Managing and Presenting User Attributes over a Decentralized Secure Name System

Martin Schanzenbach and Christian Banse

Fraunhofer AISEC, Garching b. München, Germany {schanzen, banse}@aisec.fraunhofer.de

Abstract. Today, user attributes are managed at centralized identity providers. However, two centralized identity providers dominate digital identity and access management on the web. This is increasingly becoming a privacy problem in times of mass surveillance and data mining for targeted advertisement. Existing systems for attribute sharing or credential presentation either rely on a trusted third party service or require the presentation to be online and synchronous. In this paper we propose a concept that allows the user to manage and share his attributes asynchronously with a requesting party using a secure, decentralized name system.

1 Introduction

Identity and Access Management today revolves around the presentation of attributes or credentials to services for authentication and authorization purposes. Third party identity providers (IdP), such as Google or Facebook are often used in case a service requires an asynchronous way to access user attributes. Often, users are required to share personal data, like email addresses, to use certain services (i.e., a mailing list). Such services only need the user's data when a particular action is executed (i.e., a mail is posted to the mailing list) and it must at that time be able to asynchronously access this data. Usually, this is achieved by persisting the data in a database upon registration or retrieving it from an IdP. In the first case, the data can become stale, unless the user manages the persisted data at the service. In the second case, the user and service must trust the third party IdP to provide fresh, authentic attribute data, to be available whenever needed and not to misuse attributes for user profiling.

In practice, served attributes have no freshness guarantees, the attributes are not verifiably unchanged from what the user provided and availability is either not guaranteed (in case of free offerings) or expensive. Further, IdPs have full access and control over personal, potentially sensitive data limited only by compliance laws and regulations [4], that are often subject to change or even ignored and challenged [3]. Users have no guarantee that IdPs do not indeed analyse and market personal data from attributes and requests. As digital identities are managed by a service oligopoly of two identity providers that claim over 85% of the market¹, Big Data and targeted advertisement businesses make this a valid concern.

In this paper, we present a design and implementation of a system that addresses this issue and does not rely on a centralized IdP to serve attributes. Our solution is a *decentralized* system based on a name system. It can be used to selectively share user attributes asynchronously with other parties. The users manage their attributes and asserted credentials locally on their devices and can grant access to other parties for a limited amount of time over a chosen subset of user attributes.

2 Related Work

Existing technologies and protocols related to identity and access management such as OAuth2 [7] and OpenID-Connect (OIDC) [10] are designed as centralized services which are, in practice, operated by large corporations. Systems such as Idemix address the privacy deficiencies of OAuth2 and OIDC. Idemix is an "anonymous credential system" allowing "anonymous yet authenticated and accountable transactions between users and service providers" [2]. We emphasize here that Idemix's use case is different to ours. It supports the presentation of credentials asserted by a trusted third party to a service provider in an anonymous or pseudonymous manner without disclosing the information directly. However, asynchronous presentation of the actual data is not addressed. Instead, we propose a system that allows a user to asynchronously disclose and share personal data with a relying party.

In many ways, our system addresses User-Managed-Access (UMA) [5] use cases. UMA is a system to protect a user-controlled resource server using an OAuth2-based authorization protocol. UMA also allows asynchronous access to personal data, as it requires the user to manage the data on a dedicated central resource server. However, users are only in full control over their data and authorizations when hosting their own resource and authorization server. In contrast, we propose a completely decentralized system that allows the user to manage his data locally, and selectively share it without the need of a dedicated resource server.

Finally, NameID² is a decentralized identity management system built on the blockchain-based name system namecoin³. It allows a namecoin user to create identities in the same fashion as domain names and enables the user to authenticate using a simple public key authentication scheme. However, it has one significant drawback: The data stored in the blockchain is public information. As such, storing sensitive personal information is not viable. While our approach also uses a secure name system to store data, we do not require a global, public ledger.

¹ http://www.gigya.com/blog/the-landscape-of-customer-identity-q2-2015/, accessed 2016/02/20

² https://nameid.org, accessed 2016/02/20

³ https://namecoin.info/, accessed 2016/02/23

3 Design of a Decentralized Attribute Sharing System

Our approach extends on the concept of decentralized name systems. Specifically, we base our design on the GNU Name System (GNS) [12, 11]. In GNS, a user can manage any number of namespaces and thus identities by creating key pairs – the owner of the private key is the authority of the respective namespace. In the following, we describe how our system can be used to share identity attributes in a decentralized way. It allows asynchronous sharing and provides stronger properties on authorization, availability, and freshness than current centralized IdPs, while removing the need to trust a third party.

In our design, user attributes are managed locally by the user and only published to the name system on demand in the form of *identity tokens*. This heavily relies on query privacy and non-traversable namespaces in GNS [11]. We leverage the fact that a record name in GNS can be treated as a shared secret between two parties that want to exchange information [11]. We call this shared secret name *grant* and it is used to achieve confidentiality of identity tokens. We refer to the entity that is requesting user attributes as the *client* and the entity that holds the data as the *user*. The user can authorize a client by generating a *ticket* and handing it to the client.

Our system aims to satisfy three security properties:

- 1. The grant is a shared secret between the user and an authorized client
- 2. An issued identity token cannot be retrieved by an unauthorized party
- 3. If the client is able to bind the user's public key to a trusted identity, our proposed authorization protocol also allows to authenticate the user

3.1 User Attributes and Identity Tokens

Identity attributes are key-value pairs representing user attributes, for example an email address as "email=john@doe.com". For consistency and simplicity we use GNS as a local attribute data storage. We define the record type **ID_ATTR** for records that contain identity attributes. By default ID_ATTR records are stored as private records in GNS and are therefore not remotely resolvable. Their main purpose is to store and manage attributes that a user can eventually selectively share upon request in *identity tokens*.

Identity tokens are required because if clients access ID_ATTR records directly, revocation of access would become complex as the same attributes are also shared with other clients. An identity token is issued by the user when authorizing a client and contains the attributes requested by a client. We define a record type **ID_TOKEN** for storing identity tokens. The name of an identity token record is the *grant* which is the string representation of a random number. Clients can retrieve identity tokens by querying the respective grant in a particular GNS identity namespace. Grants must be kept confidential by the user and the client.

3.2 Tickets

A ticket is a container for a grant and allows the user to securely transfer a grant to a client identified by the public key P_{client} . The user is identified by a public key P_{user} of the GNS namespace that contains the identity token he intends to share. As there is no central entity that requires the client to authenticate, the grant contained in the ticket must be cryptographically secured in such a way that it can only be decrypted by the owner of the respective private key x_{client} . This is achieved by using static-ephemeral ECDHE [1] to establish a shared symmetric encryption key K_{ticket} derived from P_{client} and a generated ephemeral private key $K_{ECDHE,priv}$. The ticket is a triple (p, k, s) consisting of the encrypted payload p, an ephemeral ECDHE public key $k = K_{ECDHE,pub}$ and the cryptographic ECDSA signature $s = S_{ticket}$ over p and k using x_{user} . The payload p is a triple (l, n, P_{user}) containing the grant l, a nonce n to prevent replay attacks and the user public key P_{user} . It is encrypted using the symmetric key K_{ticket} .

3.3 Client Authorization Protocol

In the following we are using an example to illustrate the client authorization protocol. We assume that only the email address of a user is requested and that an ID_ATTR record $R_{P_{user},email}$ exists in the namespace of P_{user} under the name email with the record data "john@doe.com". First, the client creates a request to access the email address of the user. Such a request contains three parameters: The requested attribute names - In this case "email" -, a nonce that will be included in the ticket and the public key P_{client} of the client. When the user receives the request, he must first make a decision if P_{client} is a trusted client. This process is discussed in detail in Section 3.5. If the user decides that the client P_{client} is trustworthy, the user creates an identity token including the email attribute. This token also includes a representation of his public key P_{user} as well as expiration and signature information. The user signs the token with the private key x_{user} and encrypts token and signature using a symmetric key K_{token} derived using ECDHE from the client public key P_{client} and a new ECDHE private key $K'_{ECDHE, priv}$. As a result, only the authorized client will be able to decrypt the token. The user stores the ECDHE public key $K'_{ECDHE, pub}$ along with the encrypted data in the GNS record $R_{P_{user},l}$ and publishes it under the grant l. The user responds to the client authorization request with a ticket (p, k, s) containing the grant.

When the client receives the ticket, it must verify the signature s and decrypt the ticket payload p by calculating the symmetric key K_{ticket} using the public ECDHE key $k = K_{ECDHE,pub}$ and his private key x_{client} . After checking the nonce n, the client resolves the token record $R_{P_{user},l}$ from GNS using the grant l. To decrypt the token the client must calculate the symmetric key K_{token} using the client private key x_{client} and the public ECDHE key $K'_{ECDHE,pub}$ contained in the GNS record. The client can now retrieve the identity attributes from the token. When the token expires the client can use the ticket grant again to retrieve a fresh token from GNS. If the token has been revoked or is not updated by the user it becomes invalid and must not be used any longer.

3.4 Token and Grant Management

The grants contained in tickets expire when the corresponding GNS record in the name system expires. The record expiration times are managed at the GNS-level using the respective operations and settings for records. Tokens have dedicated relative expiration times not directly related to the grant expiration time. If a token expires an updated token can be retrieved using the same grant until the grant is expired. New tokens that contain updated expiration times must be generated regularly by the user where token lifetime may be fixed or chosen by the user at issuance.

3.5 Trust Establishment

In our design we do not rely on a central authority, but on a decentralized publickey infrastructure where trust is not an absolute binary measure but rather a relational, subjective metric. This approach does not exclude the existence of highly trusted third parties, but it gives the user the option to choose what those parties are. For a user to make a reasonable decision whether or not to trust a client with a set of user attributes there must be a trust relationship between any of the user identities and the client. Technical details of trust establishment in GNS is sufficiently explained in [11] and the related work by Rivest et al [9].

In GNS, a public key can be translated into a human-readable name by performing a reverse lookup. The result of a reverse lookup reveals the trust relationship between the user and the client. If the user has a direct trust relationship with the client, a reverse lookup will return a single name that the user himself assigned, for example: "bob". If the user only has an indirect trust relationship with the client, the name will contain multiple labels separated with a ".": For example: "bob.alice.carol".

If the user has no direct or indirect trust relationship with the client, only a readable representation of the client public key instead of a name is available. The user can recognize that the client is unknown and he can decide if a token should be issued and what data it contains.

The user's decision is expected to depend on the context and the requested set of attributes. For example it is perfectly reasonable that a mailing list provider is requesting access to an email address, even if the user does not have a direct trust relationship. On the other hand interaction with the user's financial institution might incline the user to require a previously established direct trust relationship.

4 Protocol and Implementation

We have implemented the system as a components in the GNUnet peer-to-peer framework⁴. The authorization protocol is realized on top of HTTP utilizing REST services for token issuance and retrieval.



Fig. 1. Identity and attribute management of the implementation prototype.

The system consists of a user-side issue endpoint and a client-side token endpoint that both interact with GNS for token issuance and lookup, respectively. A user-side service is keeping track of all issued tokens across all user namespaces and updates expired tokens. The system's functionality for token issuance and management is exposed through a JavaScript user interface (see Figure 1). It is also used in the authorization protocol (see Figure 2) to prompt the user for authorization consent. The interface and an example client have been implemented separately and are available online 56 .



Fig. 2. Client authorization protocol.

The initial authorization request is an HTTP redirect response sent by the client to the user agent when the user accesses a client resource that requires user information, such as a web application. The redirect response contains the client

⁴ https://gnunet.org/svn

⁵ https://github.com/schanzen/gnunet-webui

⁶ https://github.com/schanzen/gnuidentity-example-rp

public key P_{client} , the nonce *n*, the requested attributes as well as a redirect URI. As the client cannot know the domain name and URI of the end user endpoint, it uses a protocol handler in the redirect response. The protocol handler redirects the user agent to the user interface. To authorize the client, the user chooses an identity, selects the attributes to share and consents to the request. If the user chooses not to accept the authorization request the protocol will conclude with an HTTP redirect back to the redirect URI including an error response. If the user consents to the authorization request an issue request is sent to the end user issue endpoint. The request includes a token expiration time, the nonce provided by the client as well as the client public key. The endpoint creates the token and adds the respective records in GNS. The endpoint responds to the issue request with an HTTP redirect response and the user agent is redirected to the redirect URI along with a new ticket as URL parameter.

The client exchanges the ticket provided in the URL parameter for a token by issuing a token exchange request to its own token endpoint. The endpoint resolves the token from GNS using the grant. After decryption and validation it returns the token in the exchange response. Issued tickets are JSON objects containing the ticket payload p, the ECDHE public key k as well as the signature s. The identity token is implemented as a JSON-Web-Token (JWT) [6].

5 Conclusion

In this paper we have demonstrated how the secure name system GNS can be used for decentralized user attribute sharing. We designed a system that removes the need for central service providers or trusted authorities by securely sharing identity tokens via the name system and relying on its inherent PKI for trust establishment. We formally verified our proposed security properties (see Section 3) using the Casper [8] script in the Appendix.

Our system fills the gap that existing privacy-preserving credential systems such as Idemix do not address. The system's availability does not depend on a third party service and it provides a requesting party with attributes signed by the user that cannot be forged by an attacker. Finally, the system allows for attributes to expire transparently and allow the requesting party to request and retrieve updated attributes on demand.

Performance measurements were out of scope for this work but are a concern and should be evaluated further. In a next step, we are planning to investigate how users can be familiarized with the management of multiple identities and how attributes can be asserted by third parties to address distributed authorization scenarios.

Acknowledgment

This work has been partially funded in the project PARADISE by the German Federal Ministry of Education and Research under the reference 16KIS0422.

References

- D. Brown. Standards for efficient cryptography, sec 1: elliptic curve cryptography. Released Standard Version, 1, 2009.
- 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*, pages 21–30. ACM, 2002.
- French data protection authority). Decision no. 2016-007 of January 26, 2016 issuing formal notice to FACEBOOK INC. and FACEBOOK IRELAND, January 2016.
- P. Gola, R. Schomerus, and C. Klug. BDSG Bundesdatenschutzgesetz : Kommentar. Beck, Mnchen, 8. berarbeitete und ergnzte auflage edition, 2005.
- 5. T. Hardjono. User-managed access (uma) profile of oauth 2.0, December 2015. https://docs.kantarainitiative.org/uma/draft-uma-core.html.
- M. Jones, J. Bradley, and N. Sakimura. Json web token (jwt). RFC 7519, RFC Editor, May 2015. http://www.rfc-editor.org/rfc/rfc7519.txt.
- M. Jones and D. Hardt. The oauth 2.0 authorization framework: Bearer token usage. RFC 6750, RFC Editor, October 2012. http://www.rfc-editor.org/rfc/ rfc6750.txt.
- G. Lowe. Casper: A compiler for the analysis of security protocols. Journal of computer security, 6(1, 2):53–84, 1998.
- R. L. Rivest and B. Lampson. Sdsi-a simple distributed security infrastructure. Crypto, 1996.
- N. Sakimura, J. Bradley, M. Jones, B. de Medeiros, and C. Mortimore. OpenID Connect Core 1.0 incorporating errata set 1. Technical report, November 2014.
- M. Wachs, M. Schanzenbach, and C. Grothoff. A censorship-resistant, privacyenhancing and fully decentralized name system. In *Cryptology and Network Secu*rity, pages 127–142. Springer, 2014.
- M. Wachs, M. Schanzenbach, and C. Grothoff. On the feasibility of a censorship resistant decentralized name system. In *Foundations and Practice of Security*, pages 19–30. Springer, 2014.

Appendix - Casper Sources

```
- Only relevant sections included for brevity.
 1
        ______
#P
2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9
        #Processes
INITIATOR(I, nc, CSK, CPK, G) knows
                                                       PK. GNSENC
       PK, GNSENC
USER(U, grant, CPK, G, data) knows
PK, SK(U), GNSENC
GNS(G) knows PK, GNSENC
                                                                                             #Specification
Secret(U, grant, [I])
Secret(U, data, [I, G])
Agreement(U, I, [G, data])
                                                                                    20 \\ 21
                                                                                     ^{22}
                                                                                     23
        #Protocol description
                                                                                     \frac{24}{25}
10
11
                     U
        0. ->I: U
1.I->U: nc
                                                                                             #Intruder Information
                     {{data}{CPK}}{GNSENC(grant, PK(U))}
% record
                                                                                     26
                                                                                              Intruder = Mallory
12
        2.U->G:
                                                                                             Intruder Knowledge =
{Gns, User, Mallory, nonce,
PK, SK(Mallory), cpk, GNSENC}
                                                                                     27
13
14
15
16
17
18
19
                                                                                     28
29
                     3.U->I:
        4.I->G:
                       gran
query
ord %
        5.G—>I:
                     record %
{{data}{CPK}}{GNSENC(grant, PK(U))}
```