16	3	1	4	1	4	2	4	1	3	1	80.0
p17	2	1	4	1	4	1	5	1	4	1	85.0
p18	4	3	2	2	3	1	3	4	2	4	50.0
p19	5	1	1	2	2	1	2	5	5	1	62.5
p20	3	1	4	2	5	1	5	1	4	1	87.5
p21	Not finished										
p22	4	2	4	1	3	1	5	2	4	1	82.5
p23	2	1	4	1	4	1	5	1	3	1	82.5
p24	Not finished										
p25	5	1	5	1	5	1	5	1	5	1	100.0
p26	5	1	5	1	4	1	5	1	5	1	97.5
p27	5	1	5	1	5	1	4	1	4	1	95.0
p28	2	1	5	3	5	1	5	1	4	1	85.0
p29	4	1	5	1	4	2	5	1	3	2	85.0
p30	4	1	2	1	1	2	3	2	2	1	62.5
p31	4	1	5	1	3	1	5	1	4	1	90.0
p32	3	2	4	1	3	2	5	4	4	1	72.5
p33	4	2	4	1	5	1	5	3	2	1	80.0
p34	Not finished										
p35	5	1	5	1	5	1	5	2	5	1	97.5

p36	4	1	4	1	5	1	5	1	5	2	92.5
p37	3	1	4	1	5	1	4	2	3	1	82.5
p38	5	2	5	1	4	2	5	1	4	1	90.0

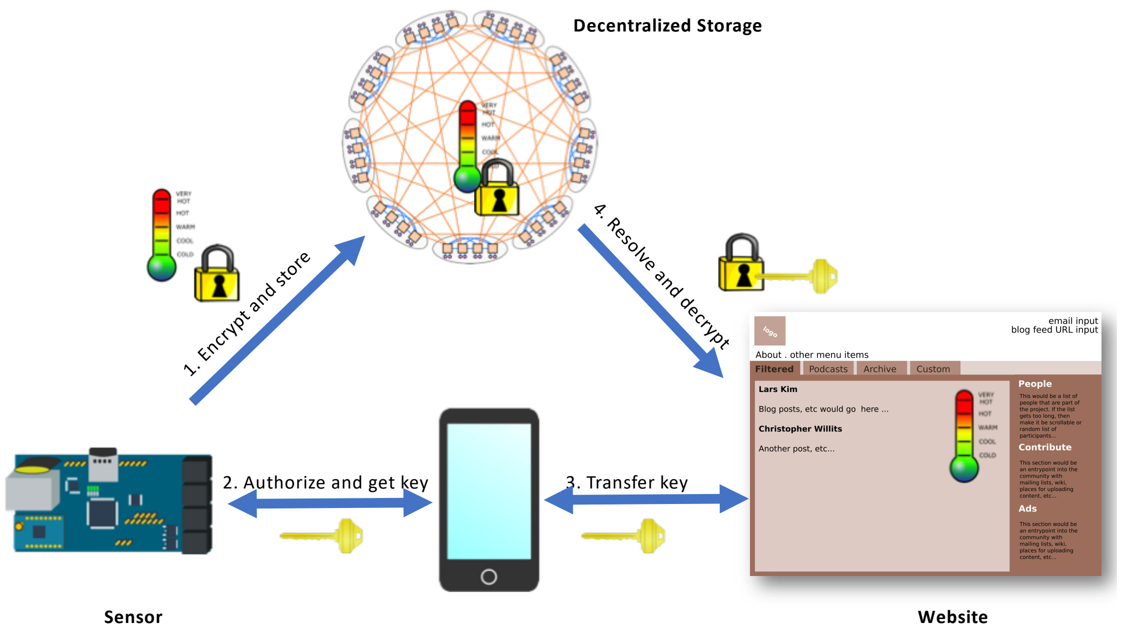
Instructions

Internet-of-Things Authentication Study

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. **O**pen the App “re:claim” on your phone.
3. **T**ap on the “Scan QR Code” to start the authorization process.
4. **S**can the QR code displayed on the webpage.
5. **T**ap on the “re:claim” icon at the right side of the “Sensors Central” item in the App.
6. **A**lign the displayed icon on the phone with the icon on the sensor AND when prompted **t**ap 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!

B.4. Security proofs

```
-- Only relevant sections included for brevity.
#Processes
INITIATOR(I, nc, CSK, CPK, G) knows
PK, GNSENC
USER(U, grant, CPK, G, data) knows
PK, SK(U), GNSENC
GNS(G) knows PK, GNSENC

#Protocol description
0. ->I: U
1.I->U: nc, I
2.U->G: {{data}{CPK}}{GNSENC(grant, PK(U))}
% record
3.U->I: {nc, grant, PK(U)}{CPK},
{{nc, grant, PK(U)}{CPK}}{SK(U)}
4.I->G: {grant}{GNSENC(grant, PK(U))}
% query
5.G->I: record %
{{data}{CPK}}{GNSENC(grant, PK(U))}

```

```
#Specification
Secret(U, grant, [I])
Secret(U, data, [I, G])
Agreement(U, I, [G, data])

#Intruder Information
Intruder = Mallory
IntruderKnowledge =
{Gns, User, Mallory, nonce,
PK, SK(Mallory), cpk, GNSENC}
```

B.5. Wire formats

	0	8	16	24
0	32-Bit Integer (Delegation set count)			
1	DATA (Delegation set)			
...				

Figure B.4.: Delegation record (type ATTR)

	0	8	16	24
0	256-Bit ECDSA Public Key (Delegation Subject)			
...				
3				
4	32-Bit Integer (Attribute count)			
5	DATA (Attribute(s))			
...				

Figure B.5.: Delegation set

	0	8	16	24
0	256-Bit ECDSA Public Key (Issuer)			
...				
3				
4	256-Bit ECDSA Public Key (Subject)			
...				
7				
8	256-Bit ECDSA Signature			
...				
11				
12	64-Bit Integer (Expiration Time)			
13				
14	32-Bit Integer (Attribute Length)			
15	DATA (Attribute)			
...				

Figure B.6.: Delegation Record (Type CRED)

Acronyms

ABC attribute-based credential. 25, 27, 37, 43, 98, 104, 112, 117–119, 126, 127

ABD attribute-based delegation. 24, 34–37, 99–103, 108–110, 112, 114–116, 126, 127, 129, 131, 133, 136, 139, 140

ABE attribute-based encryption. 32, 43, 44, 48–51, 54–60, 63, 64, 66–68, 80, 81, 83, 91, 92, 94, 97, 101, 104, 112, 129, 130, 135, 136, 140, 143

AD Microsoft Active Directory. 7

aMAC algebraic MAC. 28, 29

API application programming interface. 63, 68, 112, 113

CA certification authority. 23, 24

CLI command-line interface. 63, 68, 69, 113, 114

DAP Directory Access Protocol. 8–11, 36

DHT distributed hash table. 13, 19, 21, 23, 31, 34, 64–67, 81, 92, 96, 101, 115

DID Decentralized Identifier. 10

DIT Directory Information Tree. 8–11, 135

DLT Distributed Ledger Technology. 46, 47

DNS Domain Name System. 7, 9–13, 24, 35, 36, 42, 43, 46–48, 52, 64, 100–102, 115, 117, 135, *Glossary:* Domain Name System

DNSSEC DNS Security Extensions. 45, 101, 102

GDPR General Data Protection Regulation. 2, 3

GNS GNU Name System. 12, 13, 36, 41, 44, 45, 47, 48, 63–68, 71, 78, 80–82, 91–98, 100–102, 109, 112, 113, 115–117, 127, 129, 130, 135, 136, 140, *Glossary:* GNU Name System

GUI graphical user interface. 63, 68

IANA Internet Assigned Numbers Authority. 11

ICANN Internet Corporation for Assigned Names and Numbers. 11

IdP Identity Provider. 3–5, 14–21, 26–29, 31, 38, 42, 44, 45, 70, 71, 88–92, 98, 117, 127, 129–131, 136, *Glossary:* Identity Provider

IoT Internet-of-Things. 75, 83, 86, 129, 133, 135

IPFS Interplanetary File System. 34

ITU International Telecommunication Union. 8

LDAP Lightweight Directory Access Protocol. 7, 8

NIS Network Information Service. 7

NIZK non-interactive zero-knowledge. iii, 27, 99, 117, 118, 141

OIDC OpenID Connect 1.0. 4, 15–18, 25, 26, 28–32, 41, 53, 63, 68, 70–73, 82–85, 88–92, 94–96, 126, 127, 135, 136

PBC pairing-based cryptography. 48

PGP Pretty Good Privacy. 37

PKI public key infrastructure. iii, 9, 21, 23, 24, 34, 35, 37, 110, 111, 131

PP-ABC privacy-preserving attribute-based credential. iii, 26, 27, 117

REST representational state transfer. 18

RP Relying Party. 15–21, 24, 26–34, 38, 41, 43–50, 52, 53, 55–61, 63, 67, 69–71, 73, 75, 76, 80–86, 88–99, 117, 118, 127, 129–132, 135–137, *Glossary:* Relying Party

RT Role-based Trust Management. 38, 104

SAML Security Assertion Markup Language. 4, 15, 16, 18, 19

SDSI Simple Distributed Security Infrastructure. 24, 35–38, 64

SOAP Simple Object Access Protocol. 18

SSO Single Sign-On. 18, 27

SUS System Usability Scale. 83, 85, 87, 89, 136, *Glossary:* System Usability Scale

TLS Transport Layer Security. 24

UMA User-Managed Access. 107, 127

VC verifiable computation. 118

XML Extensible Markup Language. 18

zkSNARKs Zero-knowledge Succinct Non-interactive Arguments of Knowledge. 99, 117–119, 141

Glossary

Domain Name System The most commonly used, hierarchical name system. 7, 35, 42, 100, 135

GNU Name System A secure, decentralized alternative to the Domain Name System. 12, 36, 41, 100, 129

GNUnet An alternative network stack for building secure, decentralized and privacy-preserving distributed applications.. 113

Identity Provider A service that asserts identity information, handles authentication and often also features self-service capabilities for users. Example: OpenID Connect Identity Provider. 3, 14, 26, 42, 117, 129

Relying Party An entity that requests access to resources such as personal data. Example: A website that requires the user's email address. 26, 99

System Usability Scale A standardized, popular scoring system for usability studies of technical systems and their user interfaces.. 83, 136

X.500 A series of standards covering directory services such as (L)DAP. 4, 7–11, 23, 24, 36

X.509 The X.500 certificate format.. 4, 9, 11, 22–24, 35–37, 43, 45, 117

Bibliography

[1] A. Abdul-Rahman and S. Hailes. “A distributed trust model”. In: *Proceedings of the 1997 workshop on New security paradigms*. ACM. 1998, pp. 48–60.

[2] K. Aberer. “P-Grid: A self-organizing access structure for P2P information systems”. In: *International Conference on Cooperative Information Systems*. Springer. 2001, pp. 179–194.

[3] K. Aberer and Z. Despotovic. “Managing trust in a peer-2-peer information system”. In: *Proceedings of the tenth international conference on Information and knowledge management*. ACM. 2001, pp. 310–317.

[4] S. Agrawal and M. Chase. “FAME: fast attribute-based message encryption”. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2017, pp. 665–682.

[5] A. Ahadipour and M. Schanzenbach. “A Survey on Authorization in Distributed Systems: Information Storage, Data Retrieval and Trust Evaluation”. In: *2017 IEEE Trustcom/BigDataSE/ICESS*. Aug. 2017, pp. 1016–1023. doi: `10.1109/Trustcom/BigDataSE/ICESS.2017.346`.

[6] C. Allen, A. Brock, V. Buterin, J. Callas, D. Dorje, C. Lundkvist, P. Kravchenko, J. Nelson, D. Reed, M. Sabadello, G. Slepak, N. Thorp, and H. T. Wood. *Decentralized Public Key Infrastructure*. `https://github.com/WebOfTrustInfo/rwot1-sf/blob/master/final-documents/dpki.pdf`, accessed 2019/01/14. Dec. 2015.

[7] D. Artz and Y. Gil. “A survey of trust in computer science and the semantic web”. In: *Web Semantics: Science, Services and Agents on the World Wide Web* 5.2 (2007), pp. 58–71.

[8] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza. “SNARKs for C: Verifying program executions succinctly and in zero knowledge”. In: *Advances in Cryptology–CRYPTO 2013*. Springer, 2013, pp. 90–108.

[9] D. Berger. *Facebook: Netflix, Spotify, Amazon et al. hatten Zugriff auf persönliche Daten und private Nachrichten*. `https://heise.de/-4256069`, accessed 2018/12/20. Dec. 2018.

[10] D. J. Bernstein. *DNS database espionage*. `https://dnscurve.org/espionage2.html`, accessed 2019/02/02. Feb. 2019.

[11] V. Bertola and M. Sanz. *An Architecture for a Public Identity Infrastructure Based on DNS and OpenID Connect*. `https://datatracker.ietf.org/doc/draft-bertola-dns-openid-pidi-architecture/?include_text=1`, accessed 2019/01/18. Jan. 2019.

[12] J. Bethencourt, A. Sahai, and B. Waters. "Ciphertext-policy attribute-based encryption". In: *Security and Privacy, 2007. SP'07. IEEE Symposium on*. IEEE. 2007, pp. 321–334.

[13] J. Bethencourt, A. Sahai, and B. Waters. *cpabe – Ciphertext-Policy Attribute-Based Encryption*. `http://acsc.cs.utexas.edu/cpabe/`, accessed 2019/02/07. Feb. 2019.

[14] M. Blaze, J. Feigenbaum, and J. Lacy. "Decentralized trust management". In: *Security and Privacy, 1996. Proceedings., 1996 IEEE Symposium on*. IEEE. 1996, pp. 164–173.

[15] M. Blaze, J. Feigenbaum, and M. Strauss. "Compliance checking in the policymaker trust management system". In: *International Conference on Financial Cryptography*. Springer. 1998, pp. 254–274.

[16] M. Blaze and A. D. Keromytis. "The KeyNote trust-management system version 2". In: (1999).

[17] J. Borgh. *Attribute-Based Encryption in Systems with Resource Constrained Devices in an Information Centric Networking Context*. 2016.

[18] J. Borgh, E. Ngai, B. Ohlman, and A. M. Malik. "Employing attribute-based encryption in systems with resource constrained devices in an information-centric networking context". In: *Global Internet of Things Summit (GIoTS), 2017*. IEEE. 2017, pp. 1–6.

[19] S. Bortzmeyer. *DNS Query Name Minimisation to Improve Privacy*. RFC 7816. RFC Editor, Mar. 2016.

[20] D. Brown. "Standards for efficient cryptography, SEC 1: elliptic curve cryptography". In: *Released Standard Version* 1 (2009).

[21] O. Bünte. *Verimi: Samsung tritt europäischer Login-Allianz bei*. `https://heise.de/-4277556`, accessed 2019/01/18. Jan. 2019.

[22] J. Camenisch and E. Van Herreweghen. "Design and implementation of the idemix anonymous credential system". In: *Proceedings of the 9th ACM conference on Computer and communications security*. ACM. 2002, pp. 21–30.

[23] B. Carpenter, F. Baker, and M. Roberts. *Memorandum of Understanding Concerning the Technical Work of the Internet Assigned Numbers Authority*. RFC 2860. `http://www.rfc-editor.org/rfc/rfc2860.txt`. RFC Editor, June 2000.

[24] Christian Grothoff, Matthias Wachs, Monika Ermert, and Jacob Appelbaum. "Towards Secure Name Resolution on the Internet". In: *NDSS 2017 DNS Privacy Workshop DPRIV17 '17, San Diego, CA, USA, Febuary 26, 2017*. 2017, p. 20.

[25] D. Clarke, J.-E. Elien, C. Ellison, M. Fredette, A. Morcos, and R. L. Rivest. "Certificate chain discovery in SPKI/SDSI". In: *Journal of Computer security* 9.4 (2001), pp. 285–322.

[26] E. Comission. *Online security - Seamless personal authentication (authentication for all)*. `https://ec.europa.eu/info/funding-tenders/opportunities/portal/screen/opportunities/topic-details/onlinesecurityprize-01-2017`, accessed 2019/02/02. Jan. 2017.

[27] N. Confessore. *Cambridge Analytica and Facebook: The Scandal and the Fallout So Far, The New York Times*. `https://www.nytimes.com/2018/04/04/us/politics/cambridge-analytica-scandal-fallout.html`, accessed 2018/12/20. Apr. 2018.

[28] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC 5280. `http://www.rfc-editor.org/rfc/rfc5280.txt`. RFC Editor, May 2008.

[29] J. D'Onfro. *The drag queen who took on Facebook isn't celebrating her victory yet*. `http://www.businessinsider.de/facebook-changes-to-real-name-policy-2015-12`, accessed 2019/02/20. Dec. 2015.

[30] R. Deibert, J. Palfrey, R. Rohozinski, and J. Zittrain. *Access contested: security, identity, and resistance in Asian cyberspace*. MIT Press, 2011.

[31] V. Descombes. *Puzzling Identities*. Harvard University Press, 2016.

[32] Y. G. Desmedt. "Threshold cryptography". In: *European Transactions on Telecommunications* 5.4 (1994), pp. 449–458.

[33] A. Dey and S. Weis. "PseudoID: Enhancing privacy in federated login". In: *Hot topics in privacy enhancing technologies* (2010), pp. 95–107.

[34] S. Dickinson, D. Gillmor, and T. Reddy. *Usage Profiles for DNS over TLS and DNS over DTLS*. RFC 8310. RFC Editor, Mar. 2018.

[35] N. Doering. *Sozialpsychologie des Internet: Die Bedeutung des Internet für Kommunikationsprozesse, Identitäten, soziale Beziehungen und Gruppen*. Hogrefe, 2003.

[36] P. Dunphy and F. A. Petitcolas. "A first look at identity management schemes on the blockchain". In: *arXiv preprint arXiv:1801.03294* (2018).

[37] Eduroam. `https://www.eduroam.org/`, accessed 2020/07/12. July 2020.

[38] C. Ellison et al. "Establishing identity without certification authorities". In: *USENIX Security Symposium*. 1996, pp. 67–76.

[39] D. Eppstein, M. T. Goodrich, F. Uyeda, and G. Varghese. "What's the difference? Efficient set reconciliation without prior context". In: *ACM SIGCOMM Computer Communication Review* 41.4 (2011), pp. 218–229.

[40] Ethereum. *Ethereum*. `https://www.ethereum.org/`, accessed 2017/11/07. Nov. 2017.

[41] Y. Eudes, C. Grothoff, J. Appelbaum, M. Ermert, L. Poitras, and M. Wachs. "Morecowbell-nouvelles révélations sur les pratiques de la nsa". In: *Le Monde* 24 (2015).

[42] D. S. Evans. "The online advertising industry: Economics, evolution, and privacy". In: *Journal of Economic Perspectives* 23.3 (2009), pp. 37–60.

[43] N. S. Evans and C. Grothoff. "R5N: Randomized recursive routing for restricted-route networks." In: *NSS*. 2011, pp. 316–321.

[44] D. Fett, R. Küsters, and G. Schmitz. "Spresso: A secure, privacy-respecting single sign-on system for the web". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2015, pp. 1358–1369.

[45] A. Fiat and M. Naor. "Broadcast encryption". In: *Annual International Cryptology Conference*. Springer. 1993, pp. 480–491.

[46] S. Foundation. *Sovrin: A Protocol and Token for Self-Sovereign Identity and Decentralized Trust*. `https://sovrin.org/wp-content/uploads/2018/03/Sovrin-Protocol-and-Token-White-Paper.pdf`, accessed 2019/02/02. Jan. 2018.

[47] S. Friebe, I. Sobik, and M. Zitterbart. "DecentID: Decentralized and Privacy-Preserving Identity Storage System Using Smart Contracts". 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)*. Aug. 2018, pp. 37–42. doi: `10.1109/TrustCom/BigDataSE.2018.00016`.

[48] S. D. Galbraith, K. G. Paterson, and N. P. Smart. "Pairings for cryptographers". In: *Discrete Applied Mathematics* 156.16 (2008), pp. 3113–3121.

[49] B. Gellman and L. Poitras. "US, British intelligence mining data from nine US Internet companies in broad secret program". In: *The Washington Post* 6 (2013).

[50] R. Gennaro, S. Goldfeder, and A. Narayanan. "Threshold-optimal DSA/ECDSA signatures and an application to Bitcoin wallet security". In: *International Conference on Applied Cryptography and Network Security*. Springer. 2016, pp. 156–174.

[51] I. Giechaskiel, C. Cremers, and K. B. Rasmussen. "On Bitcoin Security in the Presence of Broken Cryptographic Primitives". In: *Computer Security – ESORICS 2016: 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26-30, 2016, Proceedings, Part II*. Ed. by I. Askoxylakis, S. Ioannidis, S. Katsikas, and C. Meadows. Cham: Springer International Publishing, 2016, pp. 201–222. isbn: 978-3-319-45741-3. doi: `10.1007/978-3-319-45741-3_11`.

[52] V. Goel and N. Perlroth. *Yahoo Says 1 Billion User Accounts Were Hacked*. https://www.nytimes.com/2016/12/14/technology/yahoo-hack.html, accessed 2019/02/20. Dec. 2016.

[53] P. Gola, R. Schomerus, and C. Klug. *BDSG - Bundesdatenschutzgesetz : Kommentar*. 8. überarbeitete und ergänzte Auflage. München: Beck, 2005. isbn: 978-3-406-52152-2.

[54] S. Goldberg, M. Naor, D. Papadopoulos, L. Reyzin, S. Vasant, and A. Ziv. "NSEC5: Provably Preventing DNSSEC Zone Enumeration." In: *NDSS*. 2015.

[55] M. Green and T. Eisenbarth. "Strength in Numbers: Threshold ECDSA to Protect Keys in the Cloud." In: *IACR Cryptology ePrint Archive* 2015 (2015), p. 1169.

[56] J. Groth. "On the size of pairing-based non-interactive arguments". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2016, pp. 305–326.

[57] C. Grothoff, M. Schanzenbach, A. Laube, E. Benoist, and P. Mainini. "Decentralized Authentication for Self-Sovereign Identities using Name Systems". In: 847382 (Oct. 2018).

[58] C. Grothoff, M. Wachs, M. Ermert, and J. Appelbaum. "Towards Secure Name Resolution on the Internet". In: *Computers & Security* (2018).

[59] H. W. P. W. Group. *An Introduction to Hyperledger*. https://www.hyperledger.org, accessed 2019/02/02. Aug. 2018.

[60] R. Hamirani. *The Landscape of Customer Identity: Facebook Dominates, Payment Providers on the Rise*. http://www.gigya.com/blog/the-landscape-of-customer-identity-q2-2015/, accessed 2016/02/20. July 2015.

[61] D. Hardt. *The OAuth 2.0 Authorization Framework*. RFC 6749. http://www.rfc-editor.org/rfc/rfc6749.txt. RFC Editor, Oct. 2012.

[62] A. Herzberg and H. Shulman. "Fragmentation Considered Poisonous, or: One-domain-to-rule-them-all.org". In: *2013 IEEE Conference on Communications and Network Security (CNS)*. Oct. 2013, pp. 224–232. doi: 10.1109/CNS.2013.6682711.

[63] P. Hoffman and J. Schlyter. *The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA*. RFC 6698. http://www.rfc-editor.org/rfc/rfc6698.txt. RFC Editor, Aug. 2012.

[64] R. Holz, L. Braun, N. Kammenhuber, and G. Carle. "The SSL landscape: a thorough analysis of the x. 509 PKI using active and passive measurements". In: *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*. ACM. 2011, pp. 427–444.

[65] Z. Hu, L. Zhu, J. Heidemann, A. Mankin, D. Wessels, and P. Hoffman. *Specification for DNS over Transport Layer Security (TLS)*. RFC 7858. RFC Editor, May 2016.

[66] L. Hui and M. Rajagopalan. "At Sina Weibo's censorship hub, China's Little Brothers cleanse online chatter". In: *Reuters, September* 11 (2013).

[67] IETF. *Public-Key Infrastructure (X.509)*. `https://datatracker.ietf.org/wg/pkix/charter/`, accessed 2019/01/14. Jan. 2019.

[68] T. Intercept. *The NSA and GCHQ's QUANTUMTHEORY Hacking Tactics*. `https://theintercept.com/document/2014/03/12/nsa-gchqs-quantumtheory-hacking-tactics/`, accessed 2017/11/07. Mar. 2014.

[69] IPFS. *IPFS*. `https://ipfs.io/`, accessed 2017/11/12. Nov. 2017.

[70] M. Isaakidis, H. Halpin, and G. Danezis. "UnlimitID: Privacy-preserving federated identity management using algebraic MACs". In: *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society*. ACM. 2016, pp. 139–142.

[71] M. Jones and D. Hardt. *The OAuth 2.0 Authorization Framework: Bearer Token Usage*. RFC 6750. `http://www.rfc-editor.org/rfc/rfc6750.txt`. RFC Editor, Oct. 2012.

[72] S. Josefsson. *Storing Certificates in the Domain Name System (DNS)*. RFC 4398. RFC Editor, Mar. 2006.

[73] S. Kamvar, M. Schlosser, and H. Garcia-Molina. "EignRep: Reputation management in P2P networks". In: *Proceedings of the World-Wide Web Conference*. 2003.

[74] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. "The eigentrust algorithm for reputation management in p2p networks". In: *Proceedings of the 12th international conference on World Wide Web*. ACM. 2003, pp. 640–651.

[75] M. Karppinen. *Data is not an asset, it's a liability*. `https://www.richie.fi/blog/data-is-a-liability.html`. Sept. 2015.

[76] J. Katz and M. Yung. "Threshold cryptosystems based on factoring". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2002, pp. 192–205.

[77] N. Koblitz and A. Menezes. "Pairing-based cryptography at high security levels". In: *IMA International Conference on Cryptography and Coding*. Springer. 2005, pp. 13–36.

[78] N. Kogan, Y. Shavitt, and A. Wool. "A practical revocation scheme for broadcast encryption using smartcards". In: *ACM Transactions on Information and System Security (TISSEC)* 9.3 (2006), pp. 325–351.

[79] D. Kraft. *NameID*. `https://nameid.org/`. 2017.

[80] B. Kulynych, M. Isaakidis, C. Troncoso, and G. Danezis. "ClaimChain: decentralized public key infrastructure". In: *arXiv preprint arXiv:1707.06279* (2017).

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

[82] A. Lenhart. "Adults and social network Web sites. Pew Internet and American life Project". In: *The Pew Center, Washington DC* (2009).

[83] A. Lewko, A. Sahai, and B. Waters. “Revocation systems with very small private keys”. In: *2010 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2010, pp. 273–285.

[84] N. Li and J. C. Mitchell. “RT: A role-based trust-management framework”. In: *DARPA Information Survivability Conference and Exposition (DISCEX)*. 2003, pp. 123–139.

[85] N. Li and J. C. Mitchell. “RT: A role-based trust-management framework”. In: *DARPA Information Survivability Conference and Exposition, 2003. Proceedings*. Vol. 1. IEEE. 2003, pp. 201–212.

[86] N. Li, W. H. Winsborough, and J. C. Mitchell. “Distributed credential chain discovery in trust management”. In: *Journal of Computer Security* 11.1 (2003), pp. 35–86.

[87] C. N. de l’Informatique et des Libertes (French data protection authority). *Decision no. 2016-007 of January 26, 2016 issuing formal notice to FACEBOOK INC. and FACEBOOK IRELAND*. `http://www.cnil.fr/fileadmin/documents/en/D2016-007_MED_FACEBOOK-INC.-FACEBOOK-IRELAND-EN.pdf`. Jan. 2016.

[88] G. Lowe. “Casper: A compiler for the analysis of security protocols”. In: *Journal of computer security* 6.1, 2 (1998), pp. 53–84.

[89] A. Maria, Z. Aviv, and V. Laurent. “Hijacking Bitcoin: Routing Attacks on Cryptocurrencies”. In: *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE. 2017.

[90] K. Y. McKenna and J. A. Bargh. “Plan 9 from cyberspace: The implications of the Internet for personality and social psychology”. In: *Personality and social psychology review* 4.1 (2000), pp. 57–75.

[91] J. McLaughlin. *Federal court lifts National Security Letter gag order; First time in 14 years*. `https://theintercept.com/2015/09/14/federal-court-fully-lifts-fbi-gag-order-first-time-14-years/`, accessed 2019/02/02. Sept. 2015.

[92] *Measuring Usability*. `https://measuringu.com/sus/`. Aug. 2018.

[93] M. S. Melara, A. Blankstein, J. Bonneau, E. W. Felten, and M. J. Freedman. “CONIKS: Bringing Key Transparency to End Users.” In: *USENIX Security Symposium*. 2015, pp. 383–398.

[94] Namecoin. *Namecoin is a decentralized open source information registration and transfer system based on the Bitcoin cryptocurrency*. `https://namecoin.info/`, accessed 2016/02/23. Feb. 2016.

[95] M. Naor and B. Pinkas. “Efficient trace and revoke schemes”. In: *International Conference on Financial Cryptography*. Springer. 2000, pp. 1–20.

[96] A. Narayanan, V. Toubiana, S. Barocas, H. Nissenbaum, and D. Boneh. “A critical look at decentralized personal data architectures”. In: *arXiv preprint arXiv:1202.4503* (2012).

[97] OASIS. *Identity Provider Discovery Service Protocol and Profile*. https://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-idp-discovery.pdf, accessed 2019/01/14. Mar. 2008.

[98] R. Ostrovsky, A. Sahai, and B. Waters. "Attribute-based encryption with non-monotonic access structures". In: *Proceedings of the 14th ACM conference on Computer and communications security*. ACM. 2007, pp. 195–203.

[99] C. Paquin. "U-prove technology overview v1. 1". In: *Microsoft Corporation Draft Revision* 1 (2011).

[100] C. Paquin and G. Zaverucha. *U-prove cryptographic specification v1. 1*. Tech. rep. revision 3. Technical report, Microsoft Corporation, 2013.

[101] T. E. Parliament and the council of the European Union. http://data.europa.eu/eli/reg/2016/679/oj. May 2016.

[102] B. Parno, J. Howell, C. Gentry, and M. Raykova. "Pinocchio: Nearly practical verifiable computation". In: *Security and Privacy (SP), 2013 IEEE Symposium on*. IEEE. 2013, pp. 238–252.

[103] P. Pearce, B. Jones, F. Li, R. Ensafi, N. Feamster, N. Weaver, and V. Paxson. "Global measurement of dns manipulation". In: *USENIX Security Symposium. USENIX*. 2017, p. 22.

[104] M. Pritikin, M. Richardson, M. Behringer, S. Bjarnason, and K. Watsen. *Bootstrapping Remote Secure Key Infrastructures (BRSKI)*. Internet-Draft draft-ietf-anima-bootstrapping-keyinfra-19. http://www.ietf.org/internet-drafts/draft-ietf-anima-bootstrapping-keyinfra-19.txt. IETF Secretariat, Mar. 2019.

[105] D. Reed, M. Sporny, D. Longley, C. Allen, R. Grant, and M. Sabadello. *Decentralized Identifiers (DIDs) v0.11*. https://w3c-ccg.github.io/did-spec. 2018.

[106] M. Research. *U-Prove*. https://www.microsoft.com/en-us/research/project/u-prove/, accessed 2019/02/07. Feb. 2019.

[107] R. L. Rivest and B. Lampson. "SDSI-a simple distributed security infrastructure". In: Crypto. 1996.

[108] R. L. Rivest and B. Lampson. "SDSI-a simple distributed security infrastructure". In: Crypto. 1996.

[109] A. Sabouri. "On the user acceptance of privacy-preserving attribute-based credentials – a qualitative study". In: *Data Privacy Management and Security Assurance*. Springer, 2016, pp. 130–145.

[110] N. Sakimura, J. Bradley, and M. B. Jones. *OpenID Connect Dynamic Client Registration 1.0 incorporating errata set 1*. https://openid.net/specs/openid-connect-registration-1_0.html, accessed 2020/02/15. Nov. 2014.

[111] N. Sakimura, J. Bradley, M. B. Jones, and E. Jay. *OpenID Connect Discovery 1.0 incorporating errata set 1*. `https://openid.net/specs/openid-connect-discovery-1_0.html`, accessed 2019/01/21. Jan. 2014.

[112] N. Sakimura, J. Bradley, M. Jones, B. de Medeiros, and C. Mortimore. *OpenID Connect Core 1.0 incorporating errata set 1*. `http://openid.net/specs/openid-connect-core-1_0.html`. 2014.

[113] 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)*. Aug. 2018, pp. 244–251. doi: `10.1109/TrustCom/BigDataSE.2018.00046`.

[114] 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)*. Aug. 2018, pp. 946–957. doi: `10.1109/TrustCom/BigDataSE.2018.00134`.

[115] M. Schanzenbach. *ABD API in GNUnet*. `https://git.gnunet.org/gnunet.git/tree/src/include/gnunet_abd_service.h`, accessed 2020/04/04. Apr. 2020.

[116] M. Schanzenbach. *ABD implementation in GNUnet*. `https://gnunet.org/git/gnunet.git/tree/src/credential`, accessed 2019/02/07. Feb. 2019.

[117] M. Schanzenbach. *Example implementation of a relying party for ABD*. `https://github.com/schanzen/gnuidentity-example-rp/tree/credential`, accessed 2019/02/07. Feb. 2019.

[118] M. Schanzenbach. *Example relying party for re:claimID predecessor*. `https://github.com/schanzen/gnuidentity-example-rp`, accessed 2019/02/07. Feb. 2019.

[119] M. Schanzenbach. *Fork of the CP ABE library libbswabe replacing openssl with libgcrypt and fixing some bugs*. `https://github.com/schanzen/libgabe`, accessed 2019/02/07. Feb. 2019.

[120] M. Schanzenbach. *re:claimID API in GNUnet*. `https://git.gnunet.org/gnunet.git/tree/src/include/gnunet_reclaim_service.h`, accessed 2020/04/04. Apr. 2020.

[121] M. Schanzenbach. *User interface for ABD*. `https://github.com/schanzen/gnunet-webui/tree/credentials`, accessed 2019/02/07. Feb. 2019.

[122] M. Schanzenbach. *User interface for re:claimID predecessor*. `https://github.com/schanzen/gnunet-webui`, accessed 2019/02/07. Feb. 2019.

[123] M. Schanzenbach and C. Banse. “Managing and Presenting User Attributes over a Decentralized Secure Name System”. In: *Data Privacy Management and Security Assurance - 11th International Workshop, DPM 2016 and 5th International Workshop, QASA 2016, Heraklion, Crete, Greece, September 26-27, 2016, Proceedings*. 2016, pp. 213–220. doi: `10.1007/978-3-319-47072-6_14`.

[124] M. Schanzenbach, T. Kilian, J. Schütte, and C. Banse. “ZKlaims: Privacy-preserving Attribute-based Credentials using Non-interactive Zero-knowledge Techniques”. In: *proceedings of the 16th International Conference on Security and Cryptography (SECRYPT 2019), part of ICETE*. 2019.

[125] M. Schanzenbach and S. Zickau. “Identity and access management in a doping control use case”. In: *Datenschutz und Datensicherheit* 41.12 (2017), pp. 724–728. doi: `10.1007/s11623-017-0867-z`.

[126] J. Schmidt. “Weblogs: eine kommunikationssoziologische Studie”. In: (2006).

[127] C. Selvaraj and S. Anand. “A survey on security issues of reputation management systems for peer-to-peer networks”. In: *Computer Science Review* 6.4 (2012), pp. 145–160.

[128] J. Sermersheim. *Lightweight Directory Access Protocol (LDAP): The Protocol*. RFC 4511. `http://www.rfc-editor.org/rfc/rfc4511.txt`. RFC Editor, June 2006.

[129] M. Stiegler. *An Introduction to Petname Systems*. `http://www.skyhunter.com/marcs/petnames/IntroPetNames.html`. June 2010.

[130] J. Sullivan. *Digital Restrictions Management and Treacherous Computing*. `https://www.fsf.org/campaigns/drm.html`, accessed 2020/07/12. July 2020.

[131] S.-T. Sun and K. Beznosov. “The devil is in the (implementation) details: an empirical analysis of OAuth SSO systems”. In: *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM. 2012, pp. 378–390.

[132] A. Swartz. *Squaring the Triangle: Secure, Decentralized, Human-Readable Names*. `http://www.aaronsw.com/weblog/squarezooko`. Accessed: 2017-10-30. Jan. 2011.

[133] T. Systems. *Anastasis*. `https://docs.taler.net/anastasis.html`, accessed 2020/04/04. Apr. 2020.

[134] T. Systems. *Backup and Synchronization Service API*. `https://docs.taler.net/core/api-sync.html`, accessed 2020/04/04. Apr. 2020.

[135] P. Szoldra. *The dark web marketplace where you can buy 200 million Yahoo accounts is under cyberattack*. `https://www.businessinsider.de/real-deal-market-ddos-2016-9`, accessed 2019/02/20. Sept. 2016.

[136] K. Szymielewicz. *Your digital identity has three layers, and you can only protect one of them*. https://qz.com/1525661/your-digital-identity-has-three-layers-and-you-can-only-protect-one-of-them/, accessed 2019/02/02. Feb. 2019.

[137] uPort. *uPort Whitepaper*. https://whitepaper.uport.me/uPort_whitepaper_DRAFT20170221.pdf, accessed 2017/11/12. Nov. 2017.

[138] uPort. *uPort Whitepaper*. http://blockchainlab.com/pdf/uPort_whitepaper_DRAFT20161020.pdf, accessed 2017/11/12. Nov. 2017.

[139] B. Van Delft and M. Oostdijk. "A security analysis of OpenID". In: *IFIP Working Conference on Policies and Research in Identity Management*. Springer. 2010, pp. 73–84.

[140] J. Vanian. *Why Data Is The New Oil*. http://fortune.com/2016/07/11/data-oil-brainstorm-tech/, accessed 2019/02/02. July 2016.

[141] Various. *The re:claimID project on gitlab*. https://gitlab.com/reclaimid, accessed 2019/02/07. Feb. 2019.

[142] M. Wachs, M. Schanzenbach, and C. Grothoff. "A censorship-resistant, privacy-enhancing and fully decentralized name system". In: *Cryptology and Network Security*. Springer, 2014, pp. 127–142.

[143] M. Wachs, M. Schanzenbach, and C. Grothoff. "On the feasibility of a censorship resistant decentralized name system". In: *Foundations and Practice of Security*. Springer, 2014, pp. 19–30.

[144] W. Wijngaards and G. Wiley. *Confidential DNS*. Internet-Draft draft-wijngaards-dnsop-confidentialdns-03. http://www.ietf.org/internet-drafts/draft-wijngaards-dnsop-confidentialdns-03.txt. IETF Secretariat, Mar. 2015.

[145] B. Womack, J. Robertson, and M. Riley. *Yahoo Says at Least 500 Million Accounts Breached in Attack*. https://www.bloomberg.com/news/articles/2016-09-22/yahoo-says-at-least-500-million-accounts-breached-in-hack-attack, accessed 2019/02/20. Sept. 2016.

[146] P. Wouters. *DNS-Based Authentication of Named Entities (DANE) Bindings for OpenPGP*. RFC 7929. RFC Editor, Aug. 2016.

[147] X. Yao, Z. Chen, and Y. Tian. "A lightweight attribute-based encryption scheme for the Internet of Things". In: *Future Generation Computer Systems* 49 (2015), pp. 104–112.

[148] K. Zeilenga. *Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map*. RFC 4510. http://www.rfc-editor.org/rfc/rfc4510.txt. RFC Editor, June 2006.