Negotiating Trust in Identity Metasystem

Mehmud Abliz
Department of Computer Science
University of Pittsburgh
Pittsburgh, Pennsylvania 15260
mehmud@cs.pitt.edu

Abstract—Many federated identity management systems have been proposed to solve the problem of authorizing users across security domains. Although these solutions attempt to follow the user-centric design approach to empower users by letting them make important decisions on whether to release sensitive information, they do not provide much help to users in making good decisions. More importantly, privacy of user's identity related data is not very well protected in many of these systems. Some even fail to meet the security requirements of identity management system, and are susceptible to replay and man-in-the-middle attacks. In this paper, we compare identity management systems against trust management systems, and attempt to integrate certain trust management and trust negotiation concepts into the federated identity management systems. We choose an existing federated identity management system, called identity metasystem, and integrate trust negotiation into the system to provide a better user-centric and privacy preserving federated identity management system. We hope the new system can be well applied to collaborative environment as well as open systems.

I. INTRODUCTION

Identity management (IdM) is an area that concerns with management of the life cycle of digital identity. The importance of securely and privately managing identity is increasing as Internet is increasingly used in almost every aspect of people’s life. Meanwhile, the requirements of designing a secure and privacy preserving identity management system are also becoming more and more challenging. From user’s perspective, minimum amount of information need to be remembered in order to use the system securely and with convenience. Furthermore, identity management system should provide good protection of user’s privacy. From the perspective of providers of various Internet services, identity management system should minimize the cost of maintenance of up-to-date user account information, and should provide strong support for collaboration needs between service providers in different security domains. Making the problem more challenging, some of the requirements for such identity management systems are often mutually conflicting. Security requirements such as full auditing of system resource access can conflict with user privacy, and full user empowerment over identity can conflict with user convenience.

In efforts to provide better systems that meet the new requirements of identity management, a large number of proposals are introduced over the past few years, including OpenID[1], SAML[2], Shibboleth[3], Liberty[4], [5], and Identity Metasystem[6]. Most of these systems are designed centering the concept of identity federation. A federation is a set of organizations that establish trust relationships with respect to the identity information. Federation of identity usually has two meanings[7]: 1) The virtual reunion of a person’s user information stored across multiple distinct identity management systems; 2) A user’s authentication process across multiple systems or organizations. Most of these federated identity management systems mainly solve the problem of minimizing maintenance cost and providing user convenience via single sign-on (SSO). Although they try to empower users by letting users make important decisions on whether to release sensitive information, they do not provide too much help to users in making good decisions. More importantly, privacy of user’s identity related data is not very well protected. Some of those systems even fail to meet the security requirements of identity management systems, and are susceptible to replay and man-in-the-middle attacks.

Distributed trust management is a different approach proposed by researchers to achieve user authorization across security domains. Instead of managing identity of each individual user, trust management tries to make a decision about whether the credentials provided by the user allows that user to perform the requested operation. Trust management systems provide a new direction to address the problem of deciding eligibility of a subject to perform an action across security domains. Although trust management systems[8] and more advanced trust negotiation[9], [10] systems seem to have a very different approach than the federated identity management systems, they share a common goal of user authorization across security domains.

Thus, in this paper we study identity management and distributed trust management systems individually and comparatively, and investigate the advantages of integrating trust negotiation in identity management systems. Our contributions are following: 1) We provide an in-depth survey of major identity management systems, and summarize their strength and weaknesses; 2) We conduct a comparative study of trust management systems and federated identity management systems, and identify features of each type of
systems that are similar to or can be adapted by the other type of systems; 2) We define a set of requirements for a system that satisfies our design goals of user-centric and privacy preserving identity management, and propose an improved identity management scheme based on the existing identity metasystem architectures. We hope the new system can be well applied to collaborative environment as well as open systems. In designing the improved system, we would like to apply some of the trust negotiation concepts to our system and achieve better security and privacy results.

The rest of the paper is organized as following. In Section II we introduce four major proposals in federated identity management systems, and explain pros and cons of each system. We then compare trust negotiation and identity management and discuss the possibility of integration in Section III. Next we introduce our design goals in Section IV, and describe a threat model assumed in this study in Section V. And we introduce the improved system in Section VI. We then discuss possible future extensions to the proposed system in Section VII, and give a conclusion of the paper in Section VIII.

II. RELATED WORK

All of the federated identity management scheme we discuss in this paper have three major players in them. So, before we give detailed description of each system, we first explain these players.

- **User:** a person with certain identity and attributes, and wants to access a particular service protected by access policy. A user usually interacts with other players through an agent, such as a browser or a smart identity client software.

- **Service Provider (SP):** provider of services to the user over a network, such as Internet. The services provided are usually subject to some form of access policy.

- **Identity Provider (IdP):** the party maintains identity and attribute information about a user and provides assertion of validity of that user’s identity or attributes.

Note that service provider and identity provider does not have to be located in two different security domains, they can be co-located. Similarly, identity provider might be co-located with user in the same physical host. Meanwhile, a service provider can become a user of another service provider. One has to consider possibly different configurations when analyzing and designing federated identity management systems.

A. **OpenID**

OpenID is a decentralized user authentication and access control standard. OpenID is mainly designed to allow web users log onto different web sites using a single digital identity. OpenID uses address-based identifiers, users can identify themselves to other parties using 1) URL Ids, and 2) XRI i-names [11] (e.g. xri://=example.user). XRIs provide a persistent, protocol-independent address for any type of digital subject. Given the identifier, the *Identity Service Discovery* is a process of retrieving an XRDS document [12] describing the services available for a particular URL or XRI. XRI Resolution 2.0 [12] can be used for XRI and URL based identity service discovery, while Yadis discovery protocol[13] can be used for URLs.

![Figure 1. OpenID authentication process](image)

Figure 1 illustrates the basic workflow of OpenID authentication process:

1) User wants to access a certain resource on service provider. After choosing OpenID as her method of authenticating herself to the SP, the SP asks for user’s OpenID identifier. User sends her identifier, which can be either a URL or an XRI.

2) **Normalization and discovery:** the SP normalizes the user-supplied identifier, and performs a discovery on it and establishes the URL of the OpenID identity provider.

3) (Optional) The SP can establish an association (a shared secret) with the IdP using Diffie-Hellman key exchange [14]. The IdP uses the shared secret to sign subsequent messages, and it removes the need for SP requesting IdP to verify the signature after each authentication request/response.

4) SP redirects end user’s user-agent (UA) to the IdP with an OpenID Authentication Request.

5) IdP asks the user to authenticate herself, the specific method of authentication (user/password etc.) is decided beforehand by IdP. User-agent asks the user for credentials, such as user-name/password, relays this information to IdP.

6) After confirming authenticity of the user, IdP sends a message to user asking if she wants to tell SP the fact that she owns the identifier. After seeing the question by IdP, the user can decide to let IdP send her identifier.
to SP.

7) Authentication Response: IdP redirects the user’s user agent to SP together with the user’s identifier and the assertion that proves the fact that the user owns the identifier or a message that authentication failed.

8) SP can verify the token with IdP, if it did not performed step 3 to share a secret with IdP. If it already has an association with IdP, then SP uses the shared key to verify the signature in the assertion message.

9) If the token is valid, then SP redirects the user agent to the requested service. At this point, the user is authenticated to the SP.

In step 1, user can enter the URL or i-name of her identity provider instead of using her own identifier. And when user is redirected to IdP in step 4, IdP can present the user with the option of 1) using her own public identifier, 2) using the identifier of one of multiple personas the user may wish to expose to SP, or 3) a randomly generated identifier just for this SP. This is the so-called private addresses in OpenID, and it can achieve unlinkability of user identity across multiple service providers.

OpenID does not require usage of SSL/TLS. Messages are sent in the clear during OpenID authentication when HTTP is used. All protocol messages are encoded in POST or GET data format or using key-value format. Redirect messages are the URL of the receiver with the OpenID authentication messages appended to the query string. Direct messages are between IdP and SP and encoded in POST for request and key-value form for responses.

The main advantages of OpenID is that it is a lightweight protocol and achieves zero footprint at the user agent. OpenID also has a very simple trust model. Moreover, it builds the identity provider discovery into its design, and enables direct interaction between identity providers and service providers.

OpenID also has several weaknesses. It’s vulnerable to phishing attacks. A malicious service provider can redirect the user agent to a fake identity provider and obtain user’s authentication credentials. Another issue is the privacy. In OpenID, identity service discovery takes place when users supply their universally resolvable identifiers to an SP. Given this, SPs can uniquely identify user to profile user activities, not to mention the trivial effort an IdP can take to keep track of user’s activities across multiple SPs using the unique user identifier. Lastly, a user has to remember multiple URLs or XRIIs, making it difficult for an average user.

B. SAML

Security Assertion Markup Language (SAML) is an XML-based standard for exchanging authentication and authorization data between security domains, that is, between an identity provider (a producer of assertions) and a service provider (a consumer of assertions). The most important problem that SAML is trying to solve is single sign-on problem where a user who has signed on to one service can move directly to another without the need to authenticate herself to the second service. At the user’s request, the identity provider passes a SAML assertion to the service provider. On the basis of this assertion, the service provider makes an access control decision. SAML defines XML-based assertions, protocols, bindings, and profiles.

SAML Assertion refers to the general syntax and semantics of assertion messages. A SAML assertion in general asserts the following: assertion A was issued at time t by issuer R regarding subject S provided conditions C are valid. Three types of statements are provided by SAML: 1) Authentication statements, 2) Attribute statements, and 3) Authorization decision statements.

SAML Protocol refers to what information are transmitted as part of SAML request and response. A SAML protocol describes how certain SAML elements (including assertions) are packaged within SAML request and response elements, and gives the processing rules that SAML entities must follow when producing or consuming these elements. The most important type of SAML protocol request is called a query. A service provider makes a query directly to an identity provider over a secure back channel. Corresponding to the three types of statements, there are three types of SAML queries: 1) Authentication query, 2) Attribute query, and 3) Authorization decision query.

SAML Binding determines how SAML requests and responses map onto standard messaging or communications protocols (such as SOAP).

![Figure 2. SAML Single Sign-On](image-url)

SAML Profile is a concrete manifestation of a defined use case using a particular combination of assertions, protocols, and bindings. The most important SAML profile is the Web Browser SSO Profile.

The protocol flow is explained in Figure 2:

1) The user requests a target resource at the service providers. The service provider performs a security check on behalf of the target resource. If a valid security context at the service provider already exists, skip steps 2 – 6.
2) The service provider responds with a \(<\text{samlp:AuthnRequest}\) element that is base64 encoded in the \(<\text{samlp:Response}\) parameter. \(<\text{samlp:AuthnRequest}\) contains: a 160-bit string ID, a time-stamp, service provider’s name, and a URL that the service provider uses to verify a SAML response. The user agent issues a POST request to the SSO service at the identity provider where the value of the \(<\text{samlp:Response}\) parameter is taken from the \(<\text{samlp:AuthnRequest}\).

3) The identity provider processes the \(<\text{samlp:AuthnRequest}\) and performs a security check. If the user does not have a valid security context, the identity provider authenticates the user.

4) The identity provider validates the request and responds with a document containing a \(<\text{samlp:Response}\) element that is base64 encoded in the \(<\text{samlp:Response}\) parameter. The SAML response also contains a signature element which includes signature info, signature, and key info. The user agent issues a POST request to the Assertion Consumer Service (ACS) at the service provider. The value of the \(<\text{samlp:Response}\) parameter is taken from the \(<\text{samlp:Response}\). Assertion Consumer Service is the service that verifies a SAML response at the service provider.

5) The assertion consumer service processes the response, creates a security context at the service provider.

6) The SP then redirects the user agent to the target resource.

SAML defines general assertion formats and protocols which can be customized to various use cases. SAML architecture strives to achieve single sign-on, identity provider auto discovery, and user privacy. But identity provider discovery is hard in the general use case. It’s also very complex and heavy-weight. Like OpenID, SAML is also vulnerable to man-in-the-middle attacks. An attacker can modify the redirect address to forward the user agent to a fake identity provider. SAML requires IdP and SP to establish trust prior to single sign-on is performed, which makes it only suitable for a closed environment where every party has a certain level of trust in any other party. Lastly, IdP in SAML protocol always know which SP a user is visiting, which becomes a big issue when trying to preserve user privacy.

C. Shibboleth

The Shibboleth Architecture extends the SAML 1.1 single sign-on and attribute exchange mechanisms by specifying service-provider-first SSO profiles and enhanced features for user privacy. It defines a set of interactions between an identity provider and a service provider to facilitate web browser single sign-on and attribute exchange.

In Shibboleth architecture, an Identity provider is composed of following subcomponents:

- **Authentication Authority**: is a SAML-defined service that issues authentication assertions about users to service providers.
- **Attribute Authority**: processes attribute requests and issues attribute assertions. The attribute authority authenticates and authorizes any requests it receives. Shibboleth additionally requires that control of attribute release to service providers be available to both administrators and users. Shibboleth attribute authority MUST have the ability to authenticate requests and MUST implement some form of access control governing the release of specific attributes and values belonging to specific users to specific requesting service providers.
- **Single Sign-On Service**: is the first point of contact at the IdP. It receives and processes authentication requests sent through the browser from service providers. The SSO service initiates the authentication process at the IdP and ultimately redirects the client to the inter-site transfer service.
- **Inter-site Transfer Service**: issues HTTP responses to the user’s browser conforming to the Browser/POST and Browser/Artifact profiles. The inter-site transfer service interacts with the authentication authority behind the scenes to produce the required authentication assertion.
- **Artifact Resolution Service**: receives requests directly from a service provider to resolve a SAML artifact into the corresponding assertion in accordance with the Browser/Artifact profile.

A service provider in Shibboleth is composed of following subcomponents:

- **Assertion Consumer Service**: is the service provider endpoint of the SSO exchange. It processes the authentication assertion returned by the SSO service; initializes an optional attribute request; establishes a security context at the SP, and redirects the client to the desired target resource.
- **Attribute Requester**: sends SAML \(<\text{samlp:Request}\) message containing a \(<\text{samlp:AttributeQuery}\) element to attribute authorities and process the resulting attribute assertions. An attribute requester at the SP and the attribute authority at the IdP may conduct a back-channel attribute exchange once a security context has been established at the SP, bypassing the browser.

An optional Where Are You From (WAYF) service is operated independent of the SP and IdP. The WAYF can be used by the SP to determine the user’s preferred IdP, with or without user interaction. WAYF proxies the authentication request from the SP to the SSO service at the IdP.

Figure 3 describes the Shibboleth single sign-on protocol...
flow. Before the SSO protocol begins, the user, via an HTTP user agent, makes an HTTP request for a secured resource at the service provider without a security context.

1) The service provider issues an authentication request and redirects the user agent to either a WAYF or directly to an identity provider. A WAYF is typically used if the service provider wishes to delegate the task of identity provider discovery.

2) If a WAYF is used, then WAYF service interacts via unspecified means with the user agent to select an identity provider to which to redirect the user agent with the service provider’s authentication request.

3) The user is identified by the identity provider by some means outside the scope of the Shibboleth specification. This may require a new act of authentication, or it may reuse an existing authenticated session.

4) The identity provider issues a SAML `<samlp:Response>` message or one or more SAML artifacts to be delivered by the user agent to the service provider. Either the SAML 1.1 Browser/POST profile or Browser/Artifact profile may be used. If the Browser/POST profile is used, then either one or more assertions or an error status is passed directly through the user agent to the service provider in the response. If the Browser/Artifact profile is used, then one or more SAML artifacts are passed through the user agent to the service provider, at which point the service provider communicates directly with the identity provider to resolve the artifact(s) into assertions.

5) The service provider optionally uses the subject of the authentication assertion it received in step 5 to send a `<samlp:AttributeQuery>` to an attribute authority of the identity provider.

6) The attribute authority associated with the identity provider processes the `<samlp:AttributeQuery>` and returns a SAML response message, possibly containing one or more assertions containing attributes that apply to the user.

7) The service provider responds to the user’s user agent with an error, or establishes its own security context for the user and returns the requested resource.

Note that identity provider MAY initiate this sequence at step 4 and issue an unsolicited SAML response message or SAML artifact(s) to a service provider without the preceding steps. In addition to steps 5 and 6 being optional, an identity provider MAY include `<saml:AttributeStatement>` elements in the assertion(s) that it returns in step 4. This is commonly referred to as “attribute push”. A service provider MAY still perform step 5 at its discretion, whether or not attributes are received in step 4, although generally this is omitted, at least initially, when attributes have been pushed.

Shibboleth architecture has very similar advantages/disadvantages that of a SAML SSO, so we do not give further explanation for the pros/cons of this architecture.

D. Identity Metasystem

Identity Metasystem is an architecture for digital identity that enables people to have and employ a collection of digital identities based on multiple underlying technologies, implementations, and providers. Windows CardSpace is an identity selector client software by Microsoft for the Identity Metasystem. Identity metasystem tries to provide controlling power to the end-users on the fact that which information should be exposed to service provider.

Identity metasystem introduces the concept of information cards. Basically an Information Card, or infocard, represents a digital identity that the user can potentially present to a service provider. Along with the visual representation, each card also contains information about a particular digital identity. This information includes: 1) which Identity Provider to contact to acquire a security token for this identity, 2) what kind of tokens this identity provider can issue, 3) exactly what claims these tokens can contain.

There are two types of infocards, managed cards and self-issued cards. Managed cards are created by external identity providers, and delivered to the user’s machine via a method decided by each IdP, such as email or HTTP. Managed cards are digitally signed and comes with the IdP’s certificate, which is used to verify the identity of the IdP itself. Once the card is on the user’s machine, the user can install this card into the standard identity selector card store. This is also when the user must approve the IdP as a source for security tokens.

A self-issued card is created by the local identity provider inside the identity selector client. This type of cards could be used on sites whose goal is just to recognize multiple accesses by the same user. Those sites usually let users
choose their own username and passwords. Self-issued cards can contain only basic information, such as the user’s name, postal address, e-mail, and phone number. When a user chooses to submit one of these cards to a service provider, the self-issued IdP on that user’s system generates a SAML token containing the information the user has placed in this card. The SAML token is also signed with the private key obtained one or more information cards from various identity providers which are available to the identity selector. The provider’s domain, and client application sends a protocol starts by user trying to access a protected resource request a specific security token with a specific set of claims for this card at this service provider. To prevent SP track a user’s activities, the self-issued IdP creates a different key pair for every SP that’s accessed with this card.

Before we start explaining the authentication process, we would like to remind readers that when we say “user selects a particular card”, that means the user is actually choosing to request a specific security token with a specific set of claims created by a specific identity provider. In the figure 4, service requester means a client application (such as a browser) with identity selector (such as Windows CardSpace, DigitalMe, or Azigo) support. We assume that the user has previously obtained one or more information cards from various identity providers which are available to the identity selector. The protocol starts by user trying to access a protected resource in service provider’s domain, and client application sends a service request for that service on user’s behalf.

1) SP responds with its security policy which includes information such as what type of tokens are required, what types of claims should be included in each token and from whom these tokens should be obtained etc.

2) Next, the client application invokes the identity selector (for example, CardSpace) and passes the security token requirements it received from SP to the identity selector.

3) The identity selector then processes the token requirements of the SP, and optionally requests security policy (which is WS-SecurityPolicy[15]) of SP using WS-MetadataExchange[16]. In simpler case, SP’s policy can be expressed using HTML, and both the policy information and the security token can be exchanged with SP using HTTPS.

4) Based on the token requirements, identity selector can then calculates which infocards fulfill the security token requirements of the SP, and

5) shows user a list of matching cards.

6) User chooses one of the cards presented to her.

7) Identity selector then requests the security policy of the identity provider security token service (STS) corresponding to the selected information card using WS-MetadataExchange[16]

8) Identity provider returns the security policy of its STS. This policy specifies the security binding to use for requesting tokens.

9) Security token request: each information card specifies the required credentials to use for authenticating the user to the identity provider STS.

• If the card selected by the user is a self-issued card, then the local identity provider co-located with identity selector, acting as a Security Token Service, will generate a security token (SAML token by default) which includes assertion of required identity information.

• If the card represents an identity at one of the identity providers (IdPs), identity selector authenticates the user to the identity provider STS using the credentials specified in the information card, and request a security token with the desired claims from the identity provider STS using WS-Trust[17].

10) Upon receiving the token request, the IdP generates a token which can be of any type the SP has declared to accept in its token requirement or policy. The IdP can also optionally generates a display token to send to the identity selector. The display token is not forwarded to the SP, but is used by the identity selector to show to the end user what information the IdP is providing to the SP. The IdP also generates a proof key for the security token. The IdP sends the security token, the proof key, and optionally the display token to identity selector using WS-Trust[17].

11) Identity selector passes the tokens and keys received from IdP or the self-generated token to the client

![Figure 4. Identity metasystem authentication process](image-url)
application. The client application finally will present them to the SP.

12) SP verifies the token and required claims in it, and creates authenticated security context for the user, and redirects the client application to the requested service.

To protect the token from being captured on the wire and to ensure that the client application has obtained a legitimate token from the identity provider, the identity metasystem offers an extra protection mechanism known as proof of possession. The key used to demonstrate the sender’s knowledge of a secret asserted by identity provider is called the proof key. Identity metasystem inter-operability profile supports symmetric, asymmetric, and no-proof-key, i.e. the token with no proof key attached, and its type is specified in service provider’s policy. For better security, the use of asymmetric proof keys is strongly recommended. If the proof key is required to be asymmetric, a public/private key pair is generated by the identity selector and the public part is included in the token request in step 9 of the authentication process. The identity provider signs the security token plus this public key from identity selector, and forms the ‘Part B’ of its security token response. The whole message is then sent back to the identity selector, which in turn can prove its knowledge of proof key by signing the ‘Part B’ of the message it received from the identity selector, using its private key.

Identity metasystem provides unlinkability of user identity between identity provider and service provider. It also provides a phishing-resistant authentication to user. Meanwhile, it empowers the user by requesting real time user consent on the disclosure of identity information. Although the identity metasystem is designed for solving the inter-operability between different identity management frameworks, it fails to do so due to its requirement of supporting WS-* protocols. Higgins[18] instead provides the means to make the identity metasystem cross-platform, thus is usually said to be an implementation of identity metasystem although its scope is much larger.

Other disadvantages of the identity metasystem include 1) need for smart identity client on the user’s machine, 2) weak support for user mobility, and 3) strong dependence on user’s correct decision for certain security-critical tasks. We also give other security and privacy limitations of identity metasystem in section 5.

III. TRUST NEGOTIATION AND IDENTITY MANAGEMENT

A. A Comparison

We compared federated identity management systems to trust negotiation systems, and summarized their advantages and disadvantages in Table I.

An automatic trust negotiation system is typically used in peer-to-peer systems, so clients and servers have the same basic architecture. In federated identity management frameworks, IdPs, SPs, and clients all have different architectural components depending on that entity’s functionality[19]. In trust negotiation systems, authorization decisions are made based on the attribute credentials of both the service requester and service provider. Identity management systems although support both attribute based and identity based authorization, identity based access control is more widely used than the attribute based one.

In federated identity management systems, identity provider needs to be online at all time to service security token requests from the user. Thus security of identity provider is crucial to the security of the whole system. If identity provider fails to operate, then user’s won’t be able to access the protected service to which user needs security token from the identity provider. With trust negotiation systems however, attributes are certified once and owners of these attributes do not need to contact the trusted party who issued these certified attributes until the expiration time.

However when it comes to revoking the attributes, it is hard to design a good revocation mechanism in trust negotiation systems. On the other hand, federated identity management systems allows very easy revocation of attributes, basically revocation is achieved simply by identity provider refusing to issue security tokens to revoked attributes/identities.

Another disadvantage of trust negotiation systems is that users are required to manage their own credentials and policy, which can be very challenging for an average user. On the other hand, identity providers help users manage their credentials and security policies in federated identity management systems.

In trust negotiation, two negotiating parties do not need any prior trust between each other. Trust negotiation aims to achieve trust between two strangers dynamically by bilateral credential disclosure. On the contrary, federated identity management systems require prior trust between service providers and identity providers in order for the service providers to successfully grant users access to protected services.

User mobility in trust negotiation systems is hard to achieve since credentials that are stored in user’s machine need to be copied over to another machine in a secure way if user switches machines and moves around. In comparison, it’s always an easy problem to solve in most federated identity management systems (except for identity metasystem).

B. Integrating Trust Negotiation into Identity Management

Spantzel et al in [19] proposed integrating trust negotiation into the identity management. This is the first and only proposal that suggested such an integration to the best of our knowledge. Although this paper observes several important similarities and differences between trust negotiation and identity management and aims to achieve a better protection of user’s sensitive information, the architecture proposed by it has several weaknesses.
First of all, paper defines session tickets and trust tickets as the main building blocks of their trust negotiation protocol[19]. A session ticket contains the service requested and user ID and signed by the service provider (the service provider defined in the paper is equal to the combination of service provider and identity provider we defined at the beginning of this section), but not encrypted. So session tickets expose user’s identity and the services the user is trying to access to anyone who’s listening, thus seriously violating the privacy of the user. Moreover, a trust ticket contains all the service providers a user has visited recently and the results of the user’s trust negotiations at these service providers. We can easily see that this can expose very sensitive user history information to anyone who is eavesdropping, and service providers do not need to do anything in order to get the user’s history of online activity. To make the matters even worse, both types of trust tickets are implemented using cookies, which, as we all know, are not a very secure way to share such sensitive information.

Secondly, according to the FAMTN architecture the paper introduced, a service provider might have to negotiate with many other service providers that a user has visited prior to visiting this service provider. This can result in long authorization delay for the user, and can easily be manipulated by DoS adversary to launch serious DoS attacks.

In comparison, our proposed system achieves better security, privacy, and usability. We explain why in section 5.

IV. OUR DESIGN GOALS

During the course of our study, we identified a large set of design requirements and principles. Kim Cameron’s laws of identity[20] defines a set of 7 rules that are widely quoted by identity management system researchers. [21] also identifies a set of requirements that user-centric identity management system should satisfy. In this section, we mainly elaborate on the four key design goals we would like to achieve and that are not strongly supported in existing systems.

1) User Control. Each user in the system must have full knowledge regarding the information about her that is to be disclosed and to whom it is about to be disclosed. Meanwhile, the user must have full control over whether to disclose that particular information. Another very important observation to be added is that user should have control over who provides/stores her identity related information.

2) Minimal Disclosure. In our definition, minimal disclosure means not only the disclosure of fewest number of claims, but also the disclosure of least identifying information. We also require sensitive informations are only disclosed to parties that have a necessary and justifiable place in a given identity relationship. This goal can be achieved by using selective disclosure and conditional release policies.

3) Privacy. Our definition of privacy includes following: 1) user anonymity, which means users remain anonymous throughout a transaction; 2) unlinkability of user’s identity across multiple transactions; 3) non-traceability of user’s transaction history. The non-trackability requirement for privacy is mutually conflicting with the auditing requirement, thus it’s very challenging to achieve both. But we insist that this feature of identity management should be provided in the system and enabled by default. The minimal disclosure goal we defined above can actually help improve user’s privacy.

4) Usability In order for a tool to be effective, it must allow intended users to accomplish their tasks in the best way possible. This goal can be achieved by providing users with intuitive and easy to use user interface, giving users consistent user experience, and automation of processes that need manual execution from users. Adding to this, we also believe it’s very important not to let users make hard security decision that we designers couldn’t make, or put it in another
way, we should have a good balance between "user empowerment" and usability. Least we can do is to provide users with good suggestions and warn them about the consequences of a decision they are about to make.

After carefully studying the four federated identity management systems we described in section 2, we find out that identity metasystem matches the most design goals of ours compared to SAML, Shibboleth and OpenID. Thus, we decided to build our system based on the current identity metasystem architecture. Yet, even identity metasystem cannot satisfy all of our requirements.

Although, identity metasystem does a good job in terms of empowering user, its usability remains a question. Especially, users are faced with the difficult decision of choosing the least identifying card or picking the least identifying set of attributes for a given card. Currently identity selector algorithm only calculates and presents to users all the satisfying cards, and once a card is picked it retrieves a token from IdP that contains all the attribute claims required by service provider. Regarding privacy, identity metasystem provides unlinkability of the user’s identity across multiple service providers, but identity provider can force identity selector to send service provider’s identity in token requests by adding \texttt{<RequireAppliesTo>} element in the information cards (see [6] section 3.3.3 Token Scope). IdP can keep track of all the activities of the user using this method. The user is never informed about this fact, nor is she provided with a choice to enable or disable such an activity tracking. In terms of minimal disclosure, there is no enforcing mechanism to achieve conditional release. All attributes required by SP is sent to the SP without requiring the SP to disclose certain evidence to justify the release of those attributes. We propose a better solutions for these problems in section 5.

V. Threat Model

Before we describe our attacker model, we would like to first point out some of the assumptions we make in our system. First of all, we assume that the cryptographic algorithms, such as signature and certificate signing algorithms, used in our system is secure, we only use cryptographic primitives that are considered safe to use for next 10-20 years, we don’t use certificates that are signed using algorithms such as MD5 which are considered harmful today[22]. Secondly, we assume certificate authorities that issue the certificates to any one of the players in our system is uncompromised. Moreover, our system assumes that a user has enrolled with at least one identity provider, and identity provider provides local authentication services to the user. We also assume each user has the smart identity selector client installed on her machine, and the identity client is not compromised.

In our system, we assume the following types of adversaries.

- **Passive eavesdropper**: if communication is not encrypted, this type of adversary can eavesdrop on communication between any two communicating parties, and collect useful information such user identity or attribute information, steal security tokens, session identifiers, and any other data which can be used against compromising the system’s security or privacy.
- **Active eavesdropper**: except for having all the ability of a passive eavesdropper, this type of eavesdropper can actively modify the content of messages sent between any two communicating party or can inject new messages, thus has the ability to compromise the integrity of messages, replay of messages, redirect communicating parties, downgrade the security of the system (also called dumbing down attack) etc.
- **Man-in-the-middle**: this type of adversary can make independent connection with the two communicating parties and relay messages between them, and making them believe that they are directly talking to each other when in fact the entire conversation is controlled by the attacker. A man-in-the-middle attack can only be successful when the attacker can impersonate the endpoint to the satisfaction of the other.
- **Malicious service provider**: malicious service providers can collude with each to correlate user attributes and try to uniquely identity users, thus violating the privacy. Malicious service provider might also be able to impersonate the user by using the security token received from the user at another service provider that accepts same type of security token.
- **Dishonest identity provider**: if identity provider gets compromised, then security of whole federated identity management system gets compromised. Thus, we assume that identity provider is not compromised. We use the term 'dishonest identity provider' to represent the case where identity provider tries to keep track of user’s online activity without a consent from the user.
- **Denial of Service adversary**: this adversary can launch DoS or DDoS attack on service provider or identity provider. We only consider the DoS attacks that are possible because of the vulnerability existed in the identity management layer of the system, we don’t consider DoS attacks on layers below or above.

VI. Trust Negotiation in Identity Metasystem

As we mentioned in design goals section, we selected identity metasystem as our base system since it satisfies largest subset of our design goals. In this section we discuss how we can improve the security of attributes and privacy of user by using trust negotiation techniques within identity metasystem architecture. In addition to that, we also discuss other possible improvements to the identity metasystem.
A. Proposed system

First, we need finer-grain control of user attributes. For each attribute assertion requested by identity selector identity provider signs it independently. Then, we need attribute release policy for each information card which is created by identity provider at the same time when information card is created. The attribute release policy is sent to the user together with information card and stored in the release policy database when information card is installed. Each attribute release policy includes a set of conditions that need to be satisfied for each attribute to be released to an external party. For managed information cards, release policy is created by external identity provider, and partly editable by the user. For self-issued information cards, default release policy is created by self-issued identity provider, and are all editable by the user. As with information cards, these policies need to be encrypted using a key derived from a password only known to the user, and are securely stored in the user’s computer.

Next, we need modification to the matching information card calculation algorithm of the identity selector. Currently, the Identity Selector Interoperability Profile[23] does not specify matching multiple sets of information cards that satisfy the token requirements of the service provider, thus we need to add such functionality to the identity selector.

To have trust negotiation capability, we need to add a trust negotiation module into our smart identity client. This trust negotiation module can work in parallel with the identity selector, and trust negotiation module plus the enhanced identity selector module comprises the smart identity client. We use an example to go through all the steps that are required for authenticating the user to a service provider.

Figure 5 explains the general structure of the smart identity client and shows the interactions between SP, user and IdP. In this example, a user wants to buy a laptop from Apple store using a discounted loan provided by Apple. Apple store requires that the user should have a good credit standing and should provide a valid mailing address. The user has several information card including an EquiFax credit verified card and a USPS address verified card. Attribute release policy for EquiFax credit verified card requires that the credit score attribute inside this information card should only be disclosed to an entity with a valid Better Business Bureau credential, and attribute release policy for USPS address verified card states that mailing address can only be disclosed to an entity with a valid Non-Share Consent credential from a state local government agency.

The authentication process starts when the user selects the laptop she wants to buy plus the loan program and submits these information to Apple store site.

First, the security policy of Apple store is retrieved by user’s identity client enabled browser, and the user’s browser invokes the identity selector and passes the Apple store’s security policy to the identity selector. The security policy of Apple store states that the user has to provide a claim asserting the credit score of the user from one of the three credit agencies EquiFax, TransUnion, or Experian. The security policy also states that the user has to provide a claim asserting a valid mailing address within the United States from USPS. Identity selector can then calculates which infocards fulfill the requirements of the Apple store’s security policy. Identity selector finds a combination of two infocards, namely EquiFax credit verified and USPS address verified cards, and prompts user if she wants to send assertions of attributes included in these infocards to the Apple store. If user says ‘yes’, then identity selector sends two security token requests for EquiFax identity provider and USPS identity provider at once.

At the same time, trust negotiation module inside the smart identity client is invoked and security policy of Apple store plus the information card references of EquiFax credit verified card and USPS address verified card are passed to the trust negotiation module. The trust negotiation module, which can work in parallel with the identity selector, can thus query the release policy for EquiFax credit verified and USPS address verified cards from release policy database. Based on the sensitivity of these two attributes, the trust negotiation module first sends the release policy for USPS address verified card to the Apple store’s trust negotiation module. Figure 5 actually does not show the details of subcomponents of the service provider, but we assume that every service provider that uses our system is configured to contain a trust negotiation service which is interoperable with the trust negotiation module in the user’s machine. Address of the trust negotiation service at the service provider is given in the service provider’s security policy.

Next, trust negotiation service of Apple store sends the Non-Share Consent credential to the trust negotiation module of the user’s identity client. The trust negotiation module then polls the identity selector to see if security token for...
USPS address verified card has been retrieved. If so, the trust negotiation module discloses this token to the Apple store’s trust negotiation service, together with the release policy for EquiFax credit verified card. Apple store’s trust negotiation service then sends Apple store’s BBB credential to user’s trust negotiation module, and user’s trust negotiation module sends back the EquiFax credit verified security token as soon as it’s available from identity selector. Apple store’s trust negotiation service then informs the Apple store the validity of user’s information. The trust negotiation module inside the user’s identity client then informs the identity selector about the successful negotiation result, which then in turn informs browser to send an HTTP request to Apple store’s website, Apple store’s website replies with an HTML page which displays the successful completion of the transaction. For all communications between user and service provider and between user and identity provider, we use SSL 3.0 or above to achieve message confidentiality and integrity.

There is one additional improvement we can make to this authentication process. After the successful negotiation between user and service provider, a service provider may choose to accept a non-identifying public key from the user, and choose to use a nonce challenge-response mechanism to authenticate user for a predefined period of time. Since identity selector defined in identity metasystem already has capability to creating and managing public/private key pairs[6], we only need trivial modification to the identity selector to achieve this functionality.

B. Minimal disclosure and privacy

Although identity metasystem emphasizes minimal disclosure, it does not provide any specific mechanism to enforce that requirement. Using trust negotiation between the user and service provider achieves conditional release of the user attributes. Attribute release policy also enables the enforcement of the requirement which says “sensitive informations should only be disclosed to parties that have a necessary and justifiable place in a given identity relationship”, since release policy enforces the receiver of an attribute to disclose satisfying credentials before the attribute assertion is disclosed to the receiver.

Next we propose that the user should have the option to enable or disable sending the identity of the service provider she’s interacting with to her identity provider. This can improve the user’s privacy since identity provider will not be able to keep track of user’s transaction history at multiple service providers, and also gives the user greater power to control her privacy. If an identity provider has policy that requiring audit of user’s activity for accountability reasons, then the user has to be informed about this IdP policy by identity selector before an infocard from this IdP is installed on the user’s machine.

C. Usability

The inclusion of the trust negotiation module in the identity client of the user helps improve the usability of the system, compared to the case without the trust negotiation module. Without the trust negotiation, the user has to verify whether the service provider is trustworthy before allowing the disclosure of her attributes to service provider, and the making of this decision is a security-critical task. To make a good decision, the user has to know all about what attribute should or should not be disclosed to which party. We can’t say much about the ability of an average user to accomplish this on her own, but one thing is sure that users need to spend great efforts in order to make a good decision. Automatic trust negotiation frees the user from deciding whether to disclose a certain attribute to service provider, thus achieving better usability.

Of course, the user is responsible for managing the release policy for self-issued information cards, but self-issued identity provider can be configured to create strict default release policy for attributes. Moreover, attributes in self-issued information cards are not certified by a trusted third party, thus lacks credibility from the service provider’s point of view. The user can take advantage of this fact and provide meaningless fake values for the attributes that she has privacy concerns for, thus providing an easy alternative to a more complicated policy management task. As we already mentioned earlier in this section, release policy for managed information cards are provided and maintained by external identity providers, thus the user does not have to do anything if she does not want to or does not know how to manage policy.

To further improve usability, we require that identity selector should help the user make better decision of choosing cards by things like ordering cards by least information disclosing to most information disclosing, when they are presented to the user for selection. Meanwhile, identity selector should provide the user with good explanation about the consequences of each decision that it wants the user to make.

VII. Future Work

We give several possible future improvements to our proposed system in the following.

• **System implementation and evaluation.** We would like to use available open source implementation of identity selector, service provider and identity provider software for identity metasystem and the open source trust negotiation frameworks to implement our proposed system. We would like to evaluate the system in terms of privacy, minimal information disclosure, usability, and performance.

• **Trust negotiation with identity provider.** In current identity metasystem standard, the user simply will not
be able to access the requested service when there is no matching information card in store that satisfies the token requirements of the service provider. We think identity selector should be able to pick an identity provider which might be able to provide a new matching information card and corresponding security tokens, given that the user holds information cards that can satisfy identity provider’s requirement for issuing the new matching information card.

- **Mobility.** Identity metasystem provides support for mobility of information cards stored in the user’s local host by defining “Information Cards Transfer Format” (see [6] section 6). In order to use the same information cards on multiple machines, the end user has to export all of his information cards, including private keys for self-issued cards, to a file, and then imports it into the identity selector on another machine. Although the import/export process is conceptually simple, the user has to remember to remove all the cards including the secret data from the machine before she leaves. Moreover, even if she remembers to do so, it’s possible that her cards and credentials are copied secretly by malicious programs during the session. Hoang et al proposed storing user credentials on the user’s trusted mobile devices to solve the mobility issue[24]. We would like to further look into this and other proposals for solving the mobility problem in a secure and user convenient manner.

- **Using zero-knowledge proofs for attributes.** Using zero-knowledge proofs, we can eliminate the need for revealing the actual values of the attributes in the system, which means that no party will have to trust any other party to the level that it has to reveal the actual values of the claims. This can greatly improve the privacy of user attributes, but it might also add significant overhead to the system. [25] proposes applying zero-knowledge proofs to protect digital identity in federated identity management system, but it’s not well suited for identity metasystem. We need to further study this issue and better evaluate the advantages and disadvantages of adopting zero-knowledge proof techniques in our system.

- **Possibility of using k-anonymity or l-diversity for data aggregation prevention.** Minimizing data aggregation at the service provider is one of the important requirements for federated identity management systems. But identity metasystem does not provide any mechanism or guideline for achieving data minimization. One possible improvement of identity metasystem in the future can be looking at the possibility of applying k-anonymity[26] and l-diversity[27] techniques to the privacy of the user attributes in the identity metasystem.

**VIII. Conclusion**

In this paper, we first evaluated the strength and weaknesses of existing federated identity management system. We find out that identity metasystem better suits our design goals than other systems studied in this project. We then compared federated identity management systems to trust negotiation systems, and summarized their advantages and disadvantages. After that we argued that trust negotiation can greatly improve user privacy and attribute security while significantly increasing the usability of the identity metasystem. Then, we proposed an improved system based on the identity management architecture, and showed how our proposed ideas can significantly improve the identity metasystem in terms of attribute security, user privacy and user convenience. Last but not least, we give several possible future improvements to our proposed systems.

**References**


