Hardening Persona – Improving Federated Web Login

Michael Dietz and Dan S. Wallach
Rice University
{mdietz, dwallach}@rice.edu

***Abstract*—Federated login protocols for the Web are intended to increase user security by reducing the proliferation of passwords that users are expected to remember and use on a day to day basis, however these protocols are vulnerable to recent attacks against TLS that allow attackers to extract session cookies and other such authentication tokens from within TLS sessions. A recent technique, TLS-OBC (origin bound certificates), allows these tokens to be hardened against extraction. This paper describes the design and engineering of OBC-based extensions to federated login protocols. We present two OBC-based variants on the popular Persona federated login protocol, formalizing them with BAN logic and using the automated proof checker from the related Nexus Authentication Logic. We also present a proof of concept implementation, exploring the necessary browser extensions and server support.**

I. Introduction

The humble username and password remain the most common mechanism for users to login to a website. While their ease of use and implementation have kept them as the de facto web login standard for years, there are a number of serious problems with them that have lead to weaknesses in the security ecosystem as a whole.

First, users reuse passwords—an inevitable consequence of the requirement to memorize credentials across multiple sites [24], [14]. Consequently, one compromised password allows for a broader impact [49], [17]. Likewise, users are still vulnerable to phishing [2], [15] attacks where an impostor website or email entices them into giving up their credentials. Finally, attackers that can compromise a root certificate authority can inject themselves as a man-in-the-middle (MITM) in a TLS connection to observe sensitive user credentials [28], [43]. Such attacks lead to account compromises, with all the attendant costs.

Federated login systems like SAML [30], OpenID [44], OpenID Connect [47], and Persona [39], were meant to solve these password management problems. Users log in once to an identity provider (IDP) and thereafter see nothing other than okay/cancel dialogs when connecting to relying party (RP) web sites. Federated login systems eliminate the need for a separate username and password at each RP. Unfortunately, these systems have seen relatively limited real-world adoption.

Federated login systems, like regular web login systems, typically store security-sensitive authentication state in HTTP cookies. A variety of attacks against TLS have, unfortunately, been able to extract these cookies (see, e.g., BEAST [46], CRIME [26], Lucky Thirteen [7], and BREACH [42]). Furthermore, attackers who can procure a fraudulent certificate, and thus conduct MITM attacks against the TLS protocol, will have access to cookies without any additional effort (see, e.g., the Diginotar debacle [6], [43]). These attacks are not theoretical; they are being employed by adversaries in the wild [51] and, as a consequence, federated login protocols that rely on TLS are vulnerable.

There have been several recent advances towards eliminating TLS MITM attacks against authentication credentials for initial and session based authentication using a technique called TLS origin-bound certificates (OBC) [21], [8], [20], but we are unaware of any work to address this threat in federated login protocols. This oversight presents a weak link in the web identity ecosystem where a user's identity at their IDP is well protected, but their identity at any RP is not.

One does not simply flip a switch, turn on TLS-OBC, and solve the MITM problem. OBCs are a cross-layer solution, using digital signatures from the TLS layer to enhance authorization cookies at the HTTP layer. As such, using OBC's facilities to improve the security of federated login will require some design and engineering. We decided to start with Persona, a recent federated login system designed at Mozilla, as a jumping off point, hardening its underlying BrowserID protocol. We want to consider how TLS-OBC's use of client-side public key certificates can be extended from its initial client-server login origins to also support the IDP/RP distinction in federated login systems. This requires extending the Persona protocol as well as creating new JavaScript API visibility to allow for the IDP and RP to communicate by proxy in the browser without compromising the underlying key material.

The key contributions of this work are:

- We formalize and analyze Persona's existing BrowserID protocol.
- We present two variants on Persona, hardening the BrowserID protocol against MITM attackers using TLS-OBC to bind identity assertions to specific TLC channels.
- We formalize and analyze our extensions to BrowserID using BAN Logic and the Nexus Authorization Logic.

NDSS '14, 23-26 February 2014, San Diego, CA, USA
Copyright 2014 Internet Society, ISBN 1-891562-35-5
http://dx.doi.org/10.14722/ndss.2014.23008

- We use the Nexus mechanized proof checker to verify our proofs.
- We discuss a proof of concept implementation of one of our Persona variants.
- We give some insight into how this work can be used to harden other existing federation protocols and discuss potential directions for future work.

II. Background

In traditional Unix clusters, the problems surrounding user authentication were simple. As user need merely present their username and password credentials to the local machine, which might then fetch hashed passwords and related credentials from a trusted directory service. When a user from one machine needed to get a resource from another, such as a file from an NFS file server, the local machine was trusted to enforce the access control rules. Obviously, this can't work in an environment when computers might be untrusted.

Kerberos [53] and related systems like Windows Active Directory approached this problem by having a user to authenticate to a central authority, allowing this central authority to *speak for* the user's identity to other services within the network, and giving the user's computer suitable credentials to access only the resources to which the user is authorized. Kerberos is an effective solution for access to services with dedicated applications and within a single intranet, but it was designed well before the web and was never meant to support Internet-scale authentication and authorization.

On the web, the problem of federated login, also sometimes called digital identity management, must necessarily scale to handle large numbers of users, and cannot assume any centrally trusted servers. Consequently, a variety of protocols have been developed that have the necessary scalability and that integrate cleanly with web browsers and servers (e.g., SAML [30], OpenID [44], OAuth [23], OAuth2 [31], and OpenID Connect [47]).

Terminology. Each of these protocols shares a common language for discussing the actors that take an active role during a run of the protocol. We assume that there are many *users* that wish to authenticate to an *identity provider* (IDP). The user and IDP are assumed to have apriori agreed upon a set of authentication credentials that the user will need to present in order to authenticate with their IDP. As we'll be focusing on federated login for the web, we assume that each user is using a browser, or user-agent, to communicate with their IDP.

After IDP authentication, users can then authenticate themselves to a *relying party* (RP). Relying parties are websites that wish to learn the user's identity, to create a local account for the user or to authenticate them on a returning visit, all without establishing another new username and password for the user to remember. In these systems, the RP must necessarily trust the IDP to vouch for a user's identity.

The message sent from an IDP to RP that establishes the user's identity is called an *identity assertion*. OAuth and OpenID Connect take this a step further and allow users to *delegate rights* from the IDP to the RP. For example, a user might grant access to their online calendar to a third-party service. All the user sees is an okay/cancel dialog, and the RP is given a credential that allows it access to the user's account on the IDP. In this paper, we're only concerned with third-party login, not with rights delegation, although our techniques could apply to both cases.

A. *Persona*

Mozilla's Persona [40], [1] is a successor to many existing federated login protocols. Persona's design goals are:

- Eliminate site-specific passwords.
- Use asymmetric key cryptography to link identity assertions to a private key controlled by a user's browser.
- Prevent the IDP from tracking the user's logins at RPs (i.e., while an IDP can potentially impersonate a user, it cannot simply record every site the user visits).
- Allow for simple implementation and deployment to the user's browser and RPs.

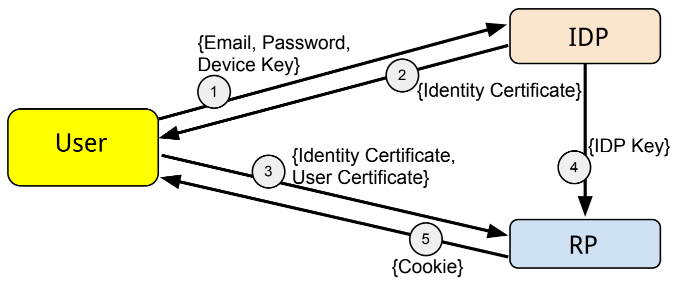
IDP
User
RP
{Email, Password, Device Key}
1
2
{Identity Certificate}
{IDP Key}
4
3
{Identity Certificate, User Certificate}
5
{Cookie}

Fig. 1: Persona Login Flow.

Persona login flow. When a user wishes to login with Persona they must first login to their IDP as shown in step 1 of Figure 1. If this is the first login on a new device, the user's browser then generates a new *device key-pair* and transmits the public key to the IDP in step 2. The IDP then generates a signed identity certificate, linking the user's email address to the device key, then sends it to the browser for storage in step 3. The device key and identity certificate are good for twenty-four hours and can be reused with multiple RPs within that validity period.

When the user wishes to login to an RP that supports Persona, the user's browser generates a user certificate: a signature with the device key over the web origin of the RP. This RP-specific user certificate and the identity certificate are then transmitted to the RP for verification in step 4. User certificates have a validity period of five minutes or less, and as such are only used for initial login. Finally, the RP verifies the user certificate and identity certificate and, if everything checks out, issues a traditional session cookie to the user's browser in step 5.

Note that the same identity certificate is used for all RP interactions over the next 24 hours without any other interaction with the IDP whatsoever. This places less load on the IDP and ensures that it learns nothing of the specific RPs who rely on it. The only RP-IDP interaction necessary is for the RP to learn the IDP's public key, used to sign the identity certificates. (We note that colluding RPs would observe the same identity certificate if the same user connects to both within the same 24 hour period. This could allow RPs to do some tracking of their own.)

The current implementation of Persona is implemented as a JavaScript shim that performs the device key generation and user certificate generation within the browser. At some point in the future, it is assumed that this functionality will be provided by the browser runtime or a browser extension rather than a JavaScript library.

B. *TLS man-in-the-middle attacks*

Persona improves upon existing federated login protocols in its ability to resist user tracking by the IDP. The binding of identity assertions to the user's device key also improves Persona's resistance to attacks that might try to impersonate the user. Unfortunately, Persona relies heavily on the protection of TLS in the initial establishment of the user's device key and to prevent attackers from stealing and reusing identity assertions.

It's commonly assumed that any communication through a TLS channel is safe from leaking information to an adversary. However, recent attacks [43], [6], [32] against the TLS certificate authorities, that serve as the root of trust for TLS authentication, have given adversaries a man-in-the-middle (MITM) attack, presenting a forged TLS server certificate to their victim while simultaneously establishing a connection to the server. More recent work [46], [26], [7], [42] has shown that an attacker doesn't even need a forged TLS certificate to extract sensitive data flowing over a TLS channel. These attacks exploit the compression and padding within TLS (or HTTP), using the server as an oracle to reveal specific values (e.g., authentication cookies/tokens). Once extracted, an attacker can then connect to the server as if it were the returning user.

Origin bound certificates (OBCs) counter these attacks by having the browser generate asymmetric key pairs for every web origin, established during the initial handshake between a new browser client and a server. Even if an attacker could extract the session cookies, the attacker could not forge the client's cryptographic authentication. The server will reject cookies that come over a channel without this authentication. As such, oracle attacks against TLS-OBC have no value for an attacker; they may still learn a cookie value, but they can't use it. Even MITM attacks against TLS-OBC fail, because the attacker cannot impersonate the client's OBC certificate. (Securing the initial login phase against an active adversary is an additional challenge, discussed in Czeskis *et al.* [20].)

In contrast to the reliance of Persona and OAuth2 on TLS to provide security, the OAuth1.0 [23] protocol instead requires applications to sign their requests using a secret key. This approach avoids the problems introduced by a TLS MITM but the tradeoff is that applications must be trusted to protect key material, which is difficult for javascript or mobile applications, and correctly implement the OAuth1.0 signing process.

MITM attacks against Persona. We will focus on two locations where Persona is vulnerable to attack by a TLS MITM. In order to reason about this attacker model, we first formalized the Persona protocol into the Security Protocol Notation [53] as shown in Figure 2. This formalization specifies the flow of messages between the principals participating in the protocol where the notation $X \rightarrow Y$ means X sends a message to Y. We use B to denote the user's browser, U is the user's email address, I is the identity provider, and R is the relying party (note that when I and R are used in a message they represent the name of the principal, in this case the web origin of the IDP and RP respectively). Key pairs take the form of $\left\langle K_X, {K_X}^{-1} \right\rangle$ for the public and private keys, respectively, for principal X. Timestamps T indicate when they are minted and by whom.

$$\begin{aligned}
B \rightarrow I &: U, K_B && (1)\\
I \rightarrow B &: \{U, I, T_I, K_B\}_{K_I^{-1}} && (2)\\
B \rightarrow R &: \{R, T_B\}_{K_B^{-1}}, \{U, I, T_I, K_B\}_{K_I^{-1}} && (3)\\
I \rightarrow R &: K_I \qquad (\mathit{cached}) && (4)
\end{aligned}$$

Fig. 2: Persona Protocol

We first consider a MITM attacker who can manipulate the first message in Figure 2. This message establishes the device key that the user's browser will use to generate future user certificates and in Persona is assumed to be sent over a TLS channel. The Persona protocol makes the assumption that TLS will prevent an attacker from manipulating, or stealing and replaying, this message. In the presence of a TLS MITM this assumption is invalid and an attacker can follow the protocol with their own device key, causing the identity provider to mint an identity certificate that links the attacker's device key to the user's email address U in the second message of Figure 2. The MITM attacker can now authenticate directly with the any RP, based on its new identity certificate, until that certificate expires (typically in 24 hours). We note that this attack requires that the attacker act both as MITM and have the ability to authenticate to the IDP as the user (via a stolen authentication cookie or phished credentials).

There are several existing techniques that could be used to prevent this attack. The secure remote password protocol [56] would allow the browser to establish ownership of the device key in message (1) while preventing a MITM attacker from injecting their own device key. TLS certificate pinning as implemented by Google's Chrome browser [35] or the Trusted Assertion for Certificate Keys [37] internet draft could also be used to allow the browser to detect the presence of a MITM attacker and abort the connection to the attacker. While there are known solutions for preventing this attack against message (1), it's important to note that a mechanism for the secure establishment of the device key is not specified in the current Persona protocol nor is it reasonable to assume that all IDPs will be able or willing to support strong authentication techniques.

The second attack we'll focus on revolves around message (3). In this message, a browser-generated user certificate and IDP-generated identity certificate are delivered to the RP for verification. In the current Persona protocol, there is no challenge-response to verify that the principal that delivers message (3) to the RP is really in possession of the device key K_B. Rather the timestamp T_B included with the message is used to determine the freshness of the user certificate. This design decision allows a TLS MITM attacker to intercept message (3) in flight and use it to login to the RP, potentially triggering the RP into minting a session cookie that the attacker can use to subsequently login as the victim.

Persona makes no requirement on how the RP should authenticate the user for subsequent requests after the initial Persona login, where the user certificate and identity certificate are verified, so it's easy to imagine that many RPs will fall back to HTTP cookie-based authentication tokens. A clever attacker therefore need not even replay message (3), but could just MITM the user's subsequent connection, stealing the HTTP cookie for subsequent logins.

C. *Threat model*

The threat model that we will use for the remainder of this work is informed by the discussion of Persona and TLS MITM attackers above.

We assume that clients and servers do not have malware on their devices. The presence of malware implies that the device cannot be trusted to protect a private key and therefore any protocol that depends on the ability of a client to keep the private key secret will fail. We note that Trusted Platform Modules (TPMs) can potentially be used to ensure a private key stays secret in the presence of malware, although malware in control of the host might still be able to use the TPM hardware as a signing oracle.

We also assume that web apps are free from cross-site scripting (XSS) exploits. An attacker that can mount an XSS attack can easily impersonate the user, as the browser has no concept of protection domains beyond the same-origin policy and cannot differentiate legitimate JavaScript sent by the web server from XSS-injected JavaScript employed by an attacker. Many different web technologies are meant to mitigate against XSS attacks (e.g., content-security policies [52]), but for our work we consider XSS attackers to be out of scope.

Note that we do allow the attacker to eavesdrop on all traffic, act as a man-in-the-middle on all traffic, and we allow the attacker to present forged TLS server certificates, impersonating a legitimate server to any client, although we generally assume that such an attacker is not present for the *initial* connection between a browser and the IDP. (In this way, our threat model resembles that of the ssh secure shell, where the ssh client will store each server's public key material before facing possible attackers.)

This work focuses on the TLS and application layers of the protocol stack. However, as the security of browser-based federated login is contingent on the security of the user's HTTP session cookie and password authentication credentials, we therefore assume that techniques similar to TLS-OBC channel bound cookies [21] and opportunistic cryptographic identity assertions [20] are used to secure the initial authentication requests leading up to the user's federated login request. This means that we assume that any authentication between the client and IDP leading up to the federated login transaction has been free from MITM attackers and therefore any established key material between the client and IDP represents the *correct* keys for the secure channel between those two principals.

III. Design

Persona's approach to strengthening federated login meshes with several other recently developed techniques for strengthening the initial and session phases of login against MITM attackers. In this section we will discuss how Persona can be extended and augmented to make use of these techniques in order to defeat TLS MITM attackers.

A. *Persona-OBC-Local*

Our first approach, called Persona-OBC-Local, aims to harden Persona against MITM attackers while still preserving Persona's goal of preventing the IDP from tracking the user's authentications to RPs. The high level approach is to use PhoneAuth and Origin Bound Certificates to establish an MITM-free channel between the user's browser and the IDP that the IDP can verify by comparing the observed OBC with the set of known OBCs for this user (i.e., verifying that the *same* browser associated with the given user is on the other end of a TLS channel). With this mutually authenticated TLS channel, the IDP can then endorse a device key, as in the existing Persona protocol. The user's browser can then present an identity assertion, signed with the device key, that includes a new OBC for use *only with the RP*.

Design. Our design goals for Persona-OBC-Local naturally fall out of this high level design. First, we want Persona-OBC-Local to feel like Persona. This means that the user experience should be entirely unchanged. There should also still be a device key, controlled by the browser, that signs new identity assertions. The device key is critical to the current operation of Persona as it cuts the IDP out of the loop during the user's logins to their RPs and protects the user's privacy; the IDP never learns the RPs to which the user connects.

At this point our goals diverge somewhat from present-day Persona. Ideally, we'd like to prevent a MITM attacker from stealing the identity and user certificates and later replaying them to the RP to login as the victim. We'd also like to secure subsequent connections between the user and the RP by presenting the RP with an OBC key that it can use to channel bind cookies to the user's browser. In our approach:

- The IDP learns the user's identity certificate, but not the OBC key used between the browser and the RP.
- Each RP receives a *unique* OBC key, signed by the user's device key. This enables RPs to detect a TLS MITM attacker, which cannot forge signatures by demonstrating ownership of these keys.
- This per-RP OBC key can be used by the RP to channel-bind authentication cookies, making them useless to a third party who can intercept them [21].
- We can also leverage the OBC between the user and the IDP to prevent an attacker from impersonating the user to get the IDP to endorse a false identity certificate, and thus allow the attacker to impersonate the user.

In order to reason about this extension to the Persona protocol, we formalized the Persona-OBC-Local protocol into security protocol notation as shown in Figure 3. The differences from the existing Persona design are evident in messages (5) and (7), message (6) is unchanged from the existing

$$\begin{aligned}
B \rightarrow I &: \{\text{Email}_{user}, \{K_{BI}, K_B\}_{K_B^{-1}}\}_{K_{BI}^{-1}} && (5)\\
I \rightarrow B &: \{\text{Email}_{user}, I, T_I, K_B\}_{K_I^{-1}} && (6)\\
B \rightarrow R &: \{\{R, T_B, K_{BR}\}_{K_B^{-1}}, \{\text{Email}_{user}, I, T_I, K_B\}_{K_I^{-1}}\}_{K_{BR}^{-1}} && (7)
\end{aligned}$$

Fig. 3: Persona-OBC-Local Protocol

Persona protocol in Figure 2. We also elide message (4) from Figure 2 as the communication of the IDP's public key to the RP is unchanged.

There are two major changes to message (5) relative to the original (1). First, the entire message is signed by K_{BI}^{-1}. We assume that K_{BI} is the public half of an apriori established OBC key pair, and that message (5) is presented on the TLS channel established using K_{BI}. This demonstrates that the browser is in possession of the private key K_{BI}^{-1} and we can therefore assume that any messages presented over that TLS channel originated from the owner of K_{BI}^{-1} and cannot be forged, even by a TLS MITM.

The second change to message (5) is the inclusion of $\{K_{BI}, K_B\}_{K_B^{-1}}$. K_B and K_B^{-1} are the public and private keys, respectively, for the browser-generated device key. This new element is therefore a signature binding the public device key to the public OBC key. The message is effectively signed twice (once with the device key and once with the OBC key), creating a cross-certification of sorts, that demonstrates to the IDP that the two private keys are present on the same browser. This ultimately gives the RP a stronger assurance that the IDP-endorsed device key K_B truly belongs to the correct user and was not compromised by a TLS MITM.

The final change required for Persona-OBC-Local is the inclusion of the K_{BR} element in the user certificate of message (7) and the communication of the identity assertion over the a TLS-OBC channel that proves the browser's ownership of K_{BR}^{-1}. The RP therefore expects a user certificate, signed by the Persona device key K_B that includes the OBC public key K_{BR}. The RP will reject any mismatching keys, defeating TLS MITM attacks. We note that, while Persona-OBC-Local assumes a pre-existing relationship between the browser and the IDP, no such relationship is assumed between the browser and the RP. Instead, we leverage the Persona key to endorse the fresh OBC key, K_{BR}.

Security claims. These modifications to the Persona protocol are meant to support two new security claims: a MITM attacker cannot influence the establishment of the device key K_B or the origin bound certificate key K_{BR} used between the browser and the RP. To address how Persona-OBC-Local enables these security claims, we'll first consider how an attacker might attempt to disrupt the protocol then formalize the protocol to prove the security claims.

Attacker models against Persona-OBC-Local. There are three locations for MITM attack in this protocol. The first, the connection between the RP and IDP has been well explored and we as well as the Persona authors [1] believe that this risk can be mitigated through the use of certificate pinning [41] as there are relatively few large IDPs so requiring RPs to maintain a list of certificate pins for the IDPs would not be difficult.

The more attractive connections for the attacker to intercept are the ones between the browser and the IDP or RP. As we assume that the IDP and browser have apriori established a known good TLS-OBC key, the IDP should be able to detect the presence of a MITM attacker using the techniques previously developed for TLS-OBC [21].

We need now focus on the final exchange of the identity assertion between the browser and RP. In the original Persona, this identity assertion is essentially a short lived bearer token. Persona-OBC-Local's modifications to message (7) make use of the Persona device key, that already exists in present day Persona, to communicate a new browser-controlled OBC public key to the RP. The resulting identity assertion must therefore be presented over a TLS channel with the *same* public key as is included in the identity assertion. This means that a MITM attacker that observes the unencrypted identity assertion cannot export and replay the identity assertion on a different TLS channel to impersonate the user. Therefore an attacker who cannot establish a TLS channel to the relying party using private key K_{BR}^{-1} cannot make use of this identity assertion.

BAN-Logic formalization of Persona-OBC-Local. In addition to reasoning about attackers abstractly, we also formalized the initial assumptions and security claims of Persona-OBC-Local into the BAN belief logic [13]. Later, in section IV, we will translate this BAN logic formalization of Persona-OBC-Local into Nexus Authorization Logic (NAL) and use the NAL proofchecker to reason about the correctness of the security claims presented in this section. The BAN logic formalization presented here will help us to understand the protocol as well as to reason about how the security claims are arrived at by deriving the security goals from the initial assumptions before moving to the more complicated NAL.

In this section I'll use BAN logic notation to express the beliefs of each principal in the protocol. A statement that $A |\equiv X$ simply means that A *believes* the statement X. A statement of the form $A \Rightarrow X$ means A has the *authority* to make statements about X, *e.g.,* some principal that believes $A \Rightarrow X$ will trust the statement X if it was asserted by A.

We arrive at the initial assumptions each principal believes before a run of Persona-OBC-Local begins by first idealizing the Persona-OBC-Local protocol as shown in Figure 4.

$$\begin{aligned}
B \rightarrow I &: \{\xrightarrow{K_B} U\}_{K_{BI}^{-1}}\\
I \rightarrow B &: \{T_I, \xrightarrow{K_B} U\}_{K_I^{-1}}\\
B \rightarrow R &: \{\{T_B, \xrightarrow{K_{BR}} U\}_{K_B^{-1}}, \{T_I, \xrightarrow{K_B} U\}_{K_I^{-1}}\}_{K_{BR}^{-1}}
\end{aligned}$$

Fig. 4: Idealized Persona-OBC-Local Protocol

The idealization of the first message replaces the user's E-Mail address at the IDP, Email_{user}, with the principal U that represents the user interacting with the browser during the login process. The statement $\xrightarrow{K_B} U$ included in this and subsequent messages is a BAN logic shorthand expressing that the key K_B *speaks for* the principal U.

The second message once again converts the Email_{user} element into the principal U that represents the user operating the browser (as identified by E-Mail address Email_{user} at the IDP). The I element is also removed as it only serves to identify the particular IDP that minted this user certificate and in the initial assumptions below we assume that the RP has already obtained and cached the public key K_I used by I. Therefore the inclusion of the I element of message two is redundant in this idealized protocol.

In the final message, the principal names I and R are removed from this idealized version as they only serve to clutter the messages[1]. One potentially confusing bit of this idealization is that the $\{R, T_B, K_{BR}\}_{K_B^{-1}}$ element is converted to $\{T_B, \xrightarrow{K_{BR}} U\}_{K_B^{-1}}$ when the original message doesn't contain the user's email address Email_{user}. This conversion makes more sense when considering the initial assumptions from figure 5 where:

$$R| \equiv I \Rightarrow \xrightarrow{K_B} U$$

meaning R *believes* I *has authority* to claim that K_B *speaks for* U. This means that the only method for R to establish trust in the key K_B is through the satisfaction of this initial assumption. Put another way, before R can verify that any incoming messages originated from U it must first establish belief in an assertion by I claiming that the key K_B *speaks for* U. Therefore the message $\{R, T_B, K_{BR}\}_{K_B^{-1}}$ is only meaningful to R after it verifies an IDP signed user certificate and establishes that R *believes* K_B *speaks for* U, at which point R will associate any subsequent messages signed by K_B^{-1} with the user principal U.

The initial assumptions we'll be using are shown in Figure 5. As in the earlier discussion of Persona, B, I, and R are the principals representing the browser, IDP, and RP respectively. U is a principal named by the user's email address at the IDP but represents the real world user that is using the browser as a proxy for communication. Finally, T_I and T_B are timestamps that are used to determine the validity period for the messages they are included in, and are converted to freshness assumptions (using the BAN logic notation where $\#(X)$ means X is fresh) in the initial assumptions.

We assume the IDP has established an OBC, K_{BI}, for communication with the user's browser. The IDP therefore believes that the browser principal B has the authority to speak for the user's persona device key K_B. Since B and I will be communicating via a TLS-OBC channel, I gets a freshness guarantee for messages sent over the channel as the TLS protocol prevents replay attacks on the established channel and OBC prevents a MITM attacker from replaying a message on a different TLS channel of her own choosing.

For the RP, we assume the RP has already established that the IDP will communicate with the key pair defined by the public key K_I. Additionally, the RP believes that the IDP is an authority for the user's Persona device key K_B and that the device key K_B, speaking for the user U, is an authority on the

[1]Here I'm assuming that a signature with some private key K_A^{-1} also identifies the signer as principal A

$$\begin{aligned}
I &\;|\equiv\; \xrightarrow{K_{BI}} B \\
I &\;|\equiv\; B \Rightarrow \xrightarrow{K_B} U \\
I &\;|\equiv\; \#(\xrightarrow{K_B} U) \\
R &\;|\equiv\; \xrightarrow{K_I} I \\
R &\;|\equiv\; I \Rightarrow \xrightarrow{K_B} U \\
R &\;|\equiv\; U \Rightarrow \xrightarrow{K_{BR}} U \\
R &\;|\equiv\; \#(T_I) \\
R &\;|\equiv\; \#(T_B)
\end{aligned}$$

Fig. 5: Persona-OBC-Local initial assumptions

OBC key K_{BR}. Finally, the RP believes that the timestamps in each message are fresh.

Defining the Persona-OBC-Local security goals. There are many possible conclusions that we could derive from the application of the BAN logic inference rules to the initial assumptions in figure 5. When analyzing existing authentication protocols like Kerberos [53] the usual goal is to reach the following conclusions:

$$A| \equiv A \overset{K_{ab}}{\longleftrightarrow} B \quad B| \equiv A \overset{K_{ab}}{\longleftrightarrow} B$$
$$A| \equiv B| \equiv A \overset{K_{ab}}{\longleftrightarrow} B \quad B| \equiv A| \equiv A \overset{K_{ab}}{\longleftrightarrow} B$$

meaning that A and B both *believe* the key K_{ab} and each *believes* the other *believes* the key K_{ab}. These conclusions can be broken into two parts. First:

$$A| \equiv A \overset{K_{ab}}{\longleftrightarrow} B \quad B| \equiv A \overset{K_{ab}}{\longleftrightarrow} B$$

meaning that the principals A and B both believe that they should use the key K_{ab} for communication with the other. This establishment of a key for communication is the most basic requirement that must be satisfied for authentication to occur. The second part:

$$A| \equiv B| \equiv A \overset{K_{ab}}{\longleftrightarrow} B \quad B| \equiv A| \equiv A \overset{K_{ab}}{\longleftrightarrow} B$$

establishes that each principal *believes* the other will be using K_{ab} for communication with the other party *e.g.,* A will use K_{ab} for communication with B and A believes that messages encrypted with K_{ab} originated from B.

The security goals for Persona-OBC-Local are similar to those of Kerberos. At the end of a run of Persona-OBC-Local we would like to know the following:

$$R \;|\equiv\; \xrightarrow{K_{BR}} U \tag{8}$$
$$R \;|\equiv\; U \;|\equiv\; \xrightarrow{K_{BR}} U \tag{9}$$
$$R \;|\equiv\; I \;|\equiv\; \xrightarrow{K_B} U \tag{10}$$

The first goal captures the end result of a run of the Persona-OBC-Local protocol where the relying party believes that the TLS-OBC channel defined by K_{BR}, and used for communication from the user's browser to the RP, speaks on behalf of the user U that is identified by Email_{user} at the IDP.

The next two goals are beliefs that fall out of the derivation from initial assumptions to the first goal as shown in figure 12. The goal in equation (10) establishes that RP believes that the IDP has vouched that the user key K_B speaks for the user. This is a necessary belief as the RP trusts **only** the IDP to speak for the user's identity and device key. Note that because the RP believes that the IDP has sole authority over $\xrightarrow{K_B} U$, it's entirely possible for an IDP to act as the user U during the Persona-OBC-Local protocol. The prevention of potential IDP impersonation of the user is a non-goal for Persona-OBC-Local as any federated login protocol necessitates that the IDP to be trusted to speak for the user's identity.

The goal in equation (9) establishes that the RP believes that U believes in key K_{BR}. This belief allows the RP to believe that both U and R have a common belief of correctness in K_{BR}. It's important to note that the beliefs:

$$U \quad |\equiv \quad \xrightarrow{K_{BR}} U \tag{11}$$

$$U \quad |\equiv \quad R|\equiv \xrightarrow{K_{BR}} U \tag{12}$$

are not listed in these final security goals for Persona-OBC-Local. We exclude equation (11) from the final security goals as the user's browser invents K_{BR} locally and therefore implicitly believes that K_{BR} *speaks for* U. Likewise, equation (12) is implicitly true upon a successful run of Persona-OBC-Local when R logs the user in and grants her access to her data stored at R. This goal is also not all that useful here as R will not be using key K_{BR} to communicate with U, but rather a different public key (or TLS session key) that establishes the authenticity of the RP during the TLS handshake, using traditional means like a certificate signed by a trusted certificate authority.

We chose to use these security goals as they capture what should be the end result of a federated login transaction between the user (and their browser), the IDP, and a RP. The first requirement upon the successful completion of federated login is that the RP can establish the user's identity at the IDP. In Persona-OBC-Local, this requirement is captured by the presentation of the user certificate $\xrightarrow{K_B} U$ that is signed by the IDP I. This user certificate establishes both user's email, Email_{user}, at the IDP as well as a key that speaks for the user, K_B, and *must* be signed by the IDP's private key so the RP can verify that the user certificate was issued by the IDP. The verification of the user certificate is shown in equation (18) of figure 12.

The second requirement that we added in the design of Persona-OBC-Local is that the RP *also* obtains a secure channel between itself and the user. Note that the Persona device key K_B could serve this purpose as R also establishes that it speaks for the user U. However there are implementation issues with this approach, *e.g.,* using K_B requires the use of a javascript cryptography library, as well as privacy issues, *e.g.,* the IDP knows K_B uniquely identifies the user and could track which RPs the user logs into. This requirement is captured by the main security goal of Persona-OBC-Local as shown in equation (8). This establishment of a secure channel between the user and the RP is a non-goal for existing federated protocols, but the establishment of this distinct channel per user is good practice and enables better security for both the user and RP.

The RP arrives at these final security goal by first establishing the validity of the IDP signed user certificate, establishing the belief $R|\equiv \xrightarrow{K_B} U$. This belief that K_B is a key that speaks for U allows the RP believe that $U|\equiv \xrightarrow{K_{BR}} U$ as shown in the second to last equation of figure 12. As an initial assumption held by R is that $U \Rightarrow \xrightarrow{K_{BR}} U$, this allows R to establish its belief that K_{BR} is a key that has been endorsed by the user U. This means that subsequent requests sent over the TLS-OBC channel defined by K_{BR} can also be associated with the user principal U.

Figure 12 in the appendix shows the derivations from these initial assumptions and observed messages to the security goals we've set out for the protocol.

Discussion. The design of Persona-OBC-Local is intended to be very similar to Persona as it exists today. These changes to the protocol preserve user privacy by keeping the IDP out of the federated login process after the initial endorsement of the device key and identity certificate. Even though the IDP is not an online actor during the federated login process, the RP still receives the IDP-endorsed identity certificate and uses that endorsement to establish the RP specific OBC key.

It's important to point out that the MITM protection between browser and RP is contingent on the OBC key, endorsed by the device key, being delivered to the RP over a TLS channel established with the *same* client side OBC key that's endorsed by the IDP. It is the potential mismatch of these keys that allows the RP to detect the presence of a MITM attacker.

B. Persona-OBC-Central

Our second approach, called Persona-OBC-Central, discards the Persona ideal of disallowing the IDP from tracking user logins and presents a protocol that more closely captures existing federated login protocols, like OAuth2, OpenID and OpenID Connect, while still providing the same channel-binding MITM protections we introduced in Persona-OBC-Local. This approach requires the IDP to be an online actor during the user's initial login to a RP. While this has a negative impact on user privacy with respect to the IDP, it enables the IDP to apply a variety of anti-fraud techniques, such as machine learning to distinguish the behaviors of bad actors, tracking of malware-infected IP addresses, and so forth. While the application of these techniques is beyond the scope of the paper, we suggest that many users would find the tradeoff of decreased privacy in favor of better protection from adversaries to be worthwhile. In short, Persona-OBC-Central creates the possibility for rapid server-side responses to systematic attacks, without waiting for the 24-hour expiration in Personal-OBC-Local.

$$\begin{aligned}
B \rightarrow I &: \{\text{Email}_{user}, K_{BR}, \{K_{BI}, K_{BR}\}_{K_{BR}^{-1}}\}_{K_{BI}^{-1}} \\
I \rightarrow B &: \{\text{Email}_{user}, I, T_I, K_{BR}\}_{K_I^{-1}} \\
B \rightarrow R &: \{R, T_B, \{\text{Email}_{user}, I, T_I, K_{BR}\}_{K_I^{-1}}\}_{K_{BR}^{-1}}
\end{aligned}$$

Fig. 6: Persona-OBC-Central Protocol

The protocol description of Persona-OBC-Central in Figure 6 is similar to Persona-OBC-Local, but the first message between the browser and IDP is modified. Rather than establishing a device key, as in Persona or Persona-OBC-Local, this first message establishes that the browser controls two TLS-OBC keys, one for use with the IDP and one for use with the RP. This modification does away with the need for a 24 hour device key; instead we use the cross-certification of the two OBCs to replace this functionality. This change brings the IDP online for every RP to which the browser wishes to connect (thus, the "central" name).

$$\begin{aligned}
B \rightarrow I &: \{\xrightarrow{K_{BR}} U\}_{K_{BI}^{-1}} \\
I \rightarrow B &: \{T_I, \xrightarrow{K_{BR}} U\}_{K_I^{-1}} \\
B \rightarrow R &: \{T_B, \{T_I, \xrightarrow{K_{BR}} U\}_{K_I^{-1}}\}_{K_{BR}^{-1}}
\end{aligned}$$

Fig. 7: Idealized Persona-OBC-Central Protocol.

As in Persona-OBC-Local, we idealized the Persona-OBC-Central protocol, as shown in Figure 7, established the initial assumptions of the protocol, and created a proof that derives our goal state from the initial assumptions. The full analysis is available in our forthcoming tech report.

The protocol is very similar to Persona-OBC-Local, the main difference being that there is no device key included in the identity certificate. Rather the IDP endorses the relying party's OBC key directly in the identity certificate instead of delegating the endorsement of this key to the browser as in Persona-OBC-Local.

Discussion. The obvious tradeoff between the Local and Central protocol implementations is the ability for the IDP to track where the user logs in. Since the Central approach does away with the 24 hour device key and browser endorsement of the OBC key in favor of having the IDP endorse the RP's OBC key directly, the IDP has the capability to track the user as they login to different relying parties.

The potential benefits of the Central approach over the Local approach are:

- The IDP can use machine learning techniques to correlate authentication requests and potential identify when a user is under attack. The centralized approach allows the IDP to respond to attacks quickly by disabling the creation of new identity certificates if a user's account is assumed to be under attack or compromised.
- The relying parties do not learn the 24 hour device key (which is a unique identifier for the user). This allows for a user to present IDP endorsed pseudonymous usernames to each RP they login to and prevent colluding RPs from tracking the user.

IV. Formalization of Protocols

There are a number of ways to model and evaluate protocols like Persona-OBC. The Border Gateway Protocol [45] and Byzantine fault tolerant systems [18] are specified by finite state machines that, when the software correctly implements the state transitions, allows us to reason about the state of each actor in the system. Formal verification tools like SPIN [12] can then be used to reason about the protocol state machine and prove that the protocol cannot enter an incorrect state. The reduction to a finite state machine is useful when each principal in the system is running the same state machine and performing the same transitions, but what about in protocols and systems where each principal is operating independently and with imperfect knowledge? These systems are more easily modeled by a belief logic.

Some of the earliest logics used for reasoning about belief in distributed protocols are the ABLP logic introduced in the TAOS [55] operating system, BAN logic [13], and Gong *et al.*'s work [25].

Each of these logics build from the notion of an atomic principal that is represented by a public key or messages sent over a channel associated with a public key. From a set of initial assumptions that represent the beliefs of each of the principals in the system, axioms are applied to the initial beliefs to derive a goal state that must be satisfied before a principal may take an action.

The Nexus Authentication Logic [48], a successor to the ABLP [55] and CCS [5] logics, further refines this idea by defining a world view for each principal in the system that is defined by the set of beliefs local to that principal. This modification has the desired properties that a fallacious belief held by one principal does not propagate into the world view of another, making it well suited for modeling distributed systems with multiple actors. Additionally, the Nexus operating system built on top of the NAL contains a stack machine-based proof checker that can be used to verify proofs presented by a principal. This proof checker enables us to use NAL proofs in our implementation to verify its correctness (see Section V) similar to the approach in Bauer's [11] proof carrying authentication.

Nexus Authorization Logic semantics and axioms. A NAL proof is based on a few core concepts: principal, statements, world views, and guards.

- A principal is an entity that may take or authorize actions. Principals are usually defined by the public half of an asymmetric key pair or the hash of a program. For our system, the communicating public keys serve as names for their principals.
- A statement is a message or utterance emitted by a principal. A statement may be implicit (i.e., it's only used as an intermediate step to establish local belief) or explicit (i.e., where the statement is passed in a message to another principal). The statement that A **says** s implies that A supports the statement s. For the purposes of this section, we assume that the statement A **says** s implies that s was received over a mutually authenticated TLS channel associated with A or a signature over s with a fresh nonce has been verified by the receiving principal.
- The set of statements uttered by a principal defines its belief set or world view. Therefore a statement that A **says** s adds the statement s to the belief set of A. This world view is represented by the statement $s \in \omega(A)$. It is important to

note that each principal begins with an empty belief set, and adds to it over time. This means that an incorrect belief by principal A cannot effect principal B unless B is willing to explicitly add the incorrect belief into it's belief set.

- A guard is a statement that protects a privileged action. For instance, a file system guard that protects access to privileged files might require a guard of $G = FileSys$ **says** read(foo) to be satisfied before allowing access to the file foo. The principal A that protects foo would therefore require that the guard statement G be derivable from the current set of statements in the belief set $\omega(A)$. The derivation of statements in $\omega(A)$ to the guard statement G are defined by the axiomatic reductions defined in NAL as well as any reductions that already exist in $\omega(A)$.
- Another useful construct in NAL is the *speaks for* relationship, represented by the $\rightarrow$ symbol. The statement that $A \rightarrow B$ is shorthand for $(\forall x : (A \textbf{ says } x) \Rightarrow (B \textbf{ says } x))$ this has the side effect of unifying the principals worldviews' such that $\omega(A) \subseteq \omega(B)$.

A. NAL Formalizations

Each of the NAL formulas to follow begins with a set of initial beliefs that the principal either already believes or has received from an untrusted source and must apply NAL derivations to enter the untrusted beliefs into the principals local world view. Finally, each formula set defines a *goal* statement that must be derived from the initial assumptions for the guard proof to be satisfied. These proofs are human readable representations of the proofs that we used with the Nexus mechanized proof checker, the mechanized proofs will be included in a forthcoming technical report.

Our use of NAL also makes use of some axiomatic rules from BAN-Logic. Specifically, we adapt the message meaning, nonce verification, freshness, and jurisdiction rules from BAN-Logic to NAL as shown in Figure 8.

$$\frac{K_A \textbf{ says } X \qquad B \textbf{ says } K_A \rightarrow A}{A \textbf{ says } X} \quad \text{Message Meaning}$$

$$\frac{A \textbf{ says } \{X, Y\} \qquad A \textbf{ says } \text{fresh}(X)}{A \textbf{ says } \text{fresh}(X, Y)} \quad \text{Freshness}$$

$$\frac{A \textbf{ says } X \qquad B \textbf{ says } \text{fresh}(X)}{B \textbf{ says } A \textbf{ says } X} \quad \text{Nonce Verification}$$

$$\frac{B \textbf{ says } A \textbf{ says } X \qquad \dfrac{B \textbf{ says } A \textbf{ says } X}{B \textbf{ says } X}}{B \textbf{ says } X} \quad \text{Jurisdiction}$$

Fig. 8: NAL version of BAN-Logic rules for Message Meaning, Nonce Verification, and Jurisdiction.

The first proof we'll consider is that of the identity provider attempting to guard the generation of an identity certificate. Our initial assumptions, shown in Figure 9, are that the browser and IDP have apriori established that the browser controls K_{BI} that it uses for communication with the IDP. We also assume that the browser has generated a device key pair K_B for the user and that the IDP believes that the browser is an authority on $K_B \rightarrow U$.

$$\begin{array}{rcl} I & \textbf{says} & K_{BI} \rightarrow B \\ I & \textbf{says} & \text{fresh}(T_I) \\ K_{BI} & \textbf{says} & \{K_B \rightarrow U, T_I\} \end{array}$$

$$\frac{I \textbf{ says } B \textbf{ says } K_B \rightarrow U}{I \textbf{ says } K_B \rightarrow U}$$

Fig. 9: NAL Initial Assumptions for Identity Provider.

$$\frac{K_{BI} \textbf{ says } \{K_B \rightarrow U, T_I\} \qquad I \textbf{ says } K_{BI} \rightarrow B}{B \textbf{ says } \{K_B \rightarrow U, T_I\}}$$

$$\frac{K_{BI} \textbf{ says } \{K_B \rightarrow U, T_I\} \qquad I \textbf{ says } \text{fresh}(T_I)}{I \textbf{ says } \text{fresh}(K_B \rightarrow U)}$$

$$\frac{B \textbf{ says } K_B \rightarrow U \qquad I \textbf{ says } \text{fresh}(K_B \rightarrow U)}{I \textbf{ says } B \textbf{ says } K_B \rightarrow U}$$

$$\frac{I \textbf{ says } B \textbf{ says } K_B \rightarrow U \qquad \dfrac{I \textbf{ says } B \textbf{ says } K_B \rightarrow U}{I \textbf{ says } K_B \rightarrow U}}{I \textbf{ says } K_B \rightarrow U}$$

Fig. 10: Guard proof derivation for Persona-OBC-Local IDP

Figure 10 shows the IDP's guard proof derivations that result in the satisfaction of the goal statement of:

$$I \textbf{ says } K_B \rightarrow U$$

This goal protects the creation of an identity certificate by the IDP and its satisfaction therefore allows for the creation of an assertion that the user is logged in at the IDP. We additionally want the IDP to believe that the key K_B is also in the possession of user's browser as the IDP will be including a statement that $K_B \rightarrow U$ in the identity certificate it generates and signs. The satisfaction of the guard proof will result in the IDP creating an identity assertion of the form $\{U, K_B, T_I\}_{K_I^{-1}}$ that allows it to assert its local beliefs to any recipient that believes $K_I \rightarrow I$.

$$\begin{array}{rcl} R & \textbf{says} & K_I \rightarrow I \\ R & \textbf{says} & \text{fresh}(T_I) \\ R & \textbf{says} & \text{fresh}(T_B) \\ K_I & \textbf{says} & \{K_B \rightarrow U, T_I\} \\ K_B & \textbf{says} & \{K_{BR} \rightarrow U, T_B\} \\ K_{BR} & \textbf{says} & \{O_R\} \end{array}$$

$$\frac{R \textbf{ says } I \textbf{ says } K_B \rightarrow U}{R \textbf{ says } K_B \rightarrow U}$$

$$\frac{R \textbf{ says } U \textbf{ says } K_{BR} \rightarrow U}{R \textbf{ says } K_{BR} \rightarrow U}$$

Fig. 11: Initial assumptions for identity assertion verification

The second proof we'll consider is the RP attempting to check the validity of the browser generated user certificate and an IDP's identity assertion that has been relayed through the user's browser. Our initial assumptions – shown in Figure 11 are that the RP knows the IDP's public key K_I, the RP believes the IDP is an authority on U and $K_B \rightarrow U$, and that the user's browser is an authority on $K_{BR} \rightarrow U$.

Figure 13, in the appendix, shows the guard proof derivations that result in the derivation of the goal statement of:

$$R \textbf{ says } K_{BR} \rightarrow U \land U \textbf{ says } O_R$$

This goal statement is equivalent to the BAN logic goal in equation (8) in section III. The goals from equation (9) and (10) are excluded here they are arrived at and placed onto the stack during the proof derivation that leads to the goal state and every belief placed on the stack during the derivation of the goal proof is present in R's belief set.

This proof allows the RP to believe that it is communicating with the a browser controlled by the user. The RP arrives at the goal state by transforming its initial assumptions into a local belief that the user's browser controls the key K_B asserted by the IDP. This local belief that $K_B \rightarrow U$ allows the RP to believe that the user's browser also controls K_{BR}. Finally, the presentation of these messages over the TLS-OBC channel associated with K_{BR} allows the RP to verify that it is communicating with the correct browser[2]. Since the RP also implicitly trusts the IDP to speak about the logged in user's identity U, the satisfaction of the guard proof will allow the RP to associate the TLS channel defined by the TLS-OBC K_{BR} to the user identity U or, optionally, mint a channel-bound cookie that is bound to the key K_{BR}.

V. Implementation

In order to demonstrate the feasibility of our approach, we built a proof of concept implementation of Persona-OBC-Central. The goal of this implementation is to help establish the correctness of the design previously discussed as well as to identify the engineering (on both the browser and server) required to put our design into practice. The implementation consists of four separate elements:

- A Linux port of the Nexus [50] OS proof checker
- A browser extension that provides a virtual TPM and cryptographic operations
- A identity provider server/client side implementation
- A relying party server/client side implementation

A. Nexus Proof Checker

We use a Linux port of the Nexus proof checker to verify the satisfaction of our guard proofs from the initial assumptions of the IDP and RP server side implementations. We also created a python wrapper that handles the setup of the proofs by managing the mapping of principal names to public key fingerprints and verification of message signatures. This python wrapper consists of approximately 203 lines of code and interacts with the Nexus proof checker binary using the python subprocess package.

The proofs used with the Nexus proof checker were first written by hand with the help of the interactive mode of the Nexus proof checker, then formalized in the proof checker's language. This language allows us to define the initial assumptions as well as the derivations applied to the stack of assumptions that eventually result, in a successful run of the proof, with only the goal statement left on the proof checker's stack.

B. Virtual TPM

As the only web server that currently supports TLS-OBC (or rather its successor project called TLS-ChannelID [9]) is Google's internal web server, we chose instead to extend the browser to support something that looks and feels like TLS-OBC without actually hooking into the TLS layer. We developed an API called PostKey that would ideally be linked to the `navigator.id` namespace within the browser. This PostKey API is similar to the PostMessage API that allows for cross origin communication between iframes with the addition of a browser-generated cross certification of TLS-OBC keys.

We modified the HTML5-Crypto-API [27] in order to support this new PostKey operation and additionally expose a NodeJS wrapper around the cryptographic operations provided by the API to server side code. The HTML5-Crypto-API works well for our needs since, like TLS-OBC, it supports the concept of performing crypto operations where the actual keys are never exposed to javascript code, rather they are stored by the HTML5-Crypto-API and javascript performs actions with the stored keys via PostMessage calls, lessening the risk of their exfiltration. Likewise, the HTML5-Crypto-API supports the idea of crypto keys visibility being linked to the web same-origin policy. In order to retain these properties we modified the HTML5-Crypto-API to operate as a browser extension and expose PostKey API functionality.

This allows us to use the browser same origin policy and extension policies to protect private key material from XSS attacks by using browser IPC to call into the extension. The virtual TPM this we expose is conceptually similar to the separation between the TLS and HTTP application layers in that the TLS key material used to established a channel is never exposed up to the HTTP layer. The PostKey operations performed by the virtual TPM can be trivially replaced by browser-provided JavaScript bindings that perform the same operations with TLS Origin-Bound Certificates, TLS ChannelIDs, or keys created by the HTML5 keygen tag as the keying material.

C. Identity Provider Implementation

Our IDP is implemented in approximately 280 lines of client side JavaScript and 1073 lines of server side Python. Additionally, 70 lines of JavaScript are used on the server side via IPC calls from the python IDP binary to a NodeJS port of the HTML5-Crypto-API in order to unify the client and server side cryptographic libraries that handle the asymmetric elliptic curve crypto for message encryption and signing.

[2]This requirement is captured with the U **says** O_R clause in the goal statement, showing that the user's browser can sign the RP's web origin with K_{BR}. The presentation of the identity and user certificates over the TLS-OBC channel with key K_{BR} is actually sufficient to prove this but difficult to show in NAL.

Our IDP allows users to login with a traditional username/password verification, then issues a channel bound cookie [21] to the client.

After the initial login, the IDP is responsible for creating the identity certificate that we discussed in Section III. It does this by exposing an `authorize` API (similar to the authorization endpoint exposed by a web server that supports OAuth2) that expects to receive a channel bound cookie and PostKey cross certification.

Upon receipt of this data, it uses the Nexus proof checker to verify the authorization guard proof we discussed previously using the observed channel-bound cookie and the PostKey cross certification sent from the browser as well as the apriori known public key associated with the user's channel-bound cookie. Assuming that the proof checker can successfully derive the goal proof from the data received from the browser, the IDP then mints an identity certificate that is transmitted to the browser and forwarded to the relying party to complete the federated login process.

D. Relying Party Implementation

The RP is implemented in approximately 300 lines of client side JavaScript and 953 lines of server side Python. It shares the same JavaScript cryptography library as the IDP. The RP is responsible for using the Nexus proof checker to verify that the user certificate generated by the browser and the IDP generated signed identity assertion are correct. If the proof checker successfully derives the federated login goal statement from the RP's belief set, then the RP will issue a channel bound cookie to the client and complete the federated login transaction.

We chose to use the Nexus proof checker in the implementations of the RP and IDP servers in order to establish the correctness of our implementation when reasoning about the validity of data received from potentially untrusted actors. In a production implementation of the RP and IDP logic we believe that this approach would be too complex and heavyweight, but the creation of a production version of the RP and IDP would be guided by the reference implementation we've constructed.

VI. Discussion

While we've primarily discussed the Persona federated login system, there's no reason why the techniques we've presented cannot be applied to other existing federated login protocols. We will now consider two of the most prevalent federated login protocols deployed today and discuss how they too can be hardened against man in the middle attackers.

OAuth2. OAuth2 [31] uses a two phase process to handle the identity exchange between the IDP and RP. First the user logs into the IDP and their browser receives a `authorization_code` that is a one time use bearer token. This `authorization_code` is then passed via a HTTP 301 redirect to the relying party where it is transmitted to the IDP *authorization* endpoint and traded in for a short lived `access_token`, the user's identity at the IDP (in the form of an email address or username), and optionally a long lived `refresh_token`.

The techniques discussed in the previous sections can be used to harden the OAuth2 `authorization_code` exchange against man-in-the-middle attackers by encapsulating it in an IDP minted identity certificate that includes the OBC key the browser will use to transmit the `authorization_code` to the relying party. This is very similar to the Persona-OBC-Central approach that we discussed previously with the minor change that the OAuth2 `authorization_code` is included in the identity certificate, rather than the user's email address or username. An `authorization_code` delivered using this method still allows the relying party to channel bind any subsequently issued cookies or access tokens to the OBC key included in the identity certificate. This technique can be used to harden browser stored OAuth2 `access_token`s but is not useful for protecting `access_token`s intended for use between the RP and IDP servers.

OpenID. OpenID [44] providers are already aware of the risks posed by MITM attackers and bearer tokens as some make use of nonces to protect the final redirect phase of the OpenID protocol from replay attacks [54]. However, this technique assumes that the attacker obtains the OpenID redirect URL, which acts as a bearer token to log in with the user's account at the relying party, after the user has already passed it on to the relying party. An active MITM attacker can observe the redirect URL as it flows from IDP to the user's browser and disrupt the OpenID login flow, then use the stolen redirect URL to login as the victim on a browser that the attacker controls.

Our approach could prevent this type of attack. An additional `openid.rp_client_key` attribute could be added to the final authorization message of the OpenID protocol. The relying party would then be responsible for validating the `openid.rp_client_key` against the observed TLS-OBC key for the connection to the client (as discussed in sections III and IV).

Smartphones. We've discussed how to improve federated login using origin bound certificates and cross certifications to communicate key material between two parties by using the browser as an intermediary for communication. We have framed these techniques as a way to improve existing federated login protocols, however these techniques can be applied to the realm of smartphone security as well.

A fundamental difference between the browser and mobile phone environments is that applications running on a mobile phone are not bound to anything like a web browser same-origin policy. A web server that serves HTML and javascript to a browser can have a reasonable expectation that code from other domains cannot manipulate the client side javascript running in the browser. Mobile phone operating systems, like Android, were designed to facilitate inter-app communication without the need to use the network and allow applications to register for "intents" that can enable malicious apps to intercept potentially sensitive URLs and data intended for another application.

Therefore a common issue when designing applications intended for mobile phones is that a server communicating with an on-phone app has no insight into which application on the phone it is communicating with or if the data it is providing is being delivered to the correct application on the phone. It would be useful to enable the servers that provide the backend for applications like mobile banking and secure communication apps (e.g. RedPhone [3]) to know that they are communicating with the correct application on a device rather than an impostor application.

The Persona-OBC-Central protocol we've discussed can be used to distribute keys to these application servers assuming there is a trusted party, in the form of the device's manufacturer or carrier (e.g., Google or Verizon Wireless), that has a strong, mutually authentication channel to the user's device. The Google Play Store already maintains a cryptographic channel from the server to every phone as part of the application installation process. This could be extended to allow the Play Store to act as an identity provider proxy (IDP proxy) for every app on the phone, under the assumption that Google has confidence that the Play Store app on the phone can protect its local key storage and communicate with the underlying operating system to obtain the application signing key for an app that calls it.

In this scenario, any on-phone app that wishes to communicate to its home server would request that the local IDP proxy generate a new key pair for the application. The IDP proxy can then create a cross certification endorsed by its trusted key that creates a mapping between an app identifier, package name on Android or bundle id on iOS, and a local key-pair unique to that device and app. When the IDP receives a new cross certification from the IDP proxy on a users phone; the IDP can then mint an assertion that demonstrates the IDP's belief that an app with package name P, on device D will use the key K_A to communicate with it's home server. This assertion can then be used by the app's home server to bootstrap a trusted channel, using the new local public key K_A for future communication between the on-phone app and home server.

This is similar to the secure RPC method discussed in Quire [22]. While Quire was interested in having the phone attests to the on-device call chain that led up to an RPC, the design discussed here is interested only in bootstrapping a trusted channel between a single on-phone app and its home server that stores potentially sensitive information on behalf of the on-phone app.

This approach could enable better RPC security on mobile phones by using the phone's OS provider (e.g., Google, Apple) to establish a TLS-OBC like channel between and on phone app and remote server. Quire [22] contemplates a variety of applications that arise from the presence of a trusted RPC attester/mediator on a device, including payment and advertising services.

VII. Related Work

OAuth 1.0 [23] makes use of message level encryption rather than encrypting the entire communication channel as in TLS. This approach has some similarity to the approach presented in this paper as OAuth 1.0 identity assertions are not bearer tokens. Instead, OAuth 1.0 used digital signatures. While this approach is commendable and provides many of the same protections as presented in this paper, the requirement for signatures at the application layer complicated things for developers, leading to a complete redesign in OAuth2.

The SAML WS-Trust [4] specification outlines a proof key technique for establishing a secure connection between a client and relying party. The main difference between the SAML approach and our approach is that in a SAML key token request the client is usually assumed to be a server rather than a user-controlled device. With SAML, a portal server is typically responsible for the creation and managements of keys intended for consumers. SAML is also targeted at native desktop application clients and server-to-server communication rather than web browsers or smartphones.

The OAuth2.0 Holder-of-Key Token internet draft [16] proposes binding tokens to ephemeral asymmetric credentials, much like we proposed in Section III. However the Holder-of-Key Token draft is concerned with binding the tokens used for API access and does not consider the risk of MITM theft of the OAuth2.0 *access_code* that could allow a MITM to trick a relying party into minting new authentication tokens associated with keys under the control of the MITM. It's likely that OBC techniques could be applied to address these concerns for OAuth2.0 in much the same way we did for Persona.

The GSS-BrowserID internet draft [33] aims to allow standalone applications to work with the existing BrowserID protocol despite the lack of a browser or JavaScript shim support. The authors also aim to support channel binding and replay prevention techniques from SASL [38] and GSS [36] at the application layer.

Kumar *et al.* recently formalized SAML single sign-on [34] using a variant of BAN logic that takes into account the use of server-authenticated TLS channels for encryption, and discovered a vulnerability that allows an attacker to manipulate cross-domain identity associations. They also advocate for a hybrid approach to protocol analysis where a belief logic is used to formalize the protocol before exploring attacker models that aim to violate the assumptions established using the belief logic formalizations.

Bauer *et al.* [10] consider the usability side of single-sign-on (SSO) and survey user's willingness to use the SSO systems of Google, Google+, and Facebook. They find that users perceive that SSO systems only perform authentication when in fact many also authorize the IDP to communicate additional information (like a Facebook friends list) to the RP. This finding speaks to the need for secure, transparent, *authorization* systems rather than SSO techniques that blur the line between authentication and authorization.

Finally, Chew *et al.* [19] and Hackett *et al.* [29] both discuss how an ideal federated identity system (and ecosystem) should operate and provide insight into future directions for this area. We believe that our work captures some of the insights presented in these works and presents a more formal approach to the abstract methods that have been explored by these prior works.

VIII. Conclusion

In this work, we presented two new variants on the Persona federated login protocol, leveraging TLS origin bound certificates to strengthen Persona in the face of recent attacks against TLS that can extract cookies or other such tokens. One of our variants follows Persona's original structure, decoupling the identity provider from the user's third-party logins. Our other variant is closer to existing OpenID and OAuth systems, giving the identity provider a more central role in the login process.

We formalized our two Persona variants using BAN logic as well as Nexus Authorization Logic, the latter allowing us to produce mechanized proofs of our design's correctness. We also built a proof-of-concept system using browser and server-side extensions, to ensure that our system could operate properly in real browsers, and be a suitable real-world approach for federated web login.

References

[1] Browserid security review. https://wiki.mozilla.org/Security/Reviews/Identity/browserid.

[2] Phishing. https://www.owasp.org/index.php/Phishing.

[3] Redphone: Architecture overview. https://github.com/WhisperSystems/RedPhone/wiki/Architecture-Overview.

[4] Ws-trust 1.4. http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/ws-trust.html.

[5] M. Abadi, M. Burrows, B. Lampson, and G. Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 15(4):706–734, 1993.

[6] H. Adkins. An update on attempted man-in-the-middle attacks. http://googleonlinesecurity.blogspot.com/2011/08/update-on-attempted-man-in-middle.html, Aug 2011.

[7] N. J. AlFardan and K. G. Paterson. Lucky Thirteen: Breaking the TLS and DTLS record protocols. In *IEEE Symposium on Security and Privacy*, 2013.

[8] D. Balfanz. TLS Origin-Bound Certificates. http://tools.ietf.org/html/draft-balfanz-tls-obc-01, Nov 2011.

[9] D. Balfanz and R. Hamilton. Transport layer security (TLS) channel ids. http://tools.ietf.org/html/draft-balfanz-tls-channelid-01, 2013.

[10] L. Bauer, C. Bravo-Lillo, E. Fragkaki, and W. Melicher. A comparison of users' perceptions of and willingness to use google, facebook, and google+ single-sign-on functionality. In *Proceedings of the 2013 ACM workshop on Digital identity management*, pages 25–36. ACM, 2013.

[11] L. Bauer, S. Garriss, and M. K. Reiter. Distributed proving in access-control systems. In *Proceedings of the 2005 IEEE Symposium on Security & Privacy*, May 2005.

[12] B. N. Bershad, S. Savage, P. Pardyak, E. G. Sirer, M. Fiuczynski, D. Becker, S. Eggers, and C. Chambers. Extensibility, safety, and performance in the SPIN operating system. In *Proceedings of the Fifteenth ACM Symposium on Operating System Principles*, pages 251–266, Copper Mountain, Colorado, Dec. 1995.

[13] A. Bleeker and L. Meertens. A semantics for BAN logic. In *Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.

[14] J. Bonneau. Measuring password re-use empirically, 2011. http://www.lightbluetouchpaper.org/2011/02/09/measuring-password-re-use-empirically/.

[15] S. Bortnik. Inside a phishing attack: 35 credit cards in 5 hours. http://www.welivesecurity.com/2011/01/26/inside-a-phishing-attack-35-credit-cards-in-5-hours/, January 2011.

[16] J. Bradley, P. Hunt, T. Nadalin, and H. Tschofenig. The OAuth 2.0 authorization framework: Holder-of-the-key token usage. http://tools.ietf.org/html/draft-tschofenig-oauth-hotk-01, July 2012.

[17] M. Brian. Gawker media is compromised. the responsible parties reach out to TNW, 2010. http://goo.gl/0SvCj.

[18] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[19] M. Chew and S. Stamm. Contextual identity: Freedom to be all your selves. In *Proceedings of the Workshop on Web 2.0 Security & Privacy*, 2013.

[20] A. Czeskis, M. Dietz, T. Kohno, D. Wallach, and D. Balfanz. Strengthening user authentication through opportunistic cryptographic identity assertions. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 404–414. ACM, 2012.

[21] M. Dietz, A. Czeskis, D. Wallach, and D. Balfanz. Origin-bound certificates: A fresh approach to strong client authentication for the web. In *Proc. 21st USENIX Security Symposium*, 2012.

[22] M. Dietz, S. Shekhar, Y. Pisetsky, A. Shu, and D. S. Wallach. Quire: Lightweight provenance for smart phone operating systems. In *20th USENIX Security Symposium*, San Francisco, CA, Aug. 2011.

[23] E. E. Hammer-Lahav. The OAuth 1.0 protocol. http://tools.ietf.org/html/rfc5849, April 2010.

[24] S. Gaw and E. W. Felten. Password management strategies for online accounts. In *Proc. SOUPS 2006, ACM Press*, pages 44–55. ACM Press, 2006.

[25] L. Gong. *Cryptographic protocols for distributed systems*. PhD thesis, 1990.

[26] D. Goodin. Crack in Internet's foundation of trust allows HTTPS session hijacking. http://arstechnica.com/security/2012/09/crime-hijacks-https-sessions/, September 2012.

[27] Google Inc. A JavaScript/HTML5 virtual TPM. https://code.google.com/p/html5-crypto-api/, July 2013.

[28] E. Grosse. Gmail account security in Iran, 2011. http://googleonlinesecurity.blogspot.com/2011/09/gmail-account-security-in-iran.html.

[29] M. Hackett and K. Hawkey. Security, privacy and usability requirements for federated identity. In *Proceedings of the Workshop on Web 2.0 Security & Privacy*, 2012.

[30] P. Hallam-Baker. Security assertions markup language. https://www.oasis-open.org/committees/security/docs/draft-sstc-baker-saml-arch-00.pdf, May 2001.

[31] E. Hammer-Lahav, D. Recordon, and D. Hardt. The OAuth 2.0 Protocol. http://tools.ietf.org/html/draft-ietf-oauth-v2-10, 2010.

[32] E. Hjelmvik. Forensics of chinese mitm on github. http://www.netresec.com/?page=Blog&month=2013-02&post=Forensics-of-Chinese-MITM-on-GitHub, 2013.

[33] L. Howard and N. Williams. A sasl and gss-api mechanism for the browserid authentication protocol. http://tools.ietf.org/html/draft-howard-gss-browserid-00, 2013.

[34] A. Kumar. Security analysis of the identity federation transaction. In *Proceedings of the Workshop on Web 2.0 Security & Privacy*, 2013.

[35] A. Langley. Public key pinning. https://www.imperialviolet.org/2011/05/04/pinning.html, 2011.

[36] J. Linn. Generic security service application program interface version 2, update 1. http://tools.ietf.org/html/rfc2743, 200.

[37] M. Marlinspike and T. Perrin. Trust assertions for certificate keys. http://tools.ietf.org/html/draft-perrin-tls-tack-00, 2012.

[38] A. Melnikov and K. Zeilenga. Simple authentication and security layer (sasl). http://tools.ietf.org/html/rfc4422, 2006.

[39] Mozilla. BrowserID, 2012. https://developer.mozilla.org/en/BrowserID.

[40] Mozilla Foundation. Persona. https://developer.mozilla.org/en-US/docs/Mozilla/Persona, July 2013.

[41] C. Palmer and C. Evans. Certificate Pinning via HSTS, 2011. http://www.ietf.org/mail-archive/web/websec/current/msg00505.html.

[42] A. Prado, N. Harris, and Y. Gluck. Ssl, gone in 30 seconds - a breach beyond crime. In *Blackhat USA*, 2013.

[43] J. Prins. Interim report DigiNotar certificate authority breach: "Operation Black Tulip". Technical report, 2011. http://www.rijksoverheid.nl/bestanden/documenten-en-publicaties/rapporten/2011/09/05/diginotar-public-report-version-1/rapport-fox-it-operation-black-tulip-v1-0.pdf.

[44] D. Recordon and B. Fitzpatrick. OpenID authentication 1.1. http://openid.net/specs/openid-authentication-1_1.html, May 2008.

[45] Y. Rekhter and T. Li. A border gateway protocol 4 (bgp-4). 1995.

[46] J. Rizzo and T. Duong. Beast. http://vnhacker.blogspot.com/2011/09/beast.html, Sept 2011.

[47] N. Sakimura, D. Bradley, B. de Mederiso, M. Jones, and E. Jay. OpenID connect standard 1.0 - draft 07. http://openid.net/specs/openid-connect-standard-1%5F0.html, Feb 2012.

[48] F. B. Schneider, K. Walsh, and E. G. Sirer. Nexus authorization logic (NAL): Design rationale and applications. *ACM Transactions on Information and System Security (TISSEC)*, 14(1):8, 2011.

[49] P. Seybold. Sony's response to the u.s. house of representatives, 2011. http://goo.gl/YkXSv.

[50] A. Shieh, D. Williams, E. G. Sirer, and F. B. Schneider. Nexus: A new operating system for trustworthy computing. In *Proceedings of the Twentieth ACM Symposium on Operating Systems Principles*, 2005.

[51] C. M. Shields and M. M. Toussain. Subterfuge: The MITM Framework. http://subterfuge.googlecode.com/files/Subterfuge-WhitePaper.pdf, 2012.

[52] S. Stamm, B. Sterne, and G. Markham. Reining in the web with content security policy. In *Proceedings of the 19th international conference on World Wide Web (WWW '10)*, 2010.

[53] J. G. Steiner, B. C. Neuman, and J. I. Schiller. Kerberos: An authentication service for open network systems. In *USENIX Winter*, pages 191–202, 1988.

[54] E. Tsyrklevich and V. Tsyrklevich. Single sign-on for the Internet: A security story. https://www.blackhat.com/presentations/bh-usa-07/Tsyrklevich/Whitepaper/bh-usa-07-tsyrklevich-WP.pdf, 2007.

[55] E. Wobber, M. Abadi, M. Burrows, and B. Lampson. Authentication in the Taos operating system. *ACM Transactions on Computer Systems (TOCS)*, 12(1):3–32, 1994.

[56] T. D. Wu. The secure remote password protocol. In *Network and Distributed Systems Security Symposium*, pages 97–111, 1998.

Appendix

Figure 12 shows the BAN logic derivations that transform our initial assumptions and messages (Figures 5 and 4 respectively) to the security goals we've set out for Persona-OBC-Local. Recall that our goal state is:

$$R |\equiv \xrightarrow{K_{BR}} U$$

To arrive at this goal, we first consider the communication between the browser and the IDP. Equation (8) establishes that the IDP has seen a message containing the user identifier U and a new device key K_B that was sent over a channel using the apriori established TLS-OBC key K_{BI}. The verification of this information allows the derivation of equation (14) that establishes the IDP's belief that the device key K_B speaks for the user's browser. This belief allows the IDP to mint an identity certificate endorsing K_B which will eventually be communicated from the browser to the RP in equation (15). After the verification of nonce freshness, the RP can use its belief that the IDP is an authority on K_B to establish its own belief that $\xrightarrow{K_B} U$ in equation (16). Finally, the RP can use this new belief to establish its own belief that $\xrightarrow{K_{BR}} U$ in equation (18).

$$\frac{I \triangleleft \{\xrightarrow{K_B} U\}_{K_{BI}^{-1}} \qquad I|\equiv \xrightarrow{K_{BI}} B}{I|\equiv B| \sim\xrightarrow{K_B} U} \quad \text{Message Meaning} \qquad (13)$$

$$\frac{I|\equiv B| \sim\xrightarrow{K_B} U \qquad I|\equiv \#(\xrightarrow{K_B} U)}{I|\equiv B|\equiv\xrightarrow{K_B} U} \quad \text{Nonce Verification}$$

$$\frac{I|\equiv B|\equiv\xrightarrow{K_B} U \qquad I|\equiv B \Rightarrow\xrightarrow{K_B} U}{I|\equiv\xrightarrow{K_B} U} \quad \text{Jurisdiction} \qquad (14)$$

$$\frac{R \triangleleft \{T_I, \xrightarrow{K_B} U\}_{K_I^{-1}} \qquad R|\equiv\xrightarrow{K_I} I}{R|\equiv I| \sim T_I, \xrightarrow{K_B} U} \quad \text{Message Meaning} \qquad (15)$$

$$\frac{R \triangleleft \{T_I, \xrightarrow{K_B} U\}_{K_I^{-1}} \qquad R|\equiv \#(T_I)}{R|\equiv \#(\xrightarrow{K_B} U)} \quad \text{Freshness}$$

$$\frac{R|\equiv I| \sim T_I, \xrightarrow{K_B} U \qquad R|\equiv \#(\xrightarrow{K_B} U)}{R|\equiv I|\equiv\xrightarrow{K_B} U} \quad \text{Nonce Verification}$$

$$\frac{R|\equiv I|\equiv\xrightarrow{K_B} U \qquad R|\equiv I \Rightarrow\xrightarrow{K_B} U}{R|\equiv\xrightarrow{K_B} U} \quad \text{Jurisdiction} \qquad (16)$$

$$\frac{R \triangleleft \{T_B, \xrightarrow{K_{BR}} U\}_{K_B^{-1}} \qquad R|\equiv\xrightarrow{K_B} U}{R|\equiv U| \sim T_B, \xrightarrow{K_{BR}} U} \quad \text{Message Meaning} \qquad (17)$$

$$\frac{R \triangleleft \{T_B, \xrightarrow{K_{BR}} U\}_{K_B^{-1}} \qquad R|\equiv \#(T_B)}{R|\equiv \#(\xrightarrow{K_{BR}} U)} \quad \text{Freshness}$$

$$\frac{R|\equiv U| \sim T_B, \xrightarrow{K_{BR}} U \qquad R|\equiv \#(\xrightarrow{K_{BR}} U)}{R|\equiv U|\equiv\xrightarrow{K_{BR}} U} \quad \text{Nonce Verification}$$

$$\frac{R|\equiv U|\equiv\xrightarrow{K_{BR}} U \qquad R|\equiv U \Rightarrow\xrightarrow{K_{BR}} U}{R|\equiv\xrightarrow{K_{BR}} U} \quad \text{Jurisdiction} \qquad (18)$$

Fig. 12: Pesona-OBC-Local derivations from initial assumptions to security goals

$$\frac{K_I \textbf{ says } \{K_B \to U, T_I\} \qquad R \textbf{ says } K_I \to I}{I \textbf{ says } \{K_B \to U, T_I\}}$$

$$\frac{K_I \textbf{ says } \{K_B \to U, T_I\} \qquad R \textbf{ says } \text{fresh}(T_I)}{R \textbf{ says } \text{fresh}(K_B \to U)}$$

$$\frac{I \textbf{ says } K_B \to U \qquad R \textbf{ says } \text{fresh}(K_B \to U)}{R \textbf{ says } I \textbf{ says } K_B \to U}$$

$$\frac{R \textbf{ says } I \textbf{ says } K_B \to U \qquad \dfrac{R \textbf{ says } I \textbf{ says } K_B \to U}{R \textbf{ says } K_B \to U}}{R \textbf{ says } K_B \to U}$$

$$\frac{K_B \textbf{ says } \{K_{BR} \to U, T_B\} \qquad R \textbf{ says } K_B \to U}{U \textbf{ says } \{K_{BR} \to U, T_B\}}$$

$$\frac{U \textbf{ says } \{K_{BR} \to U, T_B\} \qquad R \textbf{ says } \text{fresh}(T_B)}{R \textbf{ says } U \textbf{ says } K_{BR} \to U}$$

$$\frac{R \textbf{ says } U \textbf{ says } K_{BR} \to U \qquad \dfrac{R \textbf{ says } U \textbf{ says } K_{BR} \to U}{R \textbf{ says } K_{BR} \to U}}{R \textbf{ says } K_{BR} \to U}$$

$$\frac{K_{BR} \textbf{ says } O_R \qquad R \textbf{ says } K_{BR} \to U}{U \textbf{ says } O_R}$$

Fig. 13: Guard proof derivation for identity assertion verification.