Save The Implicit Flow? Enabling Privacy-Preserving RP Authentication in OpenID Connect

Maximilian Kroschewski
Hasso Plattner Institute, University of Potsdam
maximilian.kroschewski@hpi.de

Anja Lehmann
Hasso Plattner Institute, University of Potsdam
anja.lehmann@hpi.de

ABSTRACT

OpenID Connect (OIDC) is a Single Sign-On (SSO) protocol that allows users to authenticate to various Relying Parties (RPs) via an Identity Provider (IdP). The main drawback of SSO is its lack of privacy, as the IdP learns the RP's identity at each user's login. OIDC supports several protocol flows, of which only one, the Implicit Flow, gives hope for any privacy, as it does not require direct communication between the IdP and RP. This design was initially intended for RPs with technical limitations that prevent them from storing credentials and thus authenticating to the IdP. However, RP authentication is crucial to ensure that users only access properly registered RPs. As a result, the Implicit Flow is being discussed to be excluded from the OAuth specification on which OIDC is based.

This paper demonstrates a privacy-preserving approach incorporating RP authentication into the Implicit Flow. The IdP can restrict its service to authenticated RPs and tie each authentication token to a specific user and RP without acquiring knowledge of which user is accessing which RP. We formally define the desired security and privacy properties of such an authenticated Implicit Flow, propose a provably secure construction from generic building blocks, and report on an implementation of our scheme.

KEYWORDS

single sign-on, openid connect, privacy, authentication

1 INTRODUCTION

Users authenticate to online applications still predominantly by sending a username *uid* and password to the authenticating party. It is well-known that this has severe limitations for security: users are required to manage numerous access credentials, which likely leads to password reuse or the usage of weak passwords [3, 25, 36]. It also requires the application to store large password databases and verify them on authentication securely.

An approach that significantly improves security (and usability) for both users and applications is Single Sign-On (SSO). This approach adds a third party for authentication between the user and the application: the Identity Provider (IdP). Instead of authenticating directly to the application, users authenticate to an IdP, as shown in Figure 1. The IdP then sends proof of the user's identity back to the application, also referred to as a Relying Party (RP). This proof comes in the form of a token τ_{id}. It typically is a standard signature under the IdP's public key on the user's identity and some context information, binding the token to a particular session. The most widely used SSO protocol on the Internet is OpenID Connect (OIDC) [39, 41], which extends the OAuth 2.0 framework [26] to provide user authentication. SSO services are becoming increasingly prominent, particularly with large social media or consumer device enterprises, such as Microsoft, Google, or Apple serving as IdPs [17, 18, 32].

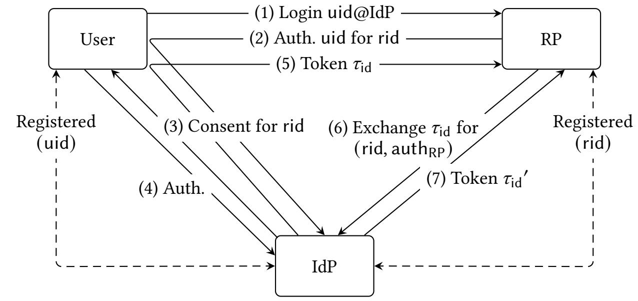

Figure 1: Authentication to an RP in the OIDC protocol. The user/RP must have previously registered with the IdP.

Security and privacy in SSO. In terms of security, users benefit from SSO as they only have to remember the access credential for the IdP, and RPs must not manage large password databases anymore but instead fully rely on the IdP for authentication. However, standard SSO also comes with risks, mainly because the IdP is a single point of failure that needs to be fully trusted. A particular drawback in SSO is its *lack of user privacy*, as the IdP learns every authentication request and thus is aware which RPs the user accesses when and in what frequency. Thus, while users might choose SSO for convenience, they pay by making all their online activities available to the IdP. Another slightly more subtle challenge is basing all RPs authentication security on the security of a single entity. If the IdP is corrupted, then the authentication towards all dependent RPs is also compromised.

Several proposals aim to improve these privacy and security limitations while preserving the convenience of SSO, such as [2, 19, 24] for hiding the users' access patterns towards the IdP and [1, 5] that provide better security by distributing the role of the IdP.

All these improved proposals have in common that they crucially rely on a particular form of SSO that is user-centric: the so-called *Implicit Flow*. This flow is necessary for achieving any form of privacy and supporting distributed IdP settings.

Implicit Flow — User-centric SSO for privacy? OIDC offers two main variants to provide a token to an RP, the *Authorization Code Flow* and the *Implicit Flow*. A third variant is the *Hybrid Flow*, which combines both. We detail these variants in Figure 2. In the Authorization Code Flow, the IdP directly communicates with the RP that

Proceedings on Privacy Enhancing Technologies 2023(4), 96–116
© 2023 Copyright held by the owner/author(s).
https://doi.org/10.56553/popets-2023-0100

the user wishes to authenticate. In contrast, in the Implicit Flow, no communication between the IdP and RP is necessary. That is, the IdP sends a token to the user, who forwards the token to the RP.

Clearly, in the Authorization Code Flow, there is no hope for achieving any user privacy in the sense of hiding the user's access patterns. However, interestingly the Implicit Flow does not allow for such privacy by default either: While the Implicit Flow does not require direct communication between the IdP and RP, the IdP is still required to learn the RP's identity, according to its specification. The reason is that the token issued by the IdP is supposed to be bound to the particular RP the user requests. Thus, the OIDC standard still assumes the user to send this RP identifier rid to the IdP, such that the IdP can sign the identifier as part of the token.

Resolving this problem was one of the core contributions of Hammann et al. [24] that propose a new Implicit Flow variant — which they call Privacy OIDC, or POIDC in short — where the user only sends a cryptographic commitment on rid to the IdP. This approach allows hiding the rid towards the IdP, but by signing the commitment, the IdP can still bind the token to the targeted RP.

RP authentication in OIDC. While the POIDC protocol by Hammann et al. now allows for a truly privacy-friendly use of the Implicit Flow, it amplifies another more fundamental problem of this OIDC variant — the lack of RP authentication.

In OIDC, users and RPs must register with the IdP. During this RP registration, the RP provides a set of metadata and operational information, which allows the IdP to check that the RP is a legitimate service and, for example, indeed owns the web domains it claims. Upon successful registration, the RP receives a unique identifier called rid. This identifier is associated with the registered metadata and an authentication method that later allows the RP to provide proof of registration to the IdP denoted by $auth_{RP}$.

In the Authorization Code Flow, RP authentication is mandatory, i.e., when a user initiates an authentication session to an RP, the RP must properly authenticate to the IdP as rid via the registered authentication method. However, RP authentication is not specified in the Implicit Flow, i.e., the RP cannot provide proof to the IdP that it initiated the request, making this variant much more prone to phishing attacks. In fact, the Implicit Flow was initially mainly aimed at RPs with no secrets. If ones assume RPs to have secrets, RP authentication can easily be added to the Implicit Flow by letting the RP send proof, via the user, that it initiated the user's authentication request. This proof can be a signature from the RP on the user's request. This is similar to the approach enabled by the Hybrid Flow.

While beneficial for security, this addition now destroys any privacy again as the authentication of the RP reveals its identity towards the IdP as part of the authentication, making all the user's interactions traceable by the IdP.

The need for a new Implicit Flow. Based on these contradicting design decisions, the upcoming OAuth framework specification, on which OIDC is based, omits the Implicit Flow and emphasizes the exclusive use of the Authorization Code Flow [27, 30]. Abandoning the Implicit Flow would clearly be detrimental to user privacy in SSO, as the Authorization Code Flow rules out any hope for privacy already on the communication level. This development leaves us with the following urgent question:

How can we improve the Implicit Flow to allow for RP authentication while preserving the privacy benefits of this flow, i.e., without revealing the identity of the RP to the IdP?

On a more practical level, adding such RP authentication is becoming much more important when offering user privacy: If users are not paying with their data anymore, the IdP needs another source of revenue. It would only be fair if users *and* RPs were paying for the service the IdP provides. However, this change requires the IdP to limit its services to registered RPs only.

1.1 Contributions

We answer the previous question by proposing the Authenticated Implicit Flow (AIF), a new Implicit Flow variant that supports explicit yet *privacy-preserving* RP authentication. In this new flow, all communication is still routed through the user, but the IdP is able to ensure that the user is authenticating to a registered RP and to blindly bind the issued token to the requested RP without learning its identity. We start by providing a formal game-based model for all desired security and privacy properties. This model enables us to formally analyze and compare the existing works (or rather slightly extended versions thereof), i.e., OIDC's Implicit Flow, POIDC [24], and show that none achieves all properties simultaneously. We then propose a new protocol, prove its security in our model, and compare its efficiency to the aforementioned solutions.

Formal security model. Our work introduces and formally defines a new OIDC variant, the Implicit Flow with RP authentication. The first challenge is to formalize the multi-message protocol run by three different entities in a generic syntax: the specified algorithms must closely follow the OIDC's communication model and be broad enough to capture the two existing and our new protocol, all being somewhat different in their cryptographic setup and achieved properties. At the same time, the system model and algorithms must be specific enough to express meaningful and (hopefully) easily digestible security properties. In the end, our model comprises 7 algorithms and 2 interactive protocols, running in 4 different phases of the protocol.

The next challenge is to identify and formalize the desired security properties for our privacy-preserving AIF. The properties must balance two seemingly conflicting needs: on the one hand, RP authentication should be done blindly towards the IdP; on the other hand, our system must still ensure proper authentication and correct binding of the IdP's blindly issued tokens. (Note that we do not model or propose how the user authenticates towards the IdP but assume that the IdP will only issue tokens for a uid when the user has provided sufficient authentication.) We capture proper RP authentication through the first two properties stated below and privacy as RP Hiding:

RP Accountability: An IdP can verify that an authentication request was initiated by a registered RP.

RP Session Binding: A token is immutably bound to the context in which it was issued, in particular to the RP authorized to make the request.

RP Hiding: An IdP receiving an authentication request cannot learn the RP's identity.

While these properties might be rather clear on an intuitive level, capturing them in a formal model is the core challenge of this work. Our security model is given in a game-based form, where the adversary needs to be given access to the multitude of algorithms and interactive protocols (when run by honest parties) and also be able to corrupt as many entities as possible. For both authentication-related properties, this meant finding a good balance between allowing corrupt behavior of RPs and still expressing strong security properties that take the inherent security "loss" stemming from blind authentication into account. This search for a good security model also led to the decision to explicitly model epoch-based credential renewal as part of the system, as this allows to capture corrupt RPs without losing all security: the security loss can then be contained to only the epochs in which an RP is corrupt and still legitimately registered. Thus, our system also implicitly includes a form of revocation, further adding to the complexity of our model.

Constructions. Our formal model for OIDC with RP authentication now allows to analyze existing protocols and improve the state of the art. Table 1 provides a comparison of the different schemes and their properties. We start by phrasing the native Implicit Flow, extended with simple signature-based RP Authentication (building upon the OIDC specification), as an instantiation $\mathsf{AIF}_{\mathsf{SIG}}$ of our generic model. When a user wants to authenticate towards a particular RP, this RP signs the user's request and lets the user forward the signature to the IdP. While trivially satisfying RP Accountability and RP Session Binding, this approach clearly does not allow for any privacy.

Next, we capture the POIDC protocol [24] by Hammann et al. as $\mathsf{AIF}_{\mathsf{COM}}$ in our model and provide the first formal security analysis of this protocol. Recall that therein tokens get blindly issued for a particular RP by letting the IdP only sign a commitment of the RP's identity *rid*. This protocol achieves the RP Hiding property but does not provide RP Accountability and only partially satisfies RP Session Binding.

We finally present our new protocol $\mathsf{AIF}_{\mathsf{ZKP}}$ and prove that it satisfies all three properties simultaneously. The new scheme builds upon $\mathsf{AIF}_{\mathsf{COM}}$, i.e., the IdP again only signs a commitment of the *rid* in the token. To provide RP authentication, we introduce (epoch-based) credentials containing the IdP's signature on the RP's *rid* and some epoch *ep*. When a user wishes to authenticate to an RP, the user creates a commitment for *rid* and asks the RP to prove that it owns a credential for that identity in the applicable epoch. The user and RP both know the opening to the commitment, allowing the user to check that the RP's identity is indeed correct. However, the IdP will only receive a zero-knowledge proof for the committed identity and corresponding credential, i.e., it can verify that a registered RP is requesting the authentication but not which. Still, by signing the commitment, the IdP's signature is bound to the explicit *rid* again. We formally prove that $\mathsf{AIF}_{\mathsf{ZKP}}$ achieves all security and privacy properties under standard assumptions.

Implementation and comparison. We further provide an implementation of $\mathsf{AIF}_{\mathsf{ZKP}}$ and compare its efficiency with the two existing protocols, $\mathsf{AIF}_{\mathsf{SIG}}$ and $\mathsf{AIF}_{\mathsf{COM}}$. Our implementation instantiates the $\mathsf{AIF}_{\mathsf{ZKP}}$ scheme with PS signatures [38] for the epoch-based credentials and combines them with Pedersen commitments [37]. Our performance evaluation of $\mathsf{AIF}_{\mathsf{ZKP}}$ on two reference devices for the different parties shows that it takes 9 milliseconds (ms) to create an RP authentication request and only 19ms for the IdP to verify it. The corresponding token finalization on the user device takes at most 9ms.

Protocol	RP Acc.	RP Sess. Bin.	RP Hiding
$\mathsf{AIF}_{\mathsf{SIG}}$	✓	✓	
$\mathsf{AIF}_{\mathsf{COM}}$		(✓)	✓
$\mathsf{AIF}_{\mathsf{ZKP}}$	✓	✓	✓

Table 1: Comparison of (partially) satisfied security properties of $\mathsf{AIF}_{\mathsf{SIG}}$ – Implicit Flow with standard signatures, $\mathsf{AIF}_{\mathsf{COM}}$ – adapted POIDC [24], and our $\mathsf{AIF}_{\mathsf{ZKP}}$ protocol. See Section 5 for a security analysis and Section 6 for a performance evaluation of all schemes.

1.2 Other Related Work

Enhancing privacy in SSO has been a lively research area, either aiming to conceal the user's access patterns from the IdP or hiding user information from corrupt RPs. These efforts crucially rely on a privacy-preserving communication pattern, such as the Implicit Flow. However, none of these proposals detail how their novel protocols would be incorporated into SSO or address the RP authentication aspect. Nonetheless, we provide a brief overview of all (somewhat) related works for completeness and to motivate our pursuit to rescue the Implicit Flow.

The SPRESSO [19] system, inspired by Mozilla's discontinued service BrowserID [34], provides a new SSO-like protocol where the IdP does not learn the RP's identity during an authentication session and also separates the communication between the RP and IdP, similar to the Implicit Flow. However, SPRESSO developed an entirely new protocol, whereas our goal is to be as OIDC-compliant as possible. Furthermore, a dedicated design choice of SPRESSO was to be an *open* system, in the sense that the user can authenticate to *any* RP without requiring any previous registration of the RP with the IdP. This contrasts our work, which requires RP registration (as demanded by OIDC) and focuses on privacy-preserving RP authentication.

PRIMA [2] and EL PASSO [42] propose user-centric SSO variants that combine classic SSO with (privacy-preserving) attribute-based credentials. In both protocols, the IdP issues short-term credentials to the user, which the user can independently show to RPs. This clearly provides the desired RP Hiding (as the IdP is no longer involved in the actual RP-specific authentication) but also makes any RP authentication impossible. That is, the IdP cannot limit its services to registered RPs. Moreover, both are entirely new protocols and rather SSO hybrids but incompatible with the current OIDC communication setting.

PseudoID [15] focuses on hiding the user's unique identity (and attributes) from the RP by having users obtain the IdP's signature on a blinded pseudonym and authenticate to an RP with the unblinded token. Neither RP authentication nor any binding of the RP's identity in an IdP's token has been considered in this work, i.e., it tackles an orthogonal aspect of privacy than our work, but also relies on an SSO setting that does not require direct interaction between the IdP and RP.

Other recent SSO security improvements utilize distributed IdP roles, with protocols like PASTA [1] and PESTO [5] ensuring unforgeable tokens and secure user passwords as long as not all IdPs are corrupted. These protocols do not address privacy concerns but also require the Implicit Flow to function: the user-centric message flow is needed as the user must interact with multiple IdPs and combine their contributions into a standard authentication token to be sent to the RP.

In conclusion, the only truly related works to our work on privacy-preserving RP authentication are the actual Implicit Flow with additional RP authentication as enabled in the OIDC specification [41] and the POIDC protocol by Hammann et al. [24]. For both, we provide an in-depth analysis: Section 5 formally analyzes their security and privacy properties in our model, and Section 6 compares the efficiency of our new protocol with both previous approaches.

2 BACKGROUND & BUILDING BLOCKS

In the first part of this section, we provide an overview of OIDC, focusing on RP registration and authentication. We discuss the token structure, its authentication protocols, and how RP authentication is handled in each variant. We then assess the native realization of RP Accountability, RP Session Binding, and RP Hiding in each protocol. In the second part, we present the necessary cryptographic building blocks needed in the studied protocols.

2.1 RP Authentication in OIDC

To enable RP authentication, eligible parties must first register with the IdP. The RP registration ensures that all protocol configurations and RP-related information (name, legal information, picture) are stored with the IdP. This information is then linked to the unique RP identifier rid. In the subsequent protocol, an RP must always provide rid to enable the IdP to look up the stored information, which is then, for example, used for the user consent dialog.

One aspect of the RP's configuration is its authentication method. In contrast to OAuth, which only specifies symmetric keys for RP authentication, OIDC specifies standard signatures for this purpose [40]. With that, an RP registers its signature public key rpk with the IdP and provides a signature in its authentication request $auth_{RP}$, which is generated with its private key rsk.

Once the requested RP and the user have authenticated to the IdP, a token is issued by the IdP to authenticate the user to the RP. Before delving into the various RP authentication protocols, we will briefly overview the exchanged token structure.

Identity token. The token that authenticates a user to an RP is a signed and short-lived JSON Web Token (JWT) [28], called an *identity token.* We denote this token by τ_{id}, which is the IdP's signature on the tuple (rid, uid, n, ctx) for a user uid, nonce n, and context ctx, holding a timestamp and requested user information. This signature results from a standard signature scheme and can be verified using the IdP's public key ipk. The successful signature verification authenticates a user to the RP rid.

RP authentication protocols. OIDC extends three OAuth protocols: the *Implicit Flow (IF)*, *Authorization Code Flow (ACF)*, and *Hybrid Flow (HF)*. The latter is a seamless fusion of the IF and ACF, enabling an RP to directly obtain an identity token and request an additional token later. Figure 2 illustrates the two main flows, IF and ACF.

The IF is designed for RPs that cannot store credentials, resulting in $auth_{RP}$ being empty and an identity token being directly issued via the *front-channel*, referring to communication through the user. Conversely, in the ACF, RPs are assumed to safeguard their credentials. Thus, they can authenticate their initial request with $auth_{RP}$ using *standard signatures*. This flow then grants a temporary *authorization code* τ_{ac} that is later exchanged for an identity token over the *back-channel*, referring to direct communication between the RP and IdP, with the RP authenticating to the IdP. By combining both flows, the HF enables using the IF in conjunction with RP authentication via standard signatures. We will leverage this as the basis for our $\mathsf{AIF}_{\mathsf{SIG}}$ construction.

Note that we simplify the protocol description in two ways. Firstly, we omit redirection information passed between the RP and IdP in steps (2)-(3) and (7)-(8). We elaborate on this in Section 6.2. Secondly, we do not introduce OAuth *access token* that allow RPs to query further user information, as this conflicts with our privacy objectives and sole focus on authentication via OIDC.

Security and privacy of native OIDC. We now examine the security and privacy properties achieved by the IF and ACF in relation to RP authentication. Our primary objectives are to ensure that the IdP can verify that a valid RP initiated the authentication request (RP Accountability), that an identity token is bound to the session of an authorized RP (RP Session Binding), and that the IdP does not learn the requested RP's rid (RP Hiding).

RP Accountability: The IF does not provide any RP authentication, while ACF ensures accountability as an RP authenticates with the IdP during the initial request or token exchange.

RP Session Binding: In all flows, the signed identity token includes the RP's rid, the user uid, a nonce n, additional user context, and operational information. The IdP's signature on these values ensures its binding — particularly to the rid. The fact that the IdP always receives the (authenticated) rid allows for verifying that the RP is registered. But due to the absence of any RP authentication in the IF, RP Session Binding only holds if all users and all RPs are honest, i.e., the IdP can rely on the correctness of the provided rid.

RP Hiding: None of the flows provide a mechanism for hiding the RP identity from the IdP. The users' access patterns become evident to the IdP by simply recording the rid that *all* protocols provide in their initial request.

In conclusion, the Implicit Flow offers a privacy advantage as all messages are proxied through the user, which prevents direct communication between the RP and IdP, limiting the amount of information disclosed to the IdP. Throughout the rest of this paper, we will leverage this variant and present a scheme that meets all three security properties.

2.2 Building Blocks

This section introduces the building blocks necessary for all considered constructions. Let $\kappa \in \mathbb{N}$ denote the security parameter. We use $\perp$ to indicate a failure, and an empty string is denoted by ϵ.

Commitment scheme. We denote $\mathsf{COM} := (\mathsf{Commit}, \mathsf{Open})$ as a commitment scheme as with message space $\mathcal{S}_{\mathsf{Com}}$. Let $(o, c) \leftarrow \mathsf{Commit}(m)$ denote a commitment c and opener o to message $m \in$

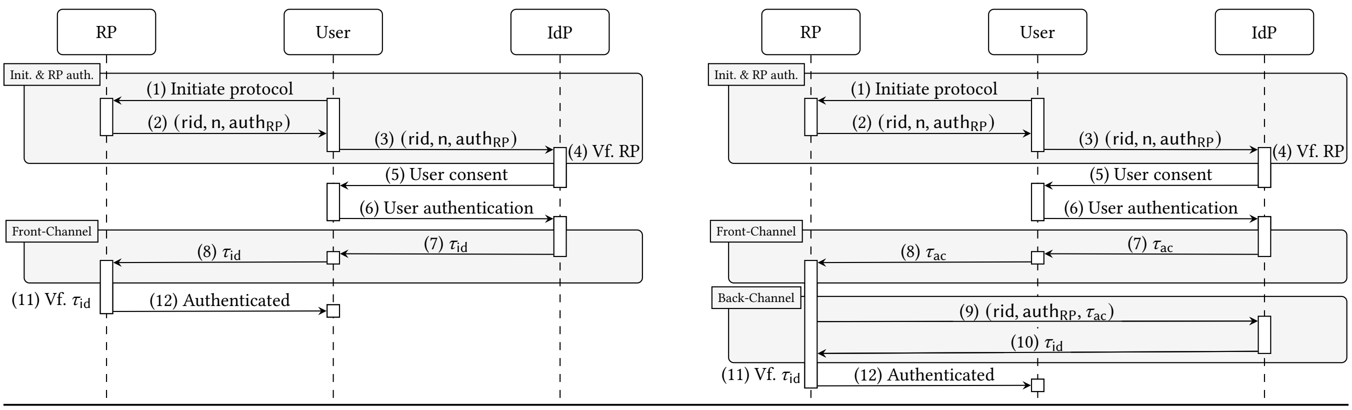

Figure 2: OIDC authentication protocols: Implicit Flow (IF) on the left, Authorization Code Flow (ACF) on the right.

$\mathcal{S}_{\mathsf{Com}}$. The algorithm $\mathsf{Open}(m, c, o)$ returns 1 if the commitment is valid and 0 otherwise. We require COM to be *hiding* and *binding*.

A simple instantiation of commitments is $c \leftarrow \mathsf{H}(o, m)$, where H is a hash function (modelled as random oracle) and o a random string. Such an instantiation will be sufficient for POIDC, resembled as construction $\mathsf{AIF}_{\mathsf{COM}}$. Our new protocol $\mathsf{AIF}_{\mathsf{ZKP}}$ requires a commitment scheme with an algebraic structure, for which we will use Pedersen commitments [37].

Zero-knowledge proofs. We denote generic *non-interactive zero-knowledge proofs* of knowledge of a witness w, such that the statement $s(w)$ is true, as $\pi \leftarrow \mathsf{NIZK}\{(w) : s(w)\}(ctx)$, where the proof π is immutably bound to some context ctx. We require these proofs to be *zero-knowledge* and *simulation-sound extractable* [23]. The latter states that even after the adversary has seen simulated proofs on arbitrary statements in a security experiment, if it constructs a new valid proof on any statement, then the environment of the adversary can extract the proof witness using extractor Ext. Due to the zero-knowledge property, there exists a simulator Sim that can be used to create verifiable NIZKs without knowing their witness.

For concrete DL-based realizations of NIZKs, i.e., generalized Schnorr-signature proofs [12], we will use the Fiat-Shamir heuristic [20] to make them non-interactive, where ctx is included in the challenge hash. These proofs are well-known to satisfy the required properties of zero-knowledge and simulation-sound extractability in the random oracle model.

Standard signature scheme. A standard signature scheme is defined as $\mathsf{SIG} := (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ consisting of the key generation algorithm $(sk, pk) \leftarrow \mathsf{KGen}(1^\kappa)$, signing algorithm $\sigma \leftarrow \mathsf{Sign}(sk, m)$ for messages $m \in \mathcal{S}_{\mathsf{Sig}}$, and verification algorithm $0/1 \leftarrow \mathsf{Vf}(pk, m, \sigma)$. We will need SIG to satisfy the standard Existential Unforgeability under Chosen Message Attack (EUF-CMA) security, the definition is given in App. A. In our instantiation, we will use RSA signatures for compatibility with existing standards.

Multi-message signature scheme. This variant $\mathsf{MMS} := (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ extends standard signatures to sign a message vector $\vec{m} := (m_1, ..., m_\ell) \in \mathcal{S}^\ell_{\mathsf{MMS}}$ at once. It consists of $\mathsf{Setup}(1^\kappa)$ that outputs the public parameter pp. The key generation algorithm $\mathsf{KGen}(pp, \ell)$ takes pp, the message vector dimension ℓ, and returns the key pair (sk, pk). $\mathsf{Sign}(sk, \vec{m})$ now creates the signature σ on the message vector $\vec{m}$, and $0/1 \leftarrow \mathsf{Vf}(pk, \vec{m}, \sigma)$ verifies them.

We need MMS to satisfy the MMS-EUF-CMA security definition which is a straightforward extension of the standard unforgeability: the adversary wins if it forges a valid signature σ^* on a fresh $\vec{m}^*$, which was not queried to the Sign-oracle before. For completeness, the definition is given in App. A. We also require that MMS supports the creation of efficient NIZKs (defined next) that prove a valid signature σ on $\vec{m} := (m_0, m_1)$ w.r.t. pk while revealing only m_1 and not revealing any information about the signature σ or message m_0 : $\mathsf{NIZK}\{(\sigma, m_0) : \mathsf{Vf}(pk, (m_0, m_1), \sigma) = 1\}(m_1)$.

Multi-message signatures with committed identities. We will combine multi-message signatures with commitments in our new construction $\mathsf{AIF}_{\mathsf{ZKP}}$. For that, we extend the previous notation and denote a NIZK that proves knowledge of a signature $\sigma \leftarrow \mathsf{Sign}(sk, \vec{m})$ on message vector $\vec{m} := (m_0, m_1)$ and an opener o for commitment $(o, c) \leftarrow \mathsf{Commit}(m_0)$ with

$$\mathsf{NIZK}\{(\sigma, m_0, o) : \mathsf{Vf}(pk, (m_0, m_1), \sigma) = \mathsf{Open}(m_0, c, o) = 1\}(m_1, c)$$

which proves knowledge of a signature σ on $\vec{m}$ under pk and an opener o that successfully opens the commitment c to the signed message m_0. The proof thereby only reveals message m_1 and the commitment c.

The underlying idea is similar to anonymous credentials and group signatures [6, 7, 14, 16], which also follow the *sign-and-encrypt-and-prove* paradigm where users authenticate by proving knowledge of a membership credential. While the core idea is the same, there is a subtle but crucial difference: we require the authentication or rather verification to be available “in parallel” in two types. Anyone who only knows the proof and commitment can verify that a valid group member created the signature. If a verifier additionally receives the opening, it also learns the identity of the signer. None of the existing group signatures or anonymous credentials support this feature out of the box, i.e., we could not use them as a building block but instead built this tailored variant from scratch.

In our instantiation, we will use PS signatures [38] for the MMS scheme, which supports all required features.

3 AUTHENTICATED IMPLICIT FLOW

This section formally defines our proposal for an Authenticated Implicit Flow scheme AIF that supports privacy-preserving RP authentication towards the IdP. That is, even without learning the identity of the RP, the IdP can ensure that the user authenticates to a *valid* RP while still being able to bind the RP's identity to the issued token.

A scheme enabling fully blind RP authentication cannot achieve RP Accountability if a single RP is corrupt, unless revocation is also used. We opted for a renewal-based approach to model such revocation, which we motivate first. We then outline our scheme's general procedures and formalize the security and privacy properties.

RP revocation via short-lived credentials. We want to define and realize meaningful security properties for RP authentication, including the possibility of corrupt RPs. In a fully privacy-preserving scheme where RP authentication happens blindly, a single corrupt RP can undermine the desired accountability — unless revocation is used. Therefore, we need to include such revocation in our model. To be compatible with blind authentication would require *privacy-preserving* revocation in any concrete instantiation. Such solutions exist [4, 8, 13, 35] but would incur significant efficiency penalties for every (privacy-preserving) scheme. While acceptable from an academic perspective, our goal is to provide a viable solution for real-world deployment.

We thus opt to model revocation by making the credentials the RPs receive short-lived and requiring them to be updated regularly. We phrase this with *epoch-based credentials* and a dedicated renewal process in our syntax, following previous works [10, 11, 29]. While this makes the syntax and model more complex — we need to define and capture this renewal process and epochs now — this later allows for very simple and highly efficient realizations.

Roughly, the idea for using epoch-based credentials for revocation is as follows: The membership credentials, determining which RP is allowed to use the IdP's service, are now bound to an *epoch* *ep* and need to be renewed for every new epoch. The benefit of this renewal process is that it is independent of a concrete user session, i.e., there is no need for privacy here. The IdP learns the RP's *rid* in every renewal request and can determine whether this RP is eligible for a new credential. In concrete deployment, the IdP will maintain a black- or whitelist of *rid*'s to determine which RPs are allowed to use its service, e.g., depending on their paid membership status or reports of misbehavior.

The revocation occurs as every RP authentication in a concrete user session must provide proof of a valid credential for the current epoch. That is, if an RP *rid* is recognized as malicious in some epoch *ep* and supposed to get revoked, the IdP will not issue the RP a new credential in $ep+1$ (or any subsequent epoch), and thus this RP loses the capability to perform such proofs. It is desirable to make the epochs relatively short for effective revocation, e.g., require updates daily or weekly. Given that renewal does not require privacy, the instantiations are very efficient and thus can easily be performed in high frequency.

3.1 System Overview

In our AIF system, the IdP is the central entity to issue identity tokens of user *uid* towards several RPs. For setup, the IdP runs SetupIdP to generate a key pair $((isk, \mathcal{M}), ipk)$ from the public parameters *pp*. All entities in the system receive *ipk*, and the tokens issued by the IdP will get verified against that public key. $\mathcal{M}$ denotes the state in which the issuer maintains the RP memberships. The procedures that are required to enable our security and privacy goals can be described along the following four phases, which are illustrated in Figure 3. In the following, we give a high-level intuition of these phases and present the detailed syntax in Section 3.2.

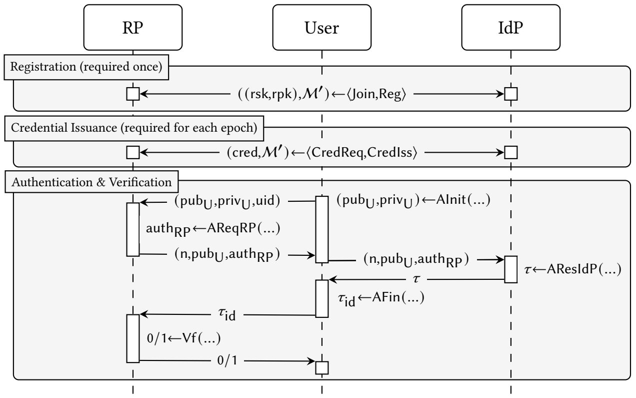
RP
User
IdP
Registration (required once)
((rsk,rpk),$\mathcal{M}'$)←⟨Join,Reg⟩
Credential Issuance (required for each epoch)
(cred,$\mathcal{M}'$)←⟨CredReq,CredIss⟩
Authentication & Verification
(pub$_U$,priv$_U$,uid)
(pub$_U$,priv$_U$)←AInit(...)
auth$_{RP}$←AReqRP(...)
(n,pub$_U$,auth$_{RP}$)
(n,pub$_U$,auth$_{RP}$)
τ←AResIdP(...)
τ
τ_{id}←AFin(...)
τ_{id}
0/1←Vf(...)
0/1

Figure 3: Overview of the different phases of an AIF protocol.

(1) Registration. Before an IdP can issue an identity token for an RP, the RP must first register via the ⟨Join, Reg⟩ protocol. In this process, an RP generates a key pair (rsk, rpk) and provides the IdP with the public key *rpk* along with its identity *rid*. The IdP then stores (rid, rpk) in the member state $\mathcal{M}$.

(2) Credential issuance and renewal. After registration, the RP must receive an authentication credential *cred* for each epoch *ep* from the IdP. This issuance is handled via the ⟨CredReq, CredIss⟩ protocol. The authentication is based on the RP keys generated at registration. We stress that there is no privacy need in this procedure, and the IdP learns (and must learn) the RP's identity *rid* in this phase. To distinguish different sessions and ensure freshness in each session, the protocol also gets a session identifier *sid* as input.

(3) Authentication. To authenticate to an RP, the user with *uid* executes AInit with the corresponding *rid*, receiving $(pub_U, priv_U)$. The user sends these values and its *uid* to the RP, which creates $auth_{RP}$ for pub_U and *cred* to authenticate as the legitimate RP in epoch *ep*. To ensure freshness, the RP provides a unique session nonce *n*, assumed to be globally unique. The user forwards the authentication request to the IdP.

When the IdP receives a request from a user *uid* for a session with nonce *n* and for an RP implicitly authenticated via $(pub_U, auth_{RP})$, it runs the algorithm AResIdP leading either to a token τ or $\perp$ if the RP authentication fails. The token may include additional information like timestamps and user attributes, denoted as *ctx*. We assume a out-of-band authentication between the user and IdP so that AResIdP is only run for verified *uid*.

Note that AResIdP does not receive *rid* as an explicit input, which is necessary for achieving RP Hiding. However, *rid* is implicitly contained in pub_U, authenticated through $auth_{RP}$. To verify the final token for a specific *rid*, we transform the token τ from the

IdP with the committed rid. The user accomplishes this using the AFin algorithm with the private value $priv_U$ generated during the authentication request. The AFin algorithm takes all the received information, including $priv_U$, as input and produces a final identity token τ_{id}.

(4) Verification. The resulting token τ_{id} can be verified to ensure its validity for the tuple (rid, uid, ctx, n, ep) w.r.t. ipk. This binds all session information, user and RP identifiers explicitly together.

3.2 Syntax

An Authenticated Implicit Flow AIF is defined as a tuple of seven algorithms and two (possibly) interactive protocols $\mathsf{AIF} :=$ (Setup, $\mathsf{SetupIdP}$, $\langle\mathsf{Join}, \mathsf{Reg}\rangle$, $\langle\mathsf{CredReq}, \mathsf{CredIss}\rangle$, AInit, AReqRP, $\mathsf{AResIdP}$, AFin, Vf):

$\mathsf{Setup}(1^\kappa) \rightarrow pp$: Takes the security parameter $\kappa \in \mathbb{N}$ and outputs the public parameters pp, which are the implicit input for all other algorithms.

$\mathsf{SetupIdP}(pp) \rightarrow ((isk, \mathcal{M}), ipk)$: Returns the IdP's keys, isk is the secret key, $\mathcal{M}$ the membership state, and ipk the public key.

$\langle\mathsf{Join}(ipk, rid), \mathsf{Reg}(rid, \mathcal{M})\rangle \rightarrow \{((rsk, rpk), \mathcal{M}'), \perp\}$: The RP with rid executes the interactive protocol to register with the IdP with ipk. Upon success, the RP obtains its key pair (rsk, rpk), and the IdP outputs an updated member state $\mathcal{M}'$. It returns $\perp$ to indicate a failure.

$\langle\mathsf{CredReq}(ipk, rid, rsk, sid, ep), \mathsf{CredIss}(rid, isk, \mathcal{M}, sid, ep)\rangle \rightarrow \{(cred, \mathcal{M}'), \perp\}$: The RP rid runs the interactive protocol with the IdP with (isk, ipk) and a session nonce sid, unique for each credential issuance. Upon success, the RP obtains its credential $cred$ issued for epoch ep, and the IdP outputs its updated member state $\mathcal{M}'$. If the RP is not a valid member, it outputs $\perp$.

$\mathsf{AInit}(ipk, rid) \rightarrow (priv_U, pub_U)$: Run by the user, returns the public pub_U and private $priv_U$ user output to initialize a token request at rid to IdP with ipk.

$\mathsf{AReqRP}(ipk, rid, cred, uid, pub_U, priv_U, n, ep) \rightarrow auth_{RP}$: Run by the RP, creates an authentication request $auth_{RP}$, requesting a token in epoch ep from the IdP with ipk for user uid and the public pub_U and private $priv_U$ user output, using its credential $cred$ and a nonce n.

$\mathsf{AResIdP}(isk, \mathcal{M}, uid, ctx, auth_{RP}, pub_U, n, ep) \rightarrow \{\tau, \perp\}$: Run by the IdP, generates a token τ for user uid, nonce n, context ctx, epoch ep, and public user output pub_U. The IdP can use $auth_{RP}$ and its member state $\mathcal{M}$ to verify whether the request is intended for a valid RP in epoch ep. If this verification fails, it returns $\perp$.

$\mathsf{AFin}(ipk, rid, uid, ctx, pub_U, priv_U, n, ep, \tau) \rightarrow \{\tau_{id}, \perp\}$: Run by the user, takes the rid and uid, context ctx, nonce n, public pub_U and private $priv_U$ user output, epoch ep, and the token τ. It outputs a final identity token τ_{id} or $\perp$, indicating that the inputs were not valid.

$\mathsf{Vf}(ipk, (rid, uid, ctx, n, ep), \tau_{id}) \rightarrow 0/1$: Returns 1 if τ_{id} is valid w.r.t. ipk for the given rid, uid, ctx, n, ep, or 0 otherwise.

For a better overview, Figure 4 summarizes all previously introduced abbreviations. Note that we denote the set of nonces as $\mathcal{Z}$ and the set of epochs as $\mathcal{T}$. The correctness of our scheme, utilizing this notation, is given in App. B.

Abbreviations	**Description**
$isk, ipk, \mathcal{M}$	IdP's secret key isk, public key ipk, member state $\mathcal{M}$
rid, rsk, rpk	RP's identity rid, secret key rsk, public key rpk
uid, ctx	User's identity uid, context ctx including user information
$pub_U, priv_U$	Public and private user output in an authentication session
sid, n, ep	Session nonces $\{sid, n\} \in \mathcal{Z}$, epoch $ep \in \mathcal{T}$
$cred$	Issued by the IdP to the RP and only valid for ep
$auth_{RP}$	Authentication data provided by an RP
τ	Token that is issued by the IdP
τ_{id}	Token finalized by the user and verified by the RP

Figure 4: Grouped abbreviation overview.

4 SECURITY MODEL

This section formally defines the security and privacy properties expected from an AIF system, excluding user authentication from our model and focusing solely on RP authentication. In summary, we aim to ensure the following properties:

RP Accountability: An IdP can ensure that a *valid* RP initiates an authentication request. $\mathsf{AResIdP}$ returns $\perp$ if the request does not originate from an RP that is properly registered (in the epoch of the request).

RP Session Binding: Even though the IdP should not learn the RP's identity rid when responding to its authentication request, the finalized identity token τ_{id} must be bound to the session in which it was requested. In particular, this session includes the RP's rid authorized to make the request in epoch ep.

RP Hiding: Despite being able to verify that an RP authentication request is intended for a valid RP, the IdP when performing $\mathsf{AResIdP}$, learns nothing about the RP identity rid.

4.1 Oracles

We now formalize the desired security properties in a game-based manner, where an adversary $\mathcal{A}$ runs an experiment with a challenger. The challenger is in charge of all honest entities and maintains their private states, provided in Figure 5. The adversary can interact with honest entities through the oracles, defined in Figure 6 and summarized below. All sets are initialized with $\emptyset$ and variables with 1 when starting the game. We assume the session nonce sid an RP uses to re-authenticate to the IdP to be globally unique.

Register with an honest IdP. The following oracles model an adversary's ability to register honest and corrupt RPs with an honest IdP. They will be available in the authentication-related properties expressed by RP Accountability and Session Binding.

Join-Reg : Registers an honest RP by running $\langle\mathsf{Join}, \mathsf{Reg}\rangle$, storing rsk in $\mathsf{HRID}[rid]$, and returning rpk. It allows $\mathcal{A}$ to initialize honest RPs and later request their authentication sessions.

Reg : Registers a corrupt RP by running Join with $\mathcal{A}$, storing rid in CRID. It models an active attack on the registration with the honest IdP, allowing $\mathcal{A}$ (as corrupt RP) to fully deviate from the protocol and control all of the RP's secret state.

CredReq-CredIss : Issues a credential $cred$ for an honest rid in the current epoch cep. It picks a session nonce sid, runs $\langle\mathsf{CredReq}, \mathsf{CredIss}\rangle$ for rid, and stores $cred$ in $\mathsf{HRID}[rid, cep]$. This oracle allows $\mathcal{A}$ to steer the behavior of honest RPs and keep them active in progressing epochs. $\mathcal{A}$ does not know any of the RPs secret states unless it corrupts them via the CrptRP oracle.

CredIss : Issues a credential $cred$ in the current epoch cep for a corrupt RP. It picks a session nonce sid and runs CredIss with the adversary, who is free to deviate from the protocol and fully controls the RP. If RP is properly registered, it puts rid in $\mathsf{CRID}[cep]$. This will mark corresponding epochs and authentications therein as trivial. Note that the oracle does *not* add rid to $\mathsf{CRID}[cep]$ if the corrupt rid was not registered or the credential issuance failed. This oracle models that an illegitimate acquisition of credentials through the renewal process is a valid attack strategy in our authentication-related security games.

Register with a corrupt IdP. The following oracles model the adversary's ability – in the role of a corrupt IdP– to register honest RPs and keep them active in progressing epochs. These oracles are only available in our privacy-related RP Hiding game. The adversary fully controls the IdP here and can arbitrarily deviate from its protocol.

Join : Registers an honest RP by running the RP's part of the Join protocol with $\mathcal{A}$. It stores rsk in $\mathsf{HRID}[rid]$ and returns rpk.

CredReq : Requests a credential $cred$ for an honest RP in epoch cep. It runs the honest CredReq protocol with $\mathcal{A}$ and upon success stores $cred$ in $\mathsf{HRID}[rid, cep]$.

Authentication with an honest party. The subsequent oracles model the adversary's ability to engage with an honest party for an authentication session. The first two oracles model $\mathcal{A}$'s interaction with an honest user or honest RP (possibly towards a corrupt IdP). The latter two allow the adversary to engage with an honest IdP.

AInit : Initializes the authentication protocol for an honest user towards a (possibly corrupt) RP. It generates and returns $(ses, pub_U, priv_U)$ and internally stores a session record $\mathsf{SES}[ses]$. This session can later be finalized through the AResIdP-AFin oracle.

AReqRP : Returns an honest RP's authentication request $auth_{RP}$ for adversarially provided user input, which it stores in REQ.

AResIdP : Returns the IdP's token τ for adversarial input. It models that an honest IdP will react to any authentication request. The input can be fully adversarially generated or fully/partially stem from the oracles above. For RP Session Binding, this oracle is only available for corrupt users.

AResIdP-AFin : Generates a token τ for an adversarial $auth_{RP}$, which an honest user finalizes to τ_{id}. This oracle completes the session $\mathsf{SES}[ses]$ initiated by the adversary through the AInit oracle and returns τ_{id}. The oracle is only available in RP Session Binding, prioritizing security from the perspective of honest users. It models the interaction between an honest user and an honest IdP, ensuring that $\mathcal{A}$ cannot intercept or modify exchanged messages. However, $\mathcal{A}$ can influence the token through the adversarial RP authentication input.

Corruption of RPs and epochs. The last two oracles allow the adversary to corrupt initially honest RPs adaptively and move epochs.

CrptRP : Exposes the secret key rsk and the credential $cred$ of the current epoch cep of an honest RP. It removes rid from HRID and marks it as corrupt in cep by adding rid to $\mathsf{CRID}[cep]$.

SetEP : Allows $\mathcal{A}$ to increase the current epoch cep, which is then used by all honest entities. If $\mathcal{A}$ updates the epoch, it must also renew the credentials (via the appropriate oracles) of all RPs that should remain authorized in that new epoch.

Variable	Description
cep	Represents the current epoch $cep \in \mathcal{T}$
HRID, CRID	Stores rids of honest/corrupt RPs registered at the IdP
$\mathsf{HRID}[rid]$	Secret key rsk of honest RP rid
$\mathsf{CRID}[cep]$	Corrupt rids of RPs with a credential in epoch cep
$\mathsf{HRID}[rid, cep]$	Credential $cred$ of honest RP rid in epoch cep
HUID, CUID	Stores uids of honest/corrupt users registered at the IdP
$\mathsf{SES}[ses]$	Stores $(rid, uid, pub_U, priv_U)$ for each session $ses \in \mathbb{N}$
REQ	Stores (uid, n, pub_U, cep) for each RP auth. request
RES	Stores (rid, uid, n, ctx, cep) for each issued identity token

Figure 5: Keys and states maintained by challenger.

4.2 RP Accountability

This property captures the security guarantees that an honest IdP has in the presence of corrupt RPs and corrupt users. Despite the IdP now performing its part of the authentication blindly — when it comes to the RP's identity — we still want to ensure that the IdP only returns a token $\tau \neq \perp$ when the request stems from an RP that is properly authenticated. Recall that legitimation of RPs happens in two stages: they first need to register with the IdP and then obtain a credential for each new epoch. Thus, "properly authenticated" means that if an honest IdP receives a request $(auth_{RP}, pub_U)$ for a session (uid, n, ctx, ep), where AResIdP does *not* output $\perp$, this request must originate from an RP that has been registered *and* owns a valid credential for epoch ep.

While formal security properties guarantee the absence of *any* successful attack in a pre-defined model, it might also be helpful to look at concrete attacks that are captured:

- An attacker (posing as corrupt RP and/or user) cannot re-use the authentication request created by an honest RP in any context other than what the honest RP wanted.
- A corrupt RP, being registered and having valid credentials for (some) epochs E but not for $ep \notin E$, cannot re-use any of its old authentication credentials to create a valid session in epoch ep.

We translate this intuition into a unforgeability-type of game, where the adversary — after interacting with a number of honest RPs and the honest IdP — wins if it can output a forgery $(uid^*, ctx^*, pub_U{}^*, n^*, auth_{RP}{}^*, cep)$ such that (1) the honest IdP "accepts" the RP authentication, i.e., AResIdP for ipk does not output $\perp$, and (2) the forgery is not trivial.

Non-trivial means that no honest RP created a request for such a session, and no corrupt RP has a valid credential for the current epoch cep. Note that the latter is unavoidable in our privacy-preserving setting: the rid of the requested RP should be entirely hidden from the IdP, and authentication merely requires a valid proof of membership — which an adversary with a credential for that epoch can trivially do. (However, we are able to define stronger security for corrupt RPs through the next RP Session Binding property, as this relates to the final token that contains the RP's identity again.)

The RP Accountability game considers both honest and corrupt RPs (with the restriction just mentioned) as follows: An adversary can register honest RPs through $\mathcal{O}$.Join-Reg, let them retrieve credentials via $\mathcal{O}$.CredReq-CredIss, make them issue authentication requests through the $\mathcal{O}$.AReqRP oracle, and receive tokens from the IdP via the $\mathcal{O}$.AResIdP oracle. It can also use $\mathcal{O}$.CrptRP to corrupt honest RPs, manipulate the current epoch with $\mathcal{O}$.SetEP, register

```
$\underline{\mathsf{Join\text{-}Reg}(\mathsf{rid})}^{\text{RP Accountability — Honest IdP, honest RP}}$
  If $(\mathsf{rid} \in \mathsf{HRID} \cup \mathsf{CRID})$ return $\bot$
  $\mathsf{HRID} := \mathsf{HRID} \cup \{\mathsf{rid}\}$
  Run $\langle \mathsf{Join}(\mathsf{ipk}, \mathsf{rid}), \mathsf{Reg}(\mathsf{rid}, \mathcal{M}) \rangle$
  Upon output $((\mathsf{rsk}, \mathsf{rpk}), \mathcal{M}')$
    $\mathsf{HRID}[\mathsf{rid}] := \mathsf{rsk}$; Return $\mathsf{rpk}$
  If the protocol fails, return $\bot$

$\underline{\mathsf{Join}(\mathsf{ipk}, \mathsf{rid})}^{\text{RP Hiding — Corrupt IdP, honest RP}}$
  If $(\mathsf{rid} \in \mathsf{HRID} \cup \mathsf{CRID})$ return $\bot$
  $\mathsf{HRID} := \mathsf{HRID} \cup \{\mathsf{rid}\}$
  Run $\mathsf{Join}(\mathsf{ipk}, \mathsf{rid})$ with $\mathcal{A}$ (being the corrupt IdP)
  Upon output $(\mathsf{rsk}, \mathsf{rpk})$
    $\mathsf{HRID}[\mathsf{rid}] := \mathsf{rsk}$; Return $\mathsf{rpk}$
  If the protocol fails, return $\bot$

$\underline{\mathsf{Reg}(\mathsf{rid})}^{\text{RP Accountability, RP Session Binding — Honest IdP, corrupt RP}}$
  If $(\mathsf{rid} \in \mathsf{HRID} \cup \mathsf{CRID})$ return $\bot$
  Run $\mathsf{Reg}(\mathsf{rid}, \mathcal{M})$ with $\mathcal{A}$ (being a corrupt RP)
  Upon output $\mathcal{M}'$
    $\mathcal{M} := \mathcal{M}'$; $\mathsf{CRID} := \mathsf{CRID} \cup \{\mathsf{rid}\}$
    Return 1
  If the protocol fails, return $\bot$

$\underline{\mathsf{CredReq\text{-}CredIss}(\mathsf{rid})}^{\text{RP Accountability — Honest IdP, honest RP}}$
  If $(\mathsf{rid} \notin \mathsf{HRID})$ return $\bot$
  $\mathsf{sid} \leftarrow_{\mathrm{R}} \mathcal{Z}$
  Run $\langle \mathsf{CredReq}(\mathsf{ipk}, \mathsf{rid}, \mathsf{HRID}[\mathsf{rid}], \mathsf{sid}, \mathsf{cep}), \mathsf{CredIss}(\mathsf{rid}, \mathsf{isk}, \mathcal{M}, \mathsf{sid}, \mathsf{cep}) \rangle$
  Upon output $(\mathsf{cred}, \mathcal{M}')$
    $\mathsf{HRID}[\mathsf{rid}, \mathsf{cep}] := \mathsf{cred}$; Return 1
  If the protocol fails, return $\bot$

$\underline{\mathsf{CredReq}(\mathsf{rid})}^{\text{RP Hiding — Corrupt IdP, honest RP}}$
  If $(\mathsf{rid} \notin \mathsf{HRID})$ return $\bot$
  $\mathsf{sid} \leftarrow_{\mathrm{R}} \mathcal{Z}$
  Run $\mathsf{CredReq}(\mathsf{ipk}, \mathsf{rid}, \mathsf{HRID}[\mathsf{rid}], \mathsf{sid}, \mathsf{cep})$ with $\mathcal{A}$ (being a corrupt IdP)
  Upon output $\mathsf{cred}$
    $\mathsf{HRID}[\mathsf{rid}, \mathsf{cep}] := \mathsf{cred}$; Return 1
  If the protocol fails, return $\bot$

$\underline{\mathsf{SetEP}}^{\text{RP Accountability, RP Session Binding}}$
  $\mathsf{cep} := \mathsf{cep} + 1$
```

```
$\underline{\mathsf{CredIss}(\mathsf{rid})}^{\text{RP Accountability, RP Session Binding — Honest IdP, corrupt RP}}$
  $\mathsf{sid} \leftarrow_{\mathrm{R}} \mathcal{Z}$
  Run $\mathsf{CredIss}(\mathsf{rid}, \mathsf{isk}, \mathcal{M}, \mathsf{sid}, \mathsf{cep})$ with $\mathcal{A}$ (in the role of the RP)
  Upon output $\mathcal{M}'$
    $\mathcal{M} := \mathcal{M}'$
    If $(\mathsf{rid} \in \mathsf{CRID})$ set $\mathsf{CRID}[\mathsf{cep}] := \mathsf{CRID}[\mathsf{cep}] \cup \{\mathsf{rid}\}$
    Return 1
  If the protocol fails, return $\bot$

$\underline{\mathsf{CrptRP}(\mathsf{rid})}^{\text{RP Accountability}}$
  If $(\mathsf{rid} \notin \mathsf{HRID})$ return $\bot$
  $\mathsf{CRID} := \mathsf{CRID} \cup \{\mathsf{rid}\}$
  $\mathsf{rsk} := \mathsf{HRID}[\mathsf{rid}]$; $\mathsf{cred} := \mathsf{HRID}[\mathsf{rid}, \mathsf{cep}]$
  If $(\mathsf{cred} \neq \epsilon)$ set $\mathsf{CRID}[\mathsf{cep}] := \mathsf{CRID}[\mathsf{cep}] \cup \{\mathsf{rid}\}$
  $\mathsf{HRID} := \mathsf{HRID} \setminus \{\mathsf{rid}\}$ // Removes all entries of rid in $\mathsf{HRID}[rid]$
  Return $(\mathsf{rsk}, \mathsf{cred})$

$\underline{\mathsf{AInit}(\mathsf{rid}, \mathsf{uid})}^{\text{RP Session Binding — Honest IdP, honest user}}$
  If $(\mathsf{uid} \in \mathsf{CUID})$ return $\bot$
  $\mathsf{HUID} := \mathsf{HUID} \cup \{\mathsf{uid}\}$
  $\mathsf{ses} := \mathsf{ses} + 1$; $(\mathsf{pub}_{\mathsf{U}}, \mathsf{priv}_{\mathsf{U}}) \leftarrow \mathsf{AInit}(\mathsf{ipk}, \mathsf{rid})$
  $\mathsf{SES}[\mathsf{ses}] := (\mathsf{rid}, \mathsf{uid}, \mathsf{pub}_{\mathsf{U}}, \mathsf{priv}_{\mathsf{U}})$
  Return $(\mathsf{ses}, \mathsf{pub}_{\mathsf{U}}, \mathsf{priv}_{\mathsf{U}})$

$\underline{\mathsf{AReqRP}(\mathsf{rid}, \mathsf{uid}, \mathsf{n}, \mathsf{pub}_{\mathsf{U}}, \mathsf{priv}_{\mathsf{U}})}^{\text{RP Accountability, RP Hiding — Honest RP}}$
  If $(\mathsf{rid} \notin \mathsf{HRID} \vee \mathsf{HRID}[\mathsf{rid}, \mathsf{cep}] = \epsilon)$ return $\bot$
  $\mathsf{REQ} := \mathsf{REQ} \cup \{(\mathsf{uid}, \mathsf{n}, \mathsf{pub}_{\mathsf{U}}, \mathsf{ep})\}$
  Return $\mathsf{auth}_{\mathsf{RP}} \leftarrow \mathsf{AReqRP}(\mathsf{ipk}, \mathsf{HRID}[\mathsf{rid}, \mathsf{cep}], \mathsf{uid}, \mathsf{pub}_{\mathsf{U}}, \mathsf{priv}_{\mathsf{U}}, \mathsf{n}, \mathsf{cep})$

$\underline{\mathsf{AResIdP}(\mathsf{uid}, \mathsf{n}, \mathsf{ctx}, \mathsf{auth}_{\mathsf{RP}}, \mathsf{pub}_{\mathsf{U}})}^{\text{RP Accountability, RP Session Binding — Honest IdP, corrupt user}}$
  If $(\mathsf{uid} \in \mathsf{HUID})$ return $\bot$
  $\mathsf{CUID} := \mathsf{CUID} \cup \{\mathsf{uid}\}$
  Return $\tau \leftarrow \mathsf{AResIdP}(\mathsf{isk}, \mathcal{M}, \mathsf{uid}, \mathsf{ctx}, \mathsf{auth}_{\mathsf{RP}}, \mathsf{pub}_{\mathsf{U}}, \mathsf{n}, \mathsf{cep})$

$\underline{\mathsf{AResIdP\text{-}AFin}(\mathsf{ses}, \mathsf{n}, \mathsf{ctx}, \mathsf{auth}_{\mathsf{RP}})}^{\text{RP Session Binding — Honest IdP, honest user}}$
  If $(\mathsf{SES}[\mathsf{ses}] = \epsilon)$ return $\bot$
  Parse $\mathsf{SES}[\mathsf{ses}]$ as $(\mathsf{rid}, \mathsf{uid}, \mathsf{pub}_{\mathsf{U}}, \mathsf{priv}_{\mathsf{U}})$
  $\mathsf{RES} := \mathsf{RES} \cup \{(\mathsf{rid}, \mathsf{uid}, \mathsf{n}, \mathsf{ctx}, \mathsf{cep})\}$
  $\tau \leftarrow \mathsf{AResIdP}(\mathsf{isk}, \mathcal{M}, \mathsf{uid}, \mathsf{ctx}, \mathsf{auth}_{\mathsf{RP}}, \mathsf{pub}_{\mathsf{U}}, \mathsf{n}, \mathsf{cep})$
  Return $\tau_{\mathsf{id}} \leftarrow \mathsf{AFin}(\mathsf{ipk}, \mathsf{rid}, \mathsf{uid}, \mathsf{ctx}, \mathsf{pub}_{\mathsf{U}}, \mathsf{priv}_{\mathsf{U}}, \mathsf{n}, \mathsf{cep}, \tau)$
```

Figure 6: Oracles for RP Accountability, RP Session Binding, and RP Hiding provided to adversaries in the experiments of Figure 7.

corrupt RPs via $\mathcal{O}$.Reg, and initialize the credential issuance for any RP with the IdP through $\mathcal{O}$.CredIss.

Definition 4.1 (RP Accountability). An AIF scheme provides RP Accountability if for all PPT adversaries $\mathcal{A}$ in the experiment stated in Figure 7, $\Pr[\mathrm{Exp}^{\mathsf{RP\ Accountability}}_{\mathcal{A},\mathsf{AIF}}(\kappa) = 1]$ is negligible in κ.

4.3 RP Session Binding

This property again considers security in a setting where the IdP is honest, but RPs are corrupt and aim to exploit the blind authentication to trick the IdP (and honest user) into wrongly authenticating for an unintended or even invalid RP. Whereas RP Accountability expressed the security guarantees for the RP's authentication $auth_{RP}$ towards the IdP, this notion is now for the *final* token τ_{id} and from the perspective of an honest user (and honest IdP).

This different perspective allows us to provide stronger security and complement RP Accountability. As users are always aware of the RP they want to authenticate to and, in particular, generate the finalized "unblinded" token τ_{id}, we can use that knowledge to express the exact context for which an authentication token was generated. We then require that each token τ_{id} is immutably bound to the same session intended for by the honest user, particularly to the rid of the designated — and legitimately authenticated — RP. This captures the following attacks, noting that security in our model guarantees the absence of *any* attack:

- An honest user wants to authenticate to a corrupt RP rid_1, which does *not* own the necessary credentials (either at all or in the current epoch) but tries to collude with another corrupt RP rid_2 that has the required credentials and wants to share them. It must be infeasible for rid_1 and rid_2 to get the IdP to issue a token for rid_1. This prevents *credential pooling*, which is particularly important when an IdP wants to offer its authentication as a *paid* service for RPs: blind authentication should not enable to bypass registration and allow a corrupt RP to operate as a proxy to other (corrupt) RPs.
- An honest user intends to authenticate to a corrupt RP rid_1, which possesses the required credentials but plans to misuse the issued token τ_{id} to impersonate the user with another RP rid_2. This scenario can be seen as a *phishing attack*, which must be prevented, despite the IdP not learning the RP's identity to which it binds its token.

Both attacks exploit that a corrupt RP has valid credentials for the epoch of the forgery, which was not allowed in the RP Accountability game. In fact, here we assume all RPs to be corrupt

```
Experiment: $\mathsf{Exp}^{\mathsf{RPAccountability}}_{\mathcal{A},\mathsf{AIF}}(\kappa)$:
  $pp \leftarrow \mathsf{Setup}(1^\kappa)$; $((isk, \mathcal{M}), ipk) \leftarrow \mathsf{SetupIdP}(pp)$
  $\mathcal{O} := \{\mathsf{Join\text{-}Reg}, \mathsf{Reg}, \mathsf{CredReq\text{-}CredIss}, \mathsf{CredIss}, \mathsf{CrptRP}, \mathsf{SetEP}, \mathsf{AReqRP}, \mathsf{AResIdP}\}$
  $(uid^*, ctx^*, {pub_U}^*, n^*, {auth_{RP}}^*) \leftarrow \mathcal{A}^{\mathcal{O}}(ipk)$
  Return 1 if
    $\mathsf{AResIdP}(isk, \mathcal{M}, uid^*, ctx^*, {auth_{RP}}^*, {pub_U}^*, n^*, cep) \neq \bot \wedge$
    $(uid^*, n^*, {pub_U}^*, cep) \notin \mathsf{REQ} \wedge |\mathsf{CRID}[cep]| = 0$

Experiment: $\mathsf{Exp}^{\mathsf{RPSession\ Binding}}_{\mathcal{A},\mathsf{AIF}}(\kappa)$:
  $pp \leftarrow \mathsf{Setup}(1^\kappa)$; $((isk, \mathcal{M}), ipk) \leftarrow \mathsf{SetupIdP}(pp)$
  $\mathcal{O} := \{\mathsf{Reg}, \mathsf{CredIss}, \mathsf{AInit}, \mathsf{SetEP}, \mathsf{AResIdP}, \mathsf{AResIdP\text{-}AFin}\}$
  $(rid^*, uid^*, ctx^*, n^*, {\tau_{id}}^*) \leftarrow \mathcal{A}^{\mathcal{O}}(ipk)$
  Return 1 if $\mathsf{Vf}(ipk, (rid^*, uid^*, ctx^*, n^*, cep), {\tau_{id}}^*) = 1 \wedge uid^* \notin \mathsf{CUID}$
     // and one of the two conditions is satisfied:
     (a) $(rid^*, uid^*, n^*, ctx^*, cep) \notin \mathsf{RES}$  // fresh session
     (b) $(rid^*, uid^*, n^*, ctx^*, cep) \in \mathsf{RES} \wedge rid^* \notin \mathsf{CRID}[cep]$ // invalid RP

Experiment: $\mathsf{Exp}^{\mathsf{RPHiding-b}}_{\mathcal{A},\mathsf{AIF}}(\kappa)$:
  $pp \leftarrow \mathsf{Setup}(1^\kappa)$; $((isk, \mathcal{M}), ipk) \leftarrow \mathsf{SetupIdP}(pp)$
  $\mathcal{O} := \{\mathsf{Join}, \mathsf{CredReq}, \mathsf{AReqRP}\}$
  $(rid_0, rid_1, uid, n, st) \leftarrow \mathcal{A}^{\mathcal{O}}(isk, ipk, \mathcal{M})$
  For $d \in \{0,1\}$: If $(rid_d \notin \mathsf{HRID} \vee \mathsf{HRID}[rid_d, cep] = \epsilon)$ abort
  $(priv_U, pub_U) \leftarrow \mathsf{AInit}(ipk, rid_b)$
  $auth_{RP} \leftarrow \mathsf{AReqRP}(ipk, \mathsf{HRID}[rid_b, cep], uid, pub_U, priv_U, n, cep)$
  Return $b^* \leftarrow \mathcal{A}^{\mathcal{O}}(st, auth_{RP}, pub_U)$
```

Figure 7: Experiments to define our AIF security properties.

for the sake of simplicity: honest RPs cannot give the adversary any advantage in winning the game and thus are omitted. This additional power of the adversary is possible as we capture the RP Session Binding property for *honest users*, allowing us to express the intended sessions.

The honest user(s) are modeled via the $\mathcal{O}$.AInit oracle that lets the adversary start a session for a *uid* towards an adversarially controlled *rid*. After a session is started, the adversary can contribute the RP authentication data $auth_{RP}$ and let the honest user finalize the session by using $\mathcal{O}$.AResIdP-AFin. Both oracles use common session information (that model the state an honest user keeps), referenced by a session identifier *ses*. This modeling of honest users allows us to keep track of the user's intended sessions, in particular the *rid*, which otherwise would be blinded in all values the honest IdP receives. The adversary can, of course, also request tokens for corrupt users via the $\mathcal{O}$.AResIdP oracle. However, we must ensure that the *uid*s passed to $\mathcal{O}$.AResIdP never appear in any query to $\mathcal{O}$.AInit nor $\mathcal{O}$.AResIdP-AFin. This is done by keeping track of honest/corrupt users in HUID/CUID, enabling queries to the $\mathcal{O}$.AResIdP oracle only for users in CUID.

Overall, security is again captured in the form of a unforgeability challenge, but here the task of the adversary is to output a valid token ${\tau_{id}}^*$ for a session $(rid^*, uid^*, ctx^*, n^*, cep)$ that verifies under the IdP's ipk. The forgery must be for an honest user uid^* and be non-trivial. Non-trivial either means that the token is used in a different context than what the honest user wanted and the IdP certified (winning condition (a) in the game in Figure 7, capturing the phishing attacks described above). Another possible win strategy is to get the user and IdP to generate a token for an *rid* that did not have valid credentials for the requested epoch. This attack is captured via our game's second winning condition (b) and reflects the credential pooling attack from above.

Definition 4.2 (RP Session Binding). An AIF scheme is RP Session Binding if for all PPT adversaries $\mathcal{A}$ in the experiment stated in Figure 7, $\Pr[\mathsf{Exp}^{\mathsf{RP\ Session\ Binding}}_{\mathcal{A},\mathsf{AIF}}(\kappa) = 1]$ is negligible in κ.

4.4 RP Hiding

The privacy guarantees provided by an AIF system, which made formalizing the two RP authentication properties challenging, are now captured in our final property that considers a corrupt IdP. This property requires that, despite the RP being properly authenticated towards the IdP, its identity *rid* remains hidden (*secrecy*). In fact, to prevent re-identification through access patterns or correlation attacks, the corrupt IdP must not even tell if a user repeatedly authenticates to the same (unknown) RP or to different ones, capturing *unlinkability*. This ensures that while the user wants to rely on the IdP for its authentication service, it does not want to expose its online behavior to the IdP.

This privacy guarantee is expressed in the standard indistinguishability style: the adversary knows the IdP's secret key *isk* and can trigger honest RPs to register via the $\mathcal{O}$.Join oracle and obtain authentication requests from such registered RPs for session information (mimicking honest users) of its choice. Eventually, $\mathcal{A}$ outputs two identities rid_0, rid_1 of honest RPs and common session information (uid, n) and receives an authentication request — comprising of $(auth_{RP}, pub_U)$ — generated for rid_b, where b is a randomly chosen bit. The task of the adversary is to determine b better than by guessing.

Definition 4.3 (RP Hiding). An AIF scheme is RP Hiding if for all PPT adversaries $\mathcal{A}$ and for the experiment stated in Figure 7, $|\Pr[\mathsf{Exp}^{\mathsf{RP\ Hiding-1}}_{\mathcal{A},\mathsf{AIF}}(\kappa) = 1] - \Pr[\mathsf{Exp}^{\mathsf{RP\ Hiding-0}}_{\mathcal{A},\mathsf{AIF}}(\kappa) = 1]|$ is negligible in κ.

5 CONSTRUCTIONS & ANALYSIS

This section first analyzes the two existing approaches, represented by $\mathsf{AIF}_{\mathsf{SIG}}$ and $\mathsf{AIF}_{\mathsf{COM}}$, and argues why neither satisfies the full set of security and privacy properties defined in the previous section. Both methods depend on standard signatures from the IdP to (partially) fulfill RP Session Binding. $\mathsf{AIF}_{\mathsf{SIG}}$ successfully attains RP Accountability and Session Binding through standard signature authentication but lacks RP Hiding. In contrast, $\mathsf{AIF}_{\mathsf{COM}}$ employs commitments to achieve RP Hiding by blindly binding the token to the RP's identity, but it falls short of providing any form of authentication. Our new construction, $\mathsf{AIF}_{\mathsf{ZKP}}$, fully implements our AIF model, leveraging multi-message signatures and commitments, and accomplishes all three AIF security properties.

5.1 Analysis of Existing Approaches

Our formal analysis must first translate the OIDC's Implicit Flow [41] and POIDC [24] to our syntax and algorithms. In the body of the paper, we focus on the core ideas and refer to the detailed mapping and analysis to App. C. In particular, for the sake of accessibility, we omit the part about epoch-based membership renewal in the following discussion summary, as realizing this renewal is technically simple but adds much complexity.

$\mathsf{AIF_{SIG}}$ *– full accountability, but no privacy.* $\mathsf{AIF_{SIG}}$, our adaption of OIDC's Implicit Flow, utilizes standard signatures for RP authentication to model the authenticated variant of OIDC's Implicit Flow [41], as outlined in Section 2.1. Figure 8 provides a simplified summary of this construction.

When an RP with rid registers, it creates a key pair (rsk, rpk) of a standard signature scheme, and the IdP stores the pair (rid, rpk). To authenticate, the RP signs the corresponding session with its rsk and sends the signature, along with its rid, to the IdP. The IdP verifies the signature by looking up the corresponding rpk for rid. If the verification is successful, the IdP generates an identity token by signing the rid, user, and session context. This protocol does not require any cryptographic operations from the user.

It is easy to see that the presence of strong signature-based RP authentication and the IdP's signature on the verified rid guarantees both authentication-related properties, as proven in App. C.1.

Theorem 5.1. $\mathsf{AIF_{SIG}}$ *achieves RP Accountability and RP Session Binding if* SIG *(used by the RP and IdP) is SIG-EUF-CMA secure.*

However, this construction cannot achieve RP Hiding, as each RP is identified by its rid and associated signature/public key.

$\mathsf{AIF_{COM}}$ *– full privacy, but no accountability.* $\mathsf{AIF_{COM}}$ represents POIDC [24] in our syntax and contains no RP registration or authentication. The simplified overview is depicted in Figure 9. The protocol uses a commitment scheme COM to hide the RP's rid in a commitment c towards the IdP while uniquely binding the IdP's token to rid by allowing the IdP to sign the committed value c. The user is privy to the opening of the commitment and can ensure that c contains the correct rid.

Due to the absence of any RP authentication, all the IdP learns about the RP is the commitment (but not the opening) of rid. Thus, RP Hiding follows trivially from the hiding property of COM.

$\mathsf{AIF_{COM}}$ also partially satisfies our notion of session-binding (if COM is binding and SIG unforgeable): the IdP's tokens are immutably bound to the session, which encompasses the rid intended by an honest user, thus meeting the first condition (a) of RP Session Binding. However, due to the absence of any RP authentication, $\mathsf{AIF_{COM}}$ cannot satisfy the second condition (b) of the session-binding property, which guarantees that the request stems from a *legitimate* RP. We call this weaker notion *Partial RP Session Binding* and provide the simple proof of the following theorem in App. C.2.

Theorem 5.2. $\mathsf{AIF_{COM}}$ *is partially RP Session Binding if* COM *is binding and* SIG *is SIG-EUF-CMA; and RP Hiding if* COM *is hiding.*

The strong privacy offered by $\mathsf{AIF_{COM}}$ comes with the drawback of sacrificing RP Accountability and *full* RP Session Binding. During the registration phase, only the rid is stored without any associated authentication, and the IdP blindly signs arbitrary rids without verification. An immediate idea to fix this lack of authentication is to let RP register a public key of a signature scheme, similar to $\mathsf{AIF_{SIG}}$, and to let the RP sign the commitment for authentication. While this would ensure RP Accountability, it would give up any privacy again. Furthermore, this modification would not be sufficient to attain full RP Session Binding, as the committed value must be strictly bound to the RP's identity – a feature that is not feasible with basic building blocks but is precisely what our new $\mathsf{AIF_{ZKP}}$ protocol achieves through the use of advanced primitives.

```
Setup & Registration
IdP : $(isk, ipk) \leftarrow \mathsf{SIG.KGen}(1^\kappa)$; RP : $(rsk, rpk) \leftarrow \mathsf{SIG.KGen}(1^\kappa)$
IdP : Registers RP as $(rid, rpk)$ in $\mathcal{M}$

User Init. & RP Authentication
AInit : Return $(pub_U := \epsilon, priv_U := \epsilon)$ // User does no cryptographic operations
AReqRP : Generate $\sigma \leftarrow \mathsf{SIG.Sign}(rsk, (uid||n))$; output $auth_{RP} := (rid, \sigma)$

Token Issuance, Finalization, Verification
AResIdP : Verify $(rid, rpk) \in \mathcal{M}$ and $\mathsf{SIG.Vf}(rpk, (uid||n), \sigma) = 1$
          Output $\tau \leftarrow \mathsf{SIG.Sign}(isk, (rid||uid||ctx||n))$
AFin : Output $\tau_{id} := \tau$
Vf : Return $\mathsf{SIG.Vf}(ipk, (rid||uid||ctx||n), \tau_{id}) = 1$
```

Figure 8: Simplified $\mathsf{AIF_{SIG}}$ based on OIDC's Implicit Flow [41]. The detailed construction incl. credential renewal is given in App. C.1.

```
Setup & Registration
IdP : $(isk, ipk) \leftarrow \mathsf{SIG.KGen}(1^\kappa)$; Register RP with $rid$ in $\mathcal{M}$

User Init. & RP Authentication
AInit : Create $(o, c) \leftarrow \mathsf{COM.Commit}(rid)$; output $(pub_U := c, priv_U := o)$
AReqRP : Output $auth_{RP} := \epsilon$ // No RP authentication

Token Issuance, Finalization, Verification
AResIdP : Output $\tau \leftarrow \mathsf{SIG.Sign}(isk, (c||uid||ctx||n))$
AFin : Verify $\mathsf{COM.Open}(rid, c, o) = 1$; output $\tau_{id} := (\tau, c, o)$
Vf : Return $\mathsf{SIG.Vf}(ipk, (c||uid||ctx||n), \tau) = 1$ and $\mathsf{COM.Open}(rid, c, o) = 1$
```

Figure 9: Simplified $\mathsf{AIF_{COM}}$ that translates POIDC [24] to our syntax. Full description is in App. C.2.

5.2 Our Fully Secure Scheme: AIF-ZKP

We now present $\mathsf{AIF_{ZKP}}$, our new construction that builds on $\mathsf{AIF_{COM}}$ and adds RP Accountability and Session Binding, becoming the first scheme to meet all properties outlined in Section 4. Our approach is rather simple: The RP registers with the public key of a standard signature scheme with the IdP, which it uses in every epoch ep to authenticate towards the IdP and obtain a new credential. The IdP-issued credential for epoch ep is a signature σ_{IdP} on (rid, ep), generated via the multi-message signature scheme MMS, which later allows the privacy-preserving authentication. If an RP is supposed to get revoked or has not renewed its membership, the IdP will not issue a new credential, allowing us to tolerate corrupt RPs and ensure RP Accountability and Session Binding.

To obtain an identity token for a specific RP, a user creates a commitment (c, o) on the RP's rid and lets the RP generate an authentication request, which proves knowledge of the IdP's MMS signature σ_{IdP} on (rid, ep). The proof thereby reveals ep but hides rid. To achieve the desired RP Session Binding, we must ensure that the commitment c contains the same rid that is signed in σ_{IdP}, which we let the RP prove in $auth_{RP}$ too. The IdP receives the commitment c on rid and the proof, allowing it to verify that the request originates from an RP with a valid credential in epoch ep without learning its identity — and most importantly, being sure that the blindly verified rid is strictly committed to in c. If the verification succeeds, the IdP generates a token τ, which is a signature on $(c||uid||ctx||n||ep)$ using the standard signature scheme SIG. The user finalizes the token τ by verifying the IdP's signature and using the opening o to confirm that the commitment c opens to the intended rid. Upon success, the user adds the opening to τ, which uniquely binds the IdP's signature to a specific RP. The detailed protocol is presented in Figure 10.

```
$\mathsf{SetupIdP}(pp) \rightarrow ((isk, \mathcal{M}), ipk)$
$(sk, pk) \leftarrow \mathsf{SIG.KGen}(1^\kappa)$; $(msk, mpk) \leftarrow \mathsf{MMS.KGen}(pp, \ell := 2)$
Return $((isk := (sk, msk), \mathcal{M} := \emptyset), ipk := (pk, mpk))$

$\langle \mathsf{Join}(ipk, rid), \mathsf{Reg}(rid, \mathcal{M}) \rangle \rightarrow \{((rsk, rpk), \mathcal{M}'), \perp\}$
RP : $(rsk, rpk) \leftarrow \mathsf{SIG.KGen}(1^\kappa)$; Send $(rid, rpk)$
IdP : If $(rid \in \mathcal{M} \lor rpk \in \mathcal{M})$ return $\perp$
    Else return $\mathcal{M}' := \mathcal{M}[rid] := rpk$
RP : Return $(rsk, rpk)$

$\langle \mathsf{CredReq}(ipk, rid, rsk, sid, ep),$
 $\mathsf{CredIss}(rid, isk, \mathcal{M}, sid, ep) \rangle \rightarrow \{(cred, \mathcal{M}'), \perp\}$
RP : Send $(rid, \sigma_{RP} \leftarrow \mathsf{SIG.Sign}(rsk, sid))$
IdP : Parse $\mathcal{M}[rid]$ as $rpk$, $isk$ as $(\cdot, msk)$
    If $(rid \notin \mathcal{M} \lor \mathsf{SIG.Vf}(rpk, sid, \sigma_{RP}) \neq 1)$ return $\perp$
    Send $\sigma_{IdP} \leftarrow \mathsf{MMS.Sign}(msk, (rid, ep))$; Return $\mathcal{M}' := \mathcal{M}$
RP : Return $cred := (\sigma_{IdP}, ep)$

$\mathsf{AInit}(ipk, rid) \rightarrow (priv_U, pub_U)$
$(c, o) \leftarrow \mathsf{COM.Commit}(rid)$; Return $(pub_U := c, priv_U := o)$

$\mathsf{AReqRP}(ipk, rid, cred, uid, pub_U, priv_U, n, ep) \rightarrow auth_{RP}$
Parse $cred$ as $(\sigma_{IdP}, ep')$, $ipk$ as $(\cdot, mpk)$, $pub_U$ as $c$, $priv_U$ as $o$
If $(ep \neq ep' \lor \mathsf{COM.Open}(rid, c, o) \neq 1)$ abort
Else return $auth_{RP} := \pi \leftarrow \mathsf{NIZK}\{(\sigma_{IdP}, rid, o) : \mathsf{COM.Open}(rid, c, o)$
    $\land\, \mathsf{MMS.Vf}(mpk, (rid, ep), \sigma_{IdP})\}(uid, n, c, ep)$

$\mathsf{AResIdP}(isk, \mathcal{M}, uid, ctx, auth_{RP}, pub_U, n, ep) \rightarrow \{\tau, \perp\}$
Parse $ipk$ as $(\cdot, mpk)$, $isk$ as $(sk, \cdot)$, $auth_{RP}$ as $\pi$, $pub_U$ as $c$
If ($\pi$ is not correct w.r.t. $(mpk, c, uid, n, ep)$) return $\perp$
Else return $\tau \leftarrow \mathsf{SIG.Sign}(sk, (c||uid||ctx||n||ep))$

$\mathsf{AFin}(ipk, rid, uid, ctx, pub_U, priv_U, n, ep, \tau) \rightarrow \{\tau_{id}, \perp\}$
Parse $ipk$ as $(pk, \cdot)$, $pub_U$ as $c$, $priv_U$ as $o$
If $(\mathsf{SIG.Vf}(pk, (c||uid||ctx||n||ep), \tau) \neq 1 \lor \mathsf{COM.Open}(rid, c, o) \neq 1)$
Return $\perp$ else return $\tau_{id} := (\tau, c, o)$

$\mathsf{Vf}(ipk, (rid, uid, ctx, n, ep), \tau_{id}) \rightarrow 0/1$
Parse $ipk$ as $(pk, \cdot)$, $\tau_{id}$ as $(\tau, c, o)$
Return $(\mathsf{SIG.Vf}(pk, (c||uid||ctx||n||ep), \tau) = \mathsf{COM.Open}(rid, c, o) = 1)$
```

Figure 10: $\mathsf{AIF}_{\mathsf{ZKP}}$ **– Our new construction.**

The scheme is initialized via $pp \leftarrow \mathsf{MMS.Setup}(1^\kappa)$, returning the public parameters pp that are an implicit input to all algorithms and also contain the security parameter 1^κ. In the concrete instantiation, this will contain the public groups and generators used by both the MMS signature and commitment scheme.

We now state and sketch the achieved security guarantees of our construction and refer for the full proofs to App. D.

Theorem 5.3. $\mathsf{AIF}_{\mathsf{ZKP}}$ *achieves RP Accountability if* SIG *(used by the RP) is SIG-EUF-CMA,* MMS *is MMS-EUF-CMA secure, and the* NIZK *is zero-knowledge as well as simulation-sound extractable.*

Proof Sketch. The adversary wins the accountability game if it outputs an authentication request $auth_{RP}{}^*$ for a fresh tuple $(uid^*, n^*, pub_U{}^*, cep)$ that is accepted by the IdP, while there are no corrupt RPs with a credential in epoch cep. In this construction, the authentication request $auth_{RP}{}^* := \pi$ is a NIZK proving knowledge of a credential $cred := (\sigma_{IdP}, cep)$, where the signature σ_{IdP} of such a credential is an IdP's MMS signature on (rid, cep).

There are two attack strategies for the adversary, which we both prove to be impossible based on the made assumption: First, $\mathcal{A}$ obtained a valid credential for cep by impersonating an honest RP rid towards the IdP during credential renewal. As the renewal requires $\mathcal{A}$ to authenticate via the honest RP's previously registered public key and standard signature for a fresh nonce, this is impossible based on the unforgeability of the underlying signature scheme SIG. The second strategy is that the adversary, without having obtained a valid MMS credential from IdP, manages to produce a convincing NIZK proof of such a credential. However, this either requires to forge the NIZK statement (which we assume to be infeasible) or forge the underlying MMS, which contradicts its unforgeability. □

Note that Theorem 5.3 is independent of the binding property of COM. Therefore, an adversary who breaks the binding property and produces a commitment that opens to a different rid' would still satisfy RP Accountability. However, such an adversary cannot satisfy RP Session Binding, which captures such stronger security for the final token.

Theorem 5.4. $\mathsf{AIF}_{\mathsf{ZKP}}$ *is RP Session Binding if* SIG *(used by the IdP) is SIG-EUF-CMA secure,* COM *is binding,* MMS *is MMS-EUF-CMA secure, and the* NIZK *is special sound.*

Proof Sketch. An adversary who succeeds in the RP Session Binding experiment outputs a finalized token $\tau_{id}{}^*$ that is valid for an honest user session $(rid^*, uid^*, ctx^*, n^*, cep)$. This session either has (a) to be fresh, i.e., it was never queried to $\mathcal{O}$.AResIdP-AFin, or (b) this session was intended by an honest user, but then rid^* must belong to some RP that does not own a credential in epoch cep. Recall that the finalized token $\tau_{id}{}^* := (\tau, c, o)$ must contain an IdP's SIG signature τ on $(c||uid^*||ctx^*||n^*||cep)$ and a correct opening o for the commitment c to rid^*.

There are two cases in which $\mathcal{A}$ can win under condition (a): either the "public" session part (uid^*, ctx^*, n^*, cep) is fresh and $\mathcal{A}$ forges an IdP's signature, or that session part is not fresh, which in turn means that rid^* must be different than in the honest query to the $\mathcal{O}$.AInit oracle, which breaks the binding property of COM.

To succeed under the second winning condition (b), i.e., the honest user session existed, but rid^* did not belong to an RP with a credential for epoch cep, $\mathcal{A}$ must have either forged the NIZK proof or forged the MMS credential. □

Theorem 5.5. $\mathsf{AIF}_{\mathsf{ZKP}}$ *is RP Hiding if* COM *is hiding and the* NIZK *is zero-knowledge.*

Proof Sketch. The authentication request $auth_{RP} := \pi$ proves knowledge of a signature on the committed identity rid_b in a commitment c. RP Hiding follows from the zero-knowledge property of π and the hiding property of COM since $\mathcal{A}$ never learns o_b. □

6 EFFICIENCY & DISCUSSION

This section details the implementation of our $\mathsf{AIF}_{\mathsf{ZKP}}$ protocol and compares its efficiency with the two existing protocols. We conclude with a discussion of some deployment challenges when using a privacy-preserving AIF protocol such as $\mathsf{AIF}_{\mathsf{ZKP}}$ or $\mathsf{AIF}_{\mathsf{COM}}$.

6.1 Implementation

We provide a JavaScript (JS) implementation of our $\mathsf{AIF}_{\mathsf{ZKP}}$ scheme on GitHub [22]. For cryptographic operations, we rely on the MCL library [33] that we used to implement COM, MMS, and the NIZK.

Ref. Env.	Low-Power Device		Server			
Entity	User		RP		IdP	
IF\Alg.	AInit	AFin	AReqRP	Vf	CredIss	AResIdP
$\mathsf{AIF}_{\mathsf{SIG}}$	0.00	7.15	1.23	0.04	0.05	1.39
$\mathsf{AIF}_{\mathsf{COM}}$	6.04	9.74	0.00	0.44	0.00	1.27
$\mathsf{AIF}_{\mathsf{ZKP}}$	6.56	9.28	8.92	0.87	2.79	18.98

Table 2: The mean of one hundred executions in milliseconds.

Eff.\Alg.	AInit$^{\star\bullet}$	AFin$^{\star\bullet}$	AReqRP$^{\bullet}$	Vf$^{\star\bullet}$	CredIss$^{\bullet}$	AResIdP$^{\bullet}$
Size	$2\mathbb{G}_1$	$2\mathbb{G}_1$	$6\mathbb{G}_1+1\mathbb{G}_T$	$2\mathbb{G}_1$	$2\mathbb{G}_1$	-
Cost	$2\mathsf{E}_{\mathbb{G}_1}$	$2\mathsf{E}_{\mathbb{G}_1}$	$7\mathsf{E}_{\mathbb{G}_1}+2\mathsf{E}_{\mathbb{G}_2}$ $+ 1\mathsf{P}$	$2\mathsf{E}_{\mathbb{G}_1}$	$5\mathsf{E}_{\mathbb{G}_1}$	$5\mathsf{E}_{\mathbb{G}_1} + 3\mathsf{E}_{\mathbb{G}_2}$ $+ 3\mathsf{P}$

Table 3: Sizes/costs of $\mathsf{AIF}_{\mathsf{COM}}{}^{\star}$ and $\mathsf{AIF}_{\mathsf{ZKP}}{}^{\bullet}$ – $\mathsf{E}_{\mathbb{G}_i}$ refers to the cost of an exponentiation in $\mathbb{G}_i (i \in \{1, 2\})$ and P to the cost of a pairing.

Cryptographic instantiations. For SIG, we used a standard implementation of RSA [21] with 2048 bit to conform to current industry standards. For MMS, we implemented PS signatures [38] and Pedersen commitments [37] for COM. App. E details the concrete NIZK instantiation of the signatures with committed messages, favoring exponentiations in $\mathbb{G}_1$ and $\mathbb{G}_2$ to minimize the number of pairings for efficiency. As an elliptic curve, we used BLS12 − 381 [9], which provides $\kappa := 128$ bit security. Thus, the public parameters pp encompass the bilinear group description $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$, also detailed in App. E, and the fixed generators $(g, h) \in \mathbb{G}_1^2$.

Performance evaluation. We seek to examine the overhead of all the repetitive operations involved in authenticating a user to an RP via the IdP. To this end, we exclude the protocol setup and RP registration $(\mathsf{Setup}, \mathsf{SetupIdP}, \langle\mathsf{Join}, \mathsf{Reg}\rangle)$ from our analysis. Additionally, we do not consider the RP-side in the renewal protocol $\langle\mathsf{CredReq}, \mathsf{CredIss}\rangle$ as it only involves one RSA signature towards the IdP. However, we include the CredIss algorithm, reflecting the cost of the IdP renewing a MMS-based credential.

We evaluated the execution times of the cryptographic operations on two reference devices: a server, which is a virtual machine[1], and a Raspberry Pi 3[2], which represents a low-power device. We run the IdP/RP operations on the server and the user operations on the low-power device. Table 2 summarizes our results, showing the mean of one hundred executions in milliseconds (ms) for each selected algorithm. Table 3 presents the group elements and their computational costs in the executed algorithms. We note that an element in $(\mathbb{Z}_p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ requires (32, 48, 92, 576) bytes. Thus, the proof generated in AReqRP requires only 816 bytes.

Clearly, our new protocol $\mathsf{AIF}_{\mathsf{ZKP}}$ is the most expensive of the three, but it is also the only one that achieves the desired security and privacy properties simultaneously. Overall, all its operations are still efficient enough for real-world deployment, with at most 10ms on the low-power device and 19ms on the server for proof verification, which took only 9ms to generate. Our evaluation of $\mathsf{AIF}_{\mathsf{COM}}$ (POIDC) must be taken with the same grain of salt though: for simplicity, we implemented $\mathsf{AIF}_{\mathsf{COM}}$ and $\mathsf{AIF}_{\mathsf{ZKP}}$ with the same *algebraic* commitment. This is necessary for our protocol but not for $\mathsf{AIF}_{\mathsf{COM}}$, which could simply use a cryptographic hash function for COM, as originally proposed [24].

6.2 Deployment Considerations

Privacy-preserving RP authentication prevents the IdP from sharing RP-related information, including names and legal details that may be provided to the user during the protocol. Note that this also eliminates the implicit authentication of such information once it is no longer served by the IdP.

User interaction and redirections. The RP information is required in the consent dialog with the user and for operational purposes, such as redirecting the user back from the IdP to the RP to provide the issued token. To maintain privacy, such information should be available in the IdP's context without being disclosed again. In our approach, the RP can directly provide this information to the user by including it in the URL fragment [31] of the initial request, allowing access in the IdP's context without disclosure.

RP information authenticity. To ensure authenticity of operational information passed directly to the user by the RP, a hash of the information along with the IdP's signature must be included in the *rid*. The user must re-hash the RP information and verify it against the *rid* and IdP's signature before finalizing a token, thereby ensuring that the RP information corresponds to the requested RP.

IdP services. An IdP might want to learn the *rid* to offer certain commercial services for it, such as orchestrating additional computational resources to ensure better availability. To provide such tailored services in our privacy-preserving setting, an IdP could create dedicated MMS public keys for each service class, or add an additional attribute to the MMS credential that must be revealed during authentication. As a result, an RP can prove to the IdP that it is has a membership for a certain service class, while the rest of our security and privacy properties remain.

Pairwise pseudonymous identifier. A challenge is the support of OIDC's protocol feature of Pairwise Pseudonymous Identifier (PPID). This privacy feature allows the IdP to replace the static *uid* with an RP-specific pseudonym to prevent RPs from correlating users. This is no longer possible with our protocol and, in fact, an advantage of the work by Hammann et al. [24]. While POIDC does not support this feature either, Hammann et al. also propose a protocol extension — Pairwise POIDC — that lets the user and IdP blindly derive such pseudonyms. Their approach requires the user to provide a zero-knowledge proof of a blindly computed pseudonym of the form $\mathsf{H}(uid, rid)$ to the IdP for attestation. This extension again does not consider any RP authentication and thus does not provide RP Accountability and Session Binding.

Their extension uses a hash function that is not immediately compatible with the algebraic construction of our protocol, and the challenge would again be to incorporate RP authentication, which we leave as an interesting open problem. We did not include it as it would significantly complicate our model and analysis. Our work focuses solely on RP authentication. Considering also RP-specific pseudonyms would require including user authentication in the security model. Given our model's complexity, we decided to focus on the main problem and show how the OIDC core functionality can be realized while overcoming the reason for deprecation.

[1] CPU: AMD 4x2.6 GHz, RAM: 8 GB [2] CPU: ARM 4x1.2 GHz, RAM: 1 GB

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable feedback throughout the review process. This research was partially funded by the HPI Research School on Data Science and Engineering.

REFERENCES

[1] Shashank Agrawal, Peihan Miao, Payman Mohassel, and Pratyay Mukherjee. 2018. PASTA: PASsword-based Threshold Authentication. In *ACM CCS 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM Press, Toronto, ON, Canada, 2042–2059. https://doi.org/10.1145/3243734.3243839

[2] Muhammad Rizwan Asghar, Michael Backes, and Milivoj Simeonovski. 2018. PRIMA: Privacy-Preserving Identity and Access Management at Internet-Scale. In *2018 IEEE International Conference on Communications (ICC)*. IEEE, Kansas, USA, 1–6. https://doi.org/10.1109/ICC.2018.8422732

[3] Ramakrishna Ayyagari. 2012. An exploratory analysis of data breaches from 2005-2011: Trends and insights. *Journal of Information Privacy and Security* 8, 2 (2012), 33–56.

[4] Niko Bari and Birgit Pfitzmann. 1997. Collision-Free Accumulators and Fail-Stop Signature Schemes Without Trees. In *EUROCRYPT'97 (LNCS, Vol. 1233)*, Walter Fumy (Ed.). Springer, Heidelberg, Germany, Konstanz, Germany, 480–494. https://doi.org/10.1007/3-540-69053-0_33

[5] Carsten Baum, Tore Frederiksen, Julia Hesse, Anja Lehmann, and Avishay Yanai. 2020. PESTO: Proactively Secure Distributed Single Sign-On, or How to Trust a Hacked Server. In *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, Virtual, 587–606. https://doi.org/10.1109/EuroSP48549.2020.00044

[6] Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. 2008. P-signatures and Noninteractive Anonymous Credentials. In *TCC 2008 (LNCS, Vol. 4948)*, Ran Canetti (Ed.). Springer, Heidelberg, Germany, San Francisco, CA, USA, 356–374. https://doi.org/10.1007/978-3-540-78524-8_20

[7] Mihir Bellare, Haixia Shi, and Chong Zhang. 2005. Foundations of Group Signatures: The Case of Dynamic Groups. In *CT-RSA 2005 (LNCS, Vol. 3376)*, Alfred Menezes (Ed.). Springer, Heidelberg, Germany, San Francisco, CA, USA, 136–153. https://doi.org/10.1007/978-3-540-30574-3_11

[8] Josh Cohen Benaloh and Michael de Mare. 1994. One-Way Accumulators: A Decentralized Alternative to Digital Sinatures (Extended Abstract). In *EUROCRYPT'93 (LNCS, Vol. 765)*, Tor Helleseth (Ed.). Springer, Heidelberg, Germany, Lofthus, Norway, 274–285. https://doi.org/10.1007/3-540-48285-7_24

[9] Sean Bowe. 2017. *BLS12-381: New zk-SNARK elliptic curve construction.* Electric Coin. https://electriccoin.co/blog/new-snark-curve/

[10] Jan Camenisch, Manu Drijvers, and Jan Hajny. 2016. Scalable Revocation Scheme for Anonymous Credentials Based on N-Times Unlinkable Proofs. In *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society* (Vienna, Austria) *(WPES '16)*. Association for Computing Machinery, New York, NY, USA, 123–133. https://doi.org/10.1145/2994620.2994625

[11] Jan Camenisch, Manu Drijvers, Anja Lehmann, Gregory Neven, and Patrick Towa. 2020. Zone Encryption with Anonymous Authentication for V2V Communication. In *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, Genoa, Italy, 405–424. https://doi.org/10.1109/EuroSP48549.2020.00033

[12] Jan Camenisch, Aggelos Kiayias, and Moti Yung. 2009. On the Portability of Generalized Schnorr Proofs. In *EUROCRYPT 2009 (LNCS, Vol. 5479)*, Antoine Joux (Ed.). Springer, Heidelberg, Germany, Cologne, Germany, 425–442. https://doi.org/10.1007/978-3-642-01001-9_25

[13] Jan Camenisch and Anna Lysyanskaya. 2002. Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials. In *CRYPTO 2002 (LNCS, Vol. 2442)*, Moti Yung (Ed.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 61–76. https://doi.org/10.1007/3-540-45708-9_5

[14] Melissa Chase and Anna Lysyanskaya. 2006. On Signatures of Knowledge. In *CRYPTO 2006 (LNCS, Vol. 4117)*, Cynthia Dwork (Ed.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 78–96. https://doi.org/10.1007/11818175_5

[15] Arkajit Dey and Stephen Weis. 2010. PseudoID: Enhancing Privacy in Federated Login. In *Hot Topics in Privacy Enhancing Technologies*. Sciendo, Berlin, Germany, 95–107.

[16] Jesus Diaz and Anja Lehmann. 2021. Group Signatures with User-Controlled and Sequential Linkability. In *PKC 2021, Part I (LNCS, Vol. 12710)*, Juan Garay (Ed.). Springer, Heidelberg, Germany, Virtual Event, 360–388. https://doi.org/10.1007/978-3-030-75245-3_14

[17] Developer Documentation. 2020. *Transferring Your Apps and Users to Another Team.* Apple. https://developer.apple.com/documentation/sign_in_with_apple/transferring_your_apps_and_users_to_another_team

[18] Facebook. 2021. *OpenID Connect.* Facebook. https://developers.facebook.com/docs/facebook-login

[19] Daniel Fett, Ralf Küsters, and Guido Schmitz. 2015. SPRESSO: A Secure, Privacy-Respecting Single Sign-On System for the Web. In *ACM CCS 2015*, Indrajit Ray, Ninghui Li, and Christopher Kruegel (Eds.). ACM Press, Denver, CO, USA, 1358–1369. https://doi.org/10.1145/2810103.2813726

[20] Amos Fiat and Adi Shamir. 1987. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *CRYPTO'86 (LNCS, Vol. 263)*, Andrew M. Odlyzko (Ed.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 186–194. https://doi.org/10.1007/3-540-47721-7_12

[21] GitHub. 2020. *Node-RSA.* GitHub. https://github.com/rzcoder/node-rsa

[22] GitHub. 2023. *Authenticated Implicit Flow.* GitHub. https://github.com/hpicrypto/aif

[23] Jens Groth. 2006. Simulation-Sound NIZK Proofs for a Practical Language and Constant Size Group Signatures. In *ASIACRYPT 2006 (LNCS, Vol. 4284)*, Xuejia Lai and Kefei Chen (Eds.). Springer, Heidelberg, Germany, Shanghai, China, 444–459. https://doi.org/10.1007/11935230_29

[24] Sven Hammann, Ralf Sasse, and David A. Basin. 2020. Privacy-Preserving OpenID Connect. In *ASIACCS 20*, Hung-Min Sun, Shiuh-Pyng Shieh, Guofei Gu, and Giuseppe Ateniese (Eds.). ACM Press, Taipei, Taiwan, 277–289. https://doi.org/10.1145/3320269.3384724

[25] S.M. Taiabul Haque, Matthew Wright, and Shannon Scielzo. 2013. A Study of User Password Strategy for Multiple Accounts. In *Proceedings of the Third ACM Conference on Data and Application Security and Privacy* (San Antonio, Texas, USA) *(CODASPY '13)*. Association for Computing Machinery, New York, NY, USA, 173–176. https://doi.org/10.1145/2435349.2435373

[26] Dick Hardt. 2012. *The OAuth 2.0 Authorization Framework.* RFC 6749. RFC Editor. http://www.rfc-editor.org/rfc/rfc6749.txt

[27] Dick Hardt, Aaron Parecki, and Torsten Lodderstedt. 2023. *The OAuth 2.1 Authorization Framework.* Internet-Draft draft-ietf-oauth-v2-1. IETF Secretariat. https://datatracker.ietf.org/doc/draft-ietf-oauth-v2-1/

[28] Michael Jones, John Bradley, and Nat Sakimura. 2015. *JSON Web Token (JWT).* RFC 7519. RFC Editor. http://www.rfc-editor.org/rfc/rfc7519.txt

[29] Jorn Lapon, Markulf Kohlweiss, Bart De Decker, and Vincent Naessens. 2011. Analysis of Revocation Strategies for Anonymous Idemix Credentials. In *Communications and Multimedia Security*, Bart De Decker, Jorn Lapon, Vincent Naessens, and Andreas Uhl (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 3–17.

[30] Torsten Lodderstedt, John Bradley, Andrey Labunets, and Daniel Fett. 2023. *OAuth 2.0 Security Best Current Practice.* Internet-Draft draft-ietf-oauth-security-topics. IETF Secretariat. https://datatracker.ietf.org/doc/draft-ietf-oauth-security-topics/

[31] Larry Masinter, Tim Berners-Lee, and Roy T Fielding. 2005. *Uniform Resource Identifier (URI): Generic Syntax.* STD 66. RFC Editor. http://www.rfc-editor.org/rfc/rfc3986.txt

[32] Microsoft. 2021. *Microsoft identity platform and OpenID Connect protocol.* Microsoft. https://docs.microsoft.com/en-us/azure/active-directory/develop/v2-protocols-oidc

[33] Shigeo Mitsunari. 2023. *A portable and fast pairing-based cryptography library.* GitHub. https://github.com/herumi/mcl

[34] Mozilla. 2023. *Persona.* GitHub. https://github.com/mozilla/persona

[35] Lan Nguyen. 2005. Accumulators from Bilinear Pairings and Applications. In *CT-RSA 2005 (LNCS, Vol. 3376)*, Alfred Menezes (Ed.). Springer, Heidelberg, Germany, San Francisco, CA, USA, 275–292. https://doi.org/10.1007/978-3-540-30574-3_19

[36] Sarah Pearman, Jeremy Thomas, Pardis Emami Naeini, Hana Habib, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, Serge Egelman, and Alain Forget. 2017. Let's Go in for a Closer Look: Observing Passwords in Their Natural Habitat. In *ACM CCS 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM Press, Dallas, TX, USA, 295–310. https://doi.org/10.1145/3133956.3133973

[37] Torben P. Pedersen. 1992. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *CRYPTO'91 (LNCS, Vol. 576)*, Joan Feigenbaum (Ed.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 129–140. https://doi.org/10.1007/3-540-46766-1_9

[38] David Pointcheval and Olivier Sanders. 2016. Short Randomizable Signatures. In *CT-RSA 2016 (LNCS, Vol. 9610)*, Kazue Sako (Ed.). Springer, Heidelberg, Germany, San Francisco, CA, USA, 111–126. https://doi.org/10.1007/978-3-319-29485-8_7

[39] David Recordon and Drummond Reed. 2006. OpenID 2.0: A Platform for User-Centric Identity Management. In *Proceedings of the Second ACM Workshop on Digital Identity Management* (Alexandria, Virginia, USA) *(DIM '06)*. Association for Computing Machinery, New York, NY, USA, 11–16. https://doi.org/10.1145/1179529.1179532

[40] Nat Sakimura, John Bradley, and Michael Jones. 2023. *The OAuth 2.0 Authorization Framework: JWT-Secured Authorization Request (JAR).* RFC 9101. IETF.

[41] Nat Sakimura, John Bradley, Michael Jones, B. de Medeirosk, and C. Mortimore. 2014. *OpenID Connect Core 1.0.* OpenID Foundation. https://openid.net/specs/openid-connect-core-1_0.html

[42] Zhiyi Zhang, Michał Król, Alberto Sonnino, Lixia Zhang, and Etienne Rivière. 2021. EL PASSO: Efficient and Lightweight Privacy-preserving Single Sign On. *Proceedings on Privacy Enhancing Technologies* 2021, 2 (2021), 70–87.

A UNFORGEABILITY DEFINITIONS

This section provides the game-based Existential Unforgeability under Chosen Message Attack (EUF-CMA) definitions for a standard signature scheme SIG and a multi-message signature scheme MMS.

Standard signature scheme. An adversary $\mathcal{A}$ attacking SIG wins the experiment in Figure 11 if it creates a valid signature σ^* w.r.t. pk on a fresh message m^*, which was not queried to the Sign-oracle before. SIG is secure if no efficient adversary can succeed in the SIG-EUF-CMA experiment with non-negligible probability.

Definition A.1. A signature scheme $\mathsf{SIG} := (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ is existentially unforgeable under an adaptive chosen-message attack if for all probabilistic polynomial-time adversaries $\mathcal{A}$, there is a negligible function negl such that:

$$\Pr[\mathsf{Exp}^{\mathsf{SIG-EUF-CMA}}_{\mathcal{A},\mathsf{SIG}}(\kappa) = 1] \leq \mathsf{negl}(\kappa)$$

```
***Experiment***: $\mathsf{Exp}^{\mathsf{SIG-EUF-CMA}}_{\mathcal{A},\mathsf{SIG}}(\kappa)$
  $(sk, pk) \leftarrow \mathsf{SIG.KGen}(1^\kappa)$
  $(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{SIG.Sign}(sk,\cdot)}(pk)$
  Return 1 iff
    $\mathsf{SIG.Vf}(pk, m^*, \sigma^*) = 1 \land \mathcal{A}$ did not query $m^*$ to SIG.Sign
```

Figure 11: The SIG-EUF-CMA experiment

Multi-message signature scheme. The next experiment in Figure 12 is almost equivalent to the previous but introduces the message vector dimension $\ell \in \mathbb{N}$ required by the scheme. An adversary $\mathcal{A}$ with non-negligible advantage can forge a valid signature σ^* on a fresh message vector $\vec{m}^*$ w.r.t. pk, which was not queried to the Sign-oracle before. MMS is secure if no efficient adversary can succeed in the MMS-EUF-CMA experiment with non-negligible probability.

Definition A.2. A multi-message signature scheme $\mathsf{MMS} := (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ is existentially unforgeable under an adaptive chosen-message attack if for all probabilistic polynomial-time adversaries $\mathcal{A}$, there is a negligible function negl such that:

$$\Pr[\mathsf{Exp}^{\mathsf{MMS-EUF-CMA}}_{\mathcal{A},\mathsf{MMS},\ell}(\kappa) = 1] \leq \mathsf{negl}(\kappa)$$

```
***Experiment***: $\mathsf{Exp}^{\mathsf{MMS-EUF-CMA}}_{\mathcal{A},\mathsf{MMS},\ell}(\kappa)$:
  $pp \leftarrow \mathsf{MMS.Setup}(1^\kappa)$
  $(sk, pk) \leftarrow \mathsf{MMS.KGen}(pp, \ell)$
  $(\vec{m}^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{MMS.Sign}(sk,\cdot)}(pk)$
  Return 1 iff
    $\mathsf{MMS.Vf}(pk, \vec{m}^*, \sigma^*) = 1 \land \mathcal{A}$ did not query $\vec{m}^*$ to MMS.Sign
```

Figure 12: The MMS-EUF-CMA experiment.

B AIF CORRECTNESS

We define the set of registered users as $\mathcal{U}$ and the set of RPs as $\mathcal{R}$ before presenting the correctness of our Authenticated Implicit Flow scheme AIF. Recall that $\mathcal{T}$ represents the set of epochs and $\mathcal{Z}$ the set of nonces.

For AIF to be considered *correct*, it must hold for all $\kappa \in \mathbb{N}$, $pp \leftarrow \mathsf{Setup}(1^\kappa)$, $((isk, \mathcal{M}), ipk) \leftarrow \mathsf{SetupIdP}(pp)$, $rid \in \mathcal{R}$, $uid \in \mathcal{U}$, $\{sid, n\} \in \mathcal{Z}$, and context ctx in epoch $ep \in \mathcal{T}$:

$$\begin{aligned}
((rpk, rsk), \mathcal{M}') &\leftarrow \langle \mathsf{Join}(ipk, rid), \mathsf{Reg}(rid, \mathcal{M}) \rangle \\
(cred, \mathcal{M}') &\leftarrow \langle \mathsf{CredReq}(ipk, rid, rsk, sid, ep), \\
&\qquad \mathsf{CredIss}(rid, isk, \mathcal{M}, sid, ep) \rangle \\
(pub_U, priv_U) &\leftarrow \mathsf{AInit}(ipk, rid) \\
auth_{RP} &\leftarrow \mathsf{AReqRP}(ipk, rid, cred, uid, pub_U, priv_U, n, ep) \\
\tau &\leftarrow \mathsf{AResIdP}(isk, \mathcal{M}, uid, ctx, auth_{RP}, pub_U, n, ep) \\
\tau_{id} &\leftarrow \mathsf{AFin}(ipk, rid, uid, ctx, pub_U, priv_U, n, ep, \tau) \\
&\mathsf{Vf}(ipk, (rid, uid, ctx, n, ep), \tau_{id}) = 1.
\end{aligned}$$

C CONSTRUCTION & ANALYSIS: AIF-SIG/COM

This section analyzes two constructions that implement our AIF model: $\mathsf{AIF}_{\mathsf{SIG}}$ and $\mathsf{AIF}_{\mathsf{COM}}$. These constructions are adapted from existing proposals to suit our AIF setting and are designed to enhance the native Implicit Flow.

C.1 Achieving RP Authentication: AIF-SIG

As previously stated, $\mathsf{AIF}_{\mathsf{SIG}}$ is based on the authenticated version of OIDC's Implicit Flow [41], which incorporates standard signatures for RP authentication, as described in Section 2.1. Figure 13 details this construction.

To register, the RP with rid generates a key pair (rpk, rsk), the IdP adds (rid, rpk) to its member state $\mathcal{M}$, and the RP keeps rsk. As our system models *epoch-based* membership credentials, we must also realize this. The simplest way is to let the RP authenticate with rsk to the IdP in each new epoch ep by signing a fresh nonce. If the authentication is successful, and the RP is supposed to remain a valid member in epoch ep, the IdP updates its member state for rid, rpk to contain ep.

During RP authentication, the user does not perform any cryptographic operation. The RP signs the uid, the session nonce n, and (empty) public user output pub_U using rsk. The signature and RP identity rid are sent via the user to the IdP, which accepts the request if the signature is valid and the rpk is contained in $\mathcal{M}$ for rid in epoch ep. The provided token is the IdP's signature on $(rid||uid||ctx||n||ep)$. As we let the RP use the same signing key rsk for two different purposes, we ensure domain separation of the signed content by prefixing messages with "0" during credential renewal and "1" for authentication requests.

This construction easily satisfies both authentication-related properties, as stated in Theorem 5.1. The theorem establishes that $\mathsf{AIF}_{\mathsf{SIG}}$ fulfills RP Accountability and RP Session Binding if SIG is unforgeable.

We start with RP Accountability which relies on the unforgeability of the RP's signature scheme and the fact that the IdP only accepts authentication requests of RPs that have previously properly authenticated (through "credential renewal") in epoch cep.

Theorem C.1. *$\mathsf{AIF}_{\mathsf{SIG}}$ achieves RP Accountability if* SIG *(used by the RP) is SIG-EUF-CMA secure.*

Proof Sketch. The task of the adversary is to produce a forgery $auth_{RP}{}^*$ for a session $(uid^*, ctx^*, pub_U{}^*, n^*, cep)$ that is non-trivial

```
$\mathsf{SetupIdP}(pp) \rightarrow ((isk, \mathcal{M}), ipk)$
$(sk, pk) \leftarrow \mathsf{SIG.KGen}(1^\kappa); isk := sk; ipk := pk$
Return $((isk, \mathcal{M} := \emptyset), ipk)$

$\langle \mathsf{Join}(ipk, rid), \mathsf{Reg}(rid, \mathcal{M}) \rangle \rightarrow \{((rsk, rpk), \mathcal{M}'), \perp\}$
RP : $(rsk, rpk) \leftarrow \mathsf{SIG.KGen}(1^\kappa)$; Send $(rid, rpk)$
IdP : If $(rid \in \mathcal{M} \vee rpk \in \mathcal{M})$ return $\perp$
      Else return $\mathcal{M}' := \mathcal{M}[rid] := rpk$
RP : Return $(rsk, rpk)$

$\langle \mathsf{CredReq}(ipk, rid, rsk, sid, ep),$
  $\mathsf{CredIss}(rid, isk, \mathcal{M}, sid, ep) \rangle \rightarrow \{(cred, \mathcal{M}'), \perp\}$
RP : $\sigma_0 \leftarrow \mathsf{SIG.Sign}(rsk, (0||sid)$; Send $(rid, \sigma_0)$
IdP : Parse $\mathcal{M}[rid]$ as $(rpk, \cdot)$, abort if $rid \notin \mathcal{M}$
      If $(\mathsf{SIG.Vf}(rpk, (0||sid), \sigma_0) \neq 1)$ return $\perp$
      Else return $\mathcal{M}' := \mathcal{M}[rid] := (rpk, ep)$
RP : Return $cred := (rsk, ep)$

$\mathsf{AInit}(ipk, rid) \rightarrow (priv_U, pub_U)$
Return $(pub_U := \epsilon, priv_U := \epsilon)$

$\mathsf{AReqRP}(ipk, rid, cred, uid, pub_U, priv_U, n, ep) \rightarrow auth_{RP}$
Parse $cred$ as $(rsk, ep')$
If $(ep \neq ep' \vee pub_U \neq \epsilon)$ abort
$\sigma_1 \leftarrow \mathsf{SIG.Sign}(rsk, (1||uid||n||pub_U||ep))$
Return $auth_{RP} := (rid, \sigma_1)$

$\mathsf{AResIdP}(isk, \mathcal{M}, uid, ctx, auth_{RP}, pub_U, n, ep) \rightarrow \{\tau, \perp\}$
Parse $auth_{RP}$ as $(rid, \sigma_1)$, $\mathcal{M}[rid]$ as $(rpk, ep')$
If $(\mathsf{SIG.Vf}(rpk, (1||uid||n||pub_U||ep), \sigma_1) \neq 1 \vee ep \neq ep')$ return $\perp$
Else return $\tau \leftarrow \mathsf{SIG.Sign}(isk, (rid||uid||ctx||n||ep))$

$\mathsf{AFin}(ipk, rid, uid, ctx, pub_U, priv_U, n, ep, \tau) \rightarrow \{\tau_{id}, \perp\}$
If $(\mathsf{SIG.Vf}(ipk, (rid||uid||ctx||n||ep), \tau) \neq 1)$ return $\perp$
Else return $\tau_{id} := \tau$

$\mathsf{Vf}(ipk, (rid, uid, ctx, n, ep), \tau_{id}) \rightarrow 0/1$
Return $\mathsf{SIG.Vf}(ipk, (rid||uid||ctx||n||ep), \tau_{id})$
```

Figure 13: $\mathsf{AIF_{SIG}}$ – OIDC Implicit Flow [41] with standard signatures.

and valid, i.e,. where AResIdP does *not* respond with $\perp$. Non-trivial means that no honest RP created an authentication request for that session $(uid^*, n^*, pub_U{}^*, cep) \notin \mathsf{REQ}$ and that no RP is corrupt *and* a valid member in the current epoch, i.e., $\mathsf{CRID}[cep] = \emptyset$.

Recall that in $\mathsf{AIF_{SIG}}$, the authentication request of an RP consists of $\sigma_1 \leftarrow \mathcal{O}.\mathsf{SIG.Sign}(rsk, (1||uid||n||pub_U||ep))$ and the rid to which the rsk allegedly corresponds to. When the IdP receives (rid, σ_1) for a session for user uid and session nonce n in an epoch ep, it first looks up the rpk enlisted with rid, ep and then checks whether the signature is valid under rpk. It responds with $\perp$ if (at least) one of the checks fails.

We will branch the proof depending on two exclusive cases:

(1) The adversary made a query to $\mathcal{O}.\mathsf{CredIss}$ for an honest $rid \in \mathsf{HRID}$ (at the moment of the query), which led to a successful membership renewal, i.e., the CredIss protocol returned $\mathcal{M}'$, indicating completion of the protocol.
(2) The adversary made no impersonation query as in Case (1).

Case (1): Impersonating an honest RP in credential renewal. In the first case, the adversary successfully impersonated an honest rid towards the IdP in some epoch ep. This might look like a benign attack, as the RPs do not receive any actual membership credentials in this protocol. However, in combination with the adaptive corruption, this would lead to a valid attack against RP Accountability as follows:

- $\mathcal{A}$ queries $\mathcal{O}.\mathsf{Join\text{-}Reg}$ for a fresh rid to register rid with the honest IdP. In the $\mathsf{AIF_{SIG}}$ protocol, the IdP's membership state $\mathcal{M}$ now contains an entry (rid, rpk) for the honestly generated public key rpk.
- $\mathcal{A}$ makes a successful impersonation query to $\mathcal{O}.\mathsf{CredIss}$ for $rid \in \mathsf{HRID}$ in some epoch ep, upon which the oracle returns 1. In the $\mathsf{AIF_{SIG}}$ protocol, this means that the honest IdP's updated membership state $\mathcal{M}'$ now contains an entry for (rid, rpk, ep) and from now on will accept any authentication requests for rpk in that epoch. Note that this query does neither add (rid, ep) to CRID (because rid is not corrupt), nor does it add $(rid, ep, cred)$ to HRID (because the honest RP did not request this membership, it was the adversary).
- $\mathcal{A}$ makes a query $\mathcal{O}.\mathsf{CrptRP}$ in that epoch for the same rid, upon it will learn the RP's secret key rsk. Note that this will only add rid to CRID, but not (rid, ep) (because the honest RP did not request a membership credential in ep).
- Trivial "forgery": Since $auth_{RP}{}^*$ is a signature under rsk and its corresponding context, $\mathcal{A}$ can exploit knowledge of the signing key to generate valid authentication requests for any desired $(uid^*, n^*, pub_U{}^*)$ in the current epoch $cep = ep$. When AResIdP receives these requests, it will respond with $\tau \neq \perp$ since (rid, rpk, ep) is registered as valid in $\mathcal{M}$. Given that $ep \notin \mathsf{CRID}$, this constitutes a valid forgery within our RP Accountability game.

This attack strategy reveals that $\mathcal{A}$ does not need to forge any signatures as part of the final $auth_{RP}{}^*$ output, as the actual security breach already happened through the impersonation in a membership renewal request. This is what we will use in the proof of this first case. That is, here we do not wait for $\mathcal{A}$'s final output but already use the forgery the adversary makes in the $\mathcal{O}.\mathsf{CredIss}$ query to break the underlying signature scheme.

Let $\mathcal{B}$ be the adversary against the EUF-CMA security of the standard signature scheme, receiving a public key pk as input and having access to a Sign-oracle $\mathcal{O}.\mathsf{SIG.Sign}$ for sk. $\mathcal{B}$ then guesses which honest RP rid_i the adversary in the RP Accountability game will impersonate, and returns pk as honest public key of rid_i. All other RPs and the honest IdP are handled exactly as in the $\mathsf{AIF_{SIG}}$ protocol. In summary, $\mathcal{B}$ provides the following oracles to $\mathcal{A}$:

$\mathcal{O}.\mathsf{Join\text{-}Reg}$: Runs the standard Join for all $rid_j \neq rid_i$. For rid_i it uses pk instead of an internally generated key pair. Reg is always run normally.
$\mathcal{O}.\mathsf{Reg}$: Executes the oracle with standard Reg.
$\mathcal{O}.\mathsf{CredReq\text{-}CredIss}$: Executes standard protocol, except for rid_i. Here it does not generate any signature σ_0. This change is entirely internal to the oracle, i.e., not noticeable by $\mathcal{A}$.
$\mathcal{O}.\mathsf{CredIss}$: Executes the oracle with standard CredIss.
$\mathcal{O}.\mathsf{CrptRP}$: Returns the current $(rsk_j, cred_j)$ of the requested $rid_j \neq rid_i$. Abort if request is for rid_i.
$\mathcal{O}.\mathsf{SetEP}$: Increases epoch.

$\mathcal{O}$.AReqRP : For all $rid_j \neq rid_i$, its runs AReqRP normally, for rid_i it gets $\sigma_1 \leftarrow \mathcal{O}$.SIG.Sign$(1||uid||n||pub_U||ep)$.
$\mathcal{O}$.AResIdP : Executes the oracle with standard AResIdP.

Note that all oracles are either identical to the original game, or simulated in a perfect way (in the case of $\mathcal{O}$.CredReq-CredIss and $\mathcal{O}$.AReqRP). There is one exception: the $\mathcal{O}$.CrptRP oracle which aborts upon query rid_i. This can happen if $\mathcal{B}$ guessed rid_i incorrectly, i.e., we loose a factor of q here where q denotes the maximal number of RPs that would register. If rid_i was correctly guessed and will be the target of the impersonation attempt, we will end our reduction before the adversary can make a corruption query for rid_i, i.e., the fact that we do not know rsk_i does not matter then.

As soon as $\mathcal{A}$ makes an $\mathcal{O}$.CredIss query for the chosen target rid_i and provides a valid signature σ_0 for the random sid (chosen by $\mathcal{B}$ in the execution of the oracle), $\mathcal{B}$ aborts the game with $\mathcal{A}$ and returns $(m^* := (0||sid), \sigma_0)$ as its forgery in the SIG-EUF-CMA game. Due to the domain separation of the signature purposes, and the fact that we never made a single query for any message of the form $(0||m)$ to the $\mathcal{O}$.SIG.Sign oracle, it is obvious that (m^*, σ_0) is a fresh and valid forgery in the SIG-EUF-CMA game.

Case (2): Forging the authentication request $auth_{RP}{}^*$. In the second case, we know that the adversary never made a successful impersonation attempt in the credential renewal. Consequently, in the epoch of the forgery cep all members must be *honest* RPs which *properly* gained membership through the $\mathcal{O}$.CredReq-CredIss oracle, i.e., all public keys in $\mathcal{M}$ for cep belong to honest RPs.

That is, the adversary must output a forgery $auth_{RP}{}^* := (rid^*, \sigma_1^*)$, where SIG.Vf$(rpk, (1||uid^*||n^*||pub_U{}^*||cep), \sigma_1^*) = 1$ for an honest RP's public key rpk as $\mathcal{M}$ must contain (rid^*, rpk, cep).

Thus, here we can do a straightforward reduction to the security of SIG. We let $\mathcal{B}$ again guess the target RP rid_i and embed $\mathcal{B}$'s challenge public key pk as rpk_i.

The simulation of oracles is the same as in Case (1) described above, but here we let $\mathcal{A}$ play the RP Accountability game until the end and use its final output as forgery in the SIG-EUF-CMA game. We again loose a factor of q through guessing the targeted RP for the final forgery. As $\mathcal{A}$ only wins if it outputs a forgery for a *fresh* tuple $(uid^*, n^*, pub_U{}^*, cep) \notin$ REQ, $\mathcal{B}$ can immediately use $(m^* := (1||uid^*||n^*||pub_U{}^*||cep), \sigma_1^*)$ as a fresh forgery against pk. □

The proof of RP Session Binding is significantly simpler since we no longer need to handle adaptive RP corruptions, as all RPs are corrupt from the beginning. The RP Session Binding property depends solely on the unforgeability of the IdP's signature and the fact that the IdP maintains immutable records of valid (rid, rpk, ep) combinations in $\mathcal{M}$.

Theorem C.2. AIF$_{\text{SIG}}$ *achieves RP Session Binding if* SIG *(used by the IdP) is SIG-EUF-CMA secure.*

Proof Sketch. In the RP Session Binding experiment, the adversary $\mathcal{A}$ outputs a finalized token $\tau_{id}{}^*$ that must be valid for the honest user session $(rid^*, uid^*, ctx^*, n^*, cep)$, where cep represents the current epoch. To succeed, the adversary must comply with at least one condition: (a) ensuring that $(rid^*, uid^*, ctx^*, n^*, cep)$ is a fresh session, meaning it was never queried to $\mathcal{O}$.AResIdP-AFin, or (b) targeting a session intended by an honest user, but rid^* does not possess a credential in epoch cep.

Forgery under condition (a). To fulfill condition (a), the adversary must output a valid signature τ under the IdP's key ipk. This signature is the same as $\tau_{id}{}^*$, which must be a SIG signature on the session $(rid^*||uid^*||ctx^*||n^*||cep)$. Since condition (a) requires the session to be fresh, this can be immediately turned into a fresh valid forgery of the IdP's signature. We omit the straightforward reduction.

Forgery under condition (b). Assume the adversary $\mathcal{A}$ wins under condition (b). This implies that the honest IdP has properly signed the session, including rid^* and epoch cep, yet rid^* was not a valid member in cep. The IdP only outputs its signature τ when it receives a request $auth_{RP}$ for rid^* and cep that verifies under rpk where $(rid^*, rpk, cep) \in \mathcal{M}$. This ensures that the IdP will never return (or even compute) a valid signature for an rid^* that is *not* a valid member in cep. Thus, even under this condition, $\mathcal{A}$ must have forged the IdP's signature.

What remains to be shown is that $rid^* \notin$ CRID$[cep]$ implies that $(rid^*, rpk, cep) \notin \mathcal{M}$. A combination of (rid, ep) gets added to CRID in the $\mathcal{O}$.CredIss oracle when it handled a valid membership request for a *corrupt rid* (i.e., where $rid \in$ CRID). Thus, the only gap an adversary could try to exploit here is if it manages to successfully enroll an RP with $rid \notin$ CRID. This is impossible in the AIF$_{\text{SIG}}$ construction, as CredIss only enrolls an RP as valid for an epoch ep if $rid \in \mathcal{M}$. Moreover, $rid \in \mathcal{M}$ implies that rid was properly registered and thus must also be in CRID.

In summary, if the adversary $\mathcal{A}$ wins under condition (b), it must have again forged the IdP's signature – as it will never see and receive a signature for such an illegitimate query. The reduction is again straightforward.

□

No support of RP Hiding. AIF$_{\text{SIG}}$ cannot achieve RP Hiding as each RP is uniquely identified towards the IdP through its rid and associated signature/public key.

C.2 Achieving RP Hiding: AIF-COM

AIF$_{\text{COM}}$ translates POIDC [24] into our AIF syntax. It uses a commitment scheme COM to hide RP's identity rid in a commitment c towards the IdP while uniquely binding the IdP's token to rid by allowing the IdP to sign the committed value in $(c||uid||ctx||n||ep)$. The user can ensure that c contains the correct rid by sitting between the RP and IdP and is privy to the opening of the commitment. AIF$_{\text{COM}}$ does not foresee or easily allow any proper RP registration or authentication, and thus the related algorithms are merely empty shells. The full construction is given in Figure 14.

Theorem C.3. AIF$_{\text{COM}}$ *is RP Hiding if* COM *is hiding.*

Proof Sketch. In this construction, the IdP only receives (and signs) the commitment of rid, but does not learn the opening or any other RP-specific information. Thus, RP Hiding follows trivially from the hiding property of COM. □

```
$\mathsf{SetupIdP}(pp) \rightarrow ((isk, \mathcal{M}), ipk)$
$(sk, pk) \leftarrow \mathsf{SIG.KGen}(1^\kappa); isk := sk; ipk := pk$
Return $((isk, \mathcal{M} := \emptyset), ipk)$

$\langle \mathsf{Join}(ipk, rid), \mathsf{Reg}(rid, \mathcal{M}) \rangle \rightarrow \{((rsk, rpk), \mathcal{M}'), \perp\}$
RP : Send $(rid)$
IdP : If $(rid \in \mathcal{M})$ Return $\perp$ else return $\mathcal{M}' := \mathcal{M}[rid] := \epsilon$
RP : Return $(rsk := \epsilon, rpk := \epsilon)$

$\langle \mathsf{CredReq}(ipk, rid, rsk, sid, ep),$
  $\mathsf{CredIss}(rid, isk, \mathcal{M}, sid, ep) \rangle \rightarrow \{(cred, \mathcal{M}'), \perp\}$
IdP : Return $\mathcal{M}' := \mathcal{M}$; RP : Return $cred := \epsilon$

$\mathsf{AInit}(ipk, rid) \rightarrow (priv_U, pub_U)$
$(c, o) \leftarrow \mathsf{COM.Commit}(rid)$
Return $(pub_U := c, priv_U := o)$

$\mathsf{AReqRP}(ipk, rid, cred, uid, pub_U, priv_U, n, ep) \rightarrow auth_{RP}$
Return $auth_{RP} := \epsilon$

$\mathsf{AResIdP}(isk, \mathcal{M}, uid, ctx, auth_{RP}, pub_U, n, ep) \rightarrow \{\tau, \perp\}$
Parse $pub_U$ as $c$
Return $\tau \leftarrow \mathsf{SIG.Sign}(isk, (c||uid||ctx||n||ep))$

$\mathsf{AFin}(ipk, rid, uid, ctx, pub_U, priv_U, n, ep, \tau) \rightarrow \{\tau_{id}, \perp\}$
Parse $pub_U$ as $c$, $priv_U$ as $o$
If $(\mathsf{SIG.Vf}(ipk, (c||uid||ctx||n||ep), \tau) \neq 1 \lor$
    $\mathsf{COM.Open}(rid, c, o) \neq 1)$ return $\perp$
Else return $\tau_{id} := (\tau, c, o)$

$\mathsf{Vf}(ipk, (rid, uid, ctx, n, ep), \tau_{id}) \rightarrow 0/1$
Parse $\tau_{id}$ as $(\tau, c, o)$
Return 1 if
  $(\mathsf{SIG.Vf}(ipk, (c||uid||ctx||n||ep), \tau) = \mathsf{COM.Open}(rid, c, o) = 1)$
```

Figure 14: $\mathsf{AIF}_{\mathsf{COM}}$ **– Translates** POIDC **[24] to our** AIF **setting.**

No support of RP Accountability. The privacy provided by the $\mathsf{AIF}_{\mathsf{COM}}$ construction comes for the cost of having no RP Accountability. During registration, only the *rid* is stored, but no authentication is associated with it and the IdP blindly signs arbitrary *rid*s in its identity tokens. We could also let an RP register a public key of a signature scheme, like in $\mathsf{AIF}_{\mathsf{SIG}}$, and let it sign the commitment during authentication. This would be sufficient for achieving RP Accountability, but immediately destroy the privacy of this construction. Such an addition would also not be sufficient for RP Session Binding, as this must ensure that the commitment contains the identity of the signer — this is not possible with basic building blocks (but exactly what our new $\mathsf{AIF}_{\mathsf{ZKP}}$ protocol does by relying on more advanced primitives).

Partial support of RP Session Binding. Let us define *Partial RP Session Binding* as the RP Session Binding Experiment in Figure 7 without the inclusion of the second *condition (b)*, which ensures that the request originates from a *legitimate* RP. Since there is no RP authentication involved, $\mathsf{AIF}_{\mathsf{COM}}$ cannot fulfill this condition.

Theorem C.4. $\mathsf{AIF}_{\mathsf{COM}}$ *is partially RP Session Binding if* COM *is binding and* SIG *is SIG-EUF-CMA.*

Proof Sketch. An adversary who breaks Partial RP Session Binding of $\mathsf{AIF}_{\mathsf{COM}}$ outputs a valid identity token ${\tau_{id}}^* := (\tau, c, o)$ which consist of a valid signature τ on $(c||uid^*||ctx^*||n^*||cep)$ under pk and a valid opening o such that $\mathsf{COM.Open}(rid^*, c, o) = 1$. To meet the winning condition, uid^* must correspond to an honest user. However, no honest session consisting of $(rid^*, uid^*, ctx^*, n^*, cep)$ exists. In this scenario, we can differentiate between two mutually exclusive cases: (1) the sub-tuple (uid^*, ctx^*, n^*, cep) is fresh, meaning it has never appeared in a query to $\mathcal{O}.\mathsf{AResIdP\text{-}AFin}$, or (2) if the sub-tuple is not fresh, then rid^* must be fresh.

The first case immediately yields a valid forgery for the issuer's standard signature scheme SIG. The second case allows the adversary to re-use an honestly obtained IdP signature on $(c||uid^*||ctx^*||n^*||cep)$, but then $\mathcal{A}$ must have been able to open c to some rid^* that is different than in the honest query to $\mathcal{O}.\mathsf{AInit}$, which breaks the binding property of COM. □

D SECURITY PROOFS: AIF-ZKP

This section presents the proofs of Theorem 5.3 and Theorem 5.4, which states RP Accountability and RP Session Binding of $\mathsf{AIF}_{\mathsf{ZKP}}$.

D.1 RP Accountability

Before proving the following Theorem D.1, let us recall the important parts of our protocol. An RP uses a standard signature scheme SIG for authentication in the credential-issuance protocol with the IdP. In this phase, the IdP learns the *rid*. Upon successful authentication in an epoch *ep*, the IdP then uses a multi-message signature scheme MMS to sign the *rid* and epoch *ep* as current membership credential. Finally, for blind RP authentication, the RP proves knowledge of such a credential via a NIZK where it reveals the epoch and proves that it also contains an *rid* that is the same as in the user provided commitment (the commitment part is irrelevant for the RP Accountability though).

We now want to prove that $\mathsf{AIF}_{\mathsf{ZKP}}$ satisfies RP Accountability. As most parts are straight-forward, we omit the concrete reductions and simply sketch each of them.

Theorem D.1. $\mathsf{AIF}_{\mathsf{ZKP}}$ *achieves RP Accountability if* SIG *(used by the RP) is SIG-EUF-CMA,* MMS *is MMS-EUF-CMA secure, and the* NIZK *is zero-knowledge as well as simulation-sound extractable.*

Proof. In the RP Accountability experiment of $\mathsf{AIF}_{\mathsf{ZKP}}$, an adversary $\mathcal{A}$ wins if it outputs an authentication request ${auth_{RP}}^*$ for a fresh tuple $(uid^*, n^*, {pub_U}^*, cep)$ that is accepted by the IdP, while there are no corrupt RPs with a credential in epoch *cep*. Fresh refers to the requirement that $(uid^*, n^*, {pub_U}^*, cep)$ must not have been queried to an honest RP via $\mathcal{O}.\mathsf{AReqRP}$.

As the game strictly requires that no corrupt RP owns a membership credential for *cep* (as otherwise producing correct authentication requests is trivial), we can split the proof in two exclusive cases, similar to the RP Accountability proof of $\mathsf{AIF}_{\mathsf{SIG}}$.

(1) The adversary made a query to $\mathcal{O}.\mathsf{CredIss}$ for an honest $rid \in \mathsf{HRID}$ (at the moment of the query), which led to a successful membership renewal, i.e., the CredIss protocol returned $\mathcal{M}'$, indicating completion of the protocol.
(2) The adversary made no impersonation query as in Case (1).

In Case (1), ${auth_{RP}}^*$ can be constructed using a valid membership credential that was legitimately issued by the IdP but provided to the adversary posing as an honest RP. In contrast, in Case (2), the resulting $auth_{RP}^*$ is a direct forgery.

Case (1): Impersonating an honest RP in credential renewal. In the first case, we know that the IdP has correctly issued a membership credential for a specific epoch ep to the adversary, who is impersonating an honest RP. This issuance is confirmed through a query to $\mathcal{O}$.CredIss. However, no honest RP has ever generated a membership request in that epoch. Consequently, $\mathcal{A}$ gains knowledge of the honest RP's membership credential for that epoch, enabling them to easily create valid $auth_{RP}^*$ for the same epoch.

In our protocol, the credential issuance is protected through a standard signature scheme SIG for which each RP creates its individual key pair and initially registers its public key rpk with the IdP. To obtain a new membership credential requires to send a valid signature σ_{RP} on a fresh session nonce sid that verifies under the rpk registered for rid. Thus, the adversary must be able to forge a signature on sid that is chosen at random when invoking $\mathcal{O}$.CredIss. This is clearly infeasible if the signature scheme SIG is existentially unforgeable. Note that the corresponding rsk is never used outside of that credential request protocol, and even when used within the request, the adversary never learns honest RP's signatures: we assume communication among two honest parties to be secured, e.g., through a TLS channel. Thus, we could rely on a very weak unforgeability property, where the adversary has no access to a Sign-oracle. We opted for the classic EUF-CMA security for the sake of convenience.

The proof in this case is almost identical to $\mathsf{AIF_{SIG}}$. Let $\mathcal{B}$ be an adversary targeting the SIG-EUF-CMA security of SIG. $\mathcal{B}$ receives a public key pk as input and has access to a signing oracle $\mathcal{O}$.SIG.Sign for sk. $\mathcal{B}$ guesses the honest RP rid_i that the adversary in the RP accountability game will impersonate and returns pk as the honest public key. All other RPs and the honest IdP are handled in the same way as in the $\mathsf{AIF_{ZKP}}$ protocol. In summary, $\mathcal{B}$ provides the following oracles to $\mathcal{A}$:

$\mathcal{O}$.Join-Reg : Runs the standard Join for all $rid_j \neq rid_i$. For rid_i it uses pk instead of an internally generated key pair. Reg is always executed normally.

$\mathcal{O}$.Reg : Executes the oracle with standard Reg.

$\mathcal{O}$.CredReq-CredIss : Executes the standard protocol, except for rid_i, where no signature σ_{RP} is generated. This change is internal and not noticeable by $\mathcal{A}$.

$\mathcal{O}$.CredIss : Executes the oracle with standard CredIss.

$\mathcal{O}$.CrptRP : Returns the current $(rsk_j, cred_j)$ of the requested $rid_j \neq rid_i$. Aborts if request is for rid_i.

$\mathcal{O}$.SetEP : Increases epoch.

$\mathcal{O}$.AReqRP : Executes the oracle with standard AReqRP (note that rsk_i is not needed here).

$\mathcal{O}$.AResIdP : Executes the oracle with standard AResIdP.

All oracles are identical to the original game, except the credential issuance protocol $\mathcal{O}$.CredReq-CredIss, which is perfectly simulated. The only exception is the $\mathcal{O}$.CrptRP oracle, which aborts when queried with rid_i. This occurs if $\mathcal{B}$ incorrectly guessed rid_i, resulting in a factor loss of q, where q represents the maximum number of RPs that would be registered. If rid_i was correctly guessed and will be the target of the impersonation attempt, we will terminate our reduction before the adversary can make a corruption query for rid_i. Therefore, not knowing rsk_i becomes irrelevant in this case.

Upon $\mathcal{A}$ making an $\mathcal{O}$.CredIss query for the selected target rid_i and providing a valid signature σ_{RP} for $\mathcal{B}$'s randomly chosen sid, $\mathcal{B}$ immediately aborts the game with $\mathcal{A}$ and returns $(m^* := sid, \sigma_{RP})$ as its forgery in the SIG-EUF-CMA game. Since we never query the $\mathcal{O}$.SIG.Sign oracle for any message, it is evident that (m^*, σ_{RP}) is a fresh and valid forgery in the SIG-EUF-CMA game.

Case (2): Forging the authentication request $auth_{RP}^*$. We are in the second case, when no such impersonation attack happened. That is, all honestly generated membership credentials are only known to the corresponding honest RPs, yet no honest RP ever made an authentication request, as $(uid^*, n^*, pub_U{}^*, cep) \notin \mathsf{REQ}$ must hold.

The adversary can thus only win if it still knows a valid membership credential (then producing the NIZK is trivial), or it does not know a suitable membership credential but forged the proof π directly. The latter is clearly infeasible based on the soundness of the NIZK proof system. Thus, what remains to be shown is how we can reduce a correct proof π of a valid membership credential to a forgery of the underlying MMS scheme.

In the reduction to the unforgeability of MMS, we leverage the knowledge extractor of the proof system to obtain rid and σ_{IdP} from $\mathcal{A}'s$ output $auth_{RP}^* := \pi$ in epoch cep and will use $(\vec{m}^* := (rid, cep), \sigma_{IdP})$ as MMS forgery. Note that rid is entirely hidden in the proof, i.e., the adversary could choose to put an *honest* RP's rid in there. Consequently, we need to take care that we never request a MMS signature on an (rid, ep) combination that the adversary might use for its forgery, as this would invalidate the freshness requirement in the unforgeability game of the MMS scheme.

Ensuring such freshness is achieved by relying on the properties of the NIZK: When honest RPs request a membership credential in some epoch ep through $\mathcal{O}$.CredReq-CredIss, we do not create the MMS signature, but simply keep a record that the membership was granted. For subsequent legitimate calls to $\mathcal{O}$.AReqRP that would normally create a NIZK from the honestly issued MMS signature, we merely simulate the proof π.

There is one caveat in the simulation here: Recall that our game allows adaptive corruption of honest RPs, upon which we must return the rsk and current membership credential $cred$, containing the IdP's MMS signature σ_{IdP}. If such a corruption happens, our reduction calls the signing oracle $\mathcal{O}$.MMS.Sign on the proper (rid_i, ep) in the MMS-EUF-CMA game and returns the MMS signature. This does not harm our reduction though, as the epoch ep in which that happens immediately becomes invalid for any forgery. So already the epoch cep of $\mathcal{A}$'s finally forgery (which is also signed with the MMS signature) must be different to any epoch where $\mathcal{A}$ could have made such a corruption (or requested a membership credential for a corrupt RP), which implies that the extracted rid in combination with the fresh cep has never been queried to $\mathcal{O}$.MMS.Sign. We also note that we do not rely on any properties of the standard signature here (this was handled separately in Case (1) above), and thus, we can simply generate standard signature keys for all honest RPs and output them upon corruption.

In summary, we let an adversary $\mathcal{B}$ aiming to break the unforgeability of MMS by simulating the RP Accountability game towards $\mathcal{A}$ as follows: $\mathcal{B}$ first sets ipk as the mpk received from the MMS-EUF-CMA game and then uses its access to the O.MMS.Sign oracle as well as the fact that proofs π can be perfectly simulated to mimic the parts that would require knowledge of msk (or credentials thereof).

O.Join-Reg : Runs the standard Join-Reg protocol. Note that here msk (which is now unknown) is not needed.

O.Reg : Executes the oracle with standard Reg. Note that here msk (which is now unknown) is not needed.

O.CredReq-CredIss : Does not issue any membership credential to the honest RP, but merely keeps track that RP rid_i is a valid member in ep. This change is internal to the oracle, and thus not noticeable by $\mathcal{A}$.

O.CredIss : Executes the oracle with the help of the O.MMS.Sign oracle to which it sends (rid_i, ep) and uses the response as (σ_{IdP}, ep). As we are in Case (2), we know that all requests are for corrupt rid_i's, i.e., the epoch ep immediately becomes invalid for any forgery for $\mathcal{A}$.

O.CrptRP : Returns the requested rsk_i of the honest RP. If the honest RP was also a valid member in the current epoch, $\mathcal{B}$ requests the membership credential $\sigma_{IdP} \leftarrow O.\mathsf{MMS.Sign}(rid_i, ep)$ and returns (σ_{IdP}, ep). If this happens, the epoch ep becomes invalid for any forgery for $\mathcal{A}$.

O.SetEP : Increases epoch.

O.AReqRP : If rid_i was registered as a valid member in that epoch, $\mathcal{B}$ simulates the NIZK π.

O.AResIdP : Executes the oracle with standard AResIdP. Note that here only the secret key of the standard signature SIG is needed not the msk (which is unknown in the reduction).

The oracles in the game are either executed identically to the original game or perfectly simulated. Specifically, O.CredIss and O.CrptRP return the same credentials with the assistance of the O.MMS.Sign oracle from the MMS game. Since both credentials also sign an epoch ep, which cannot be utilized for a forgery, this does not compromise the freshness requirement of the final forgery.

The simulation of O.AReqRP is indistinguishable by the zero-knowledge property of the NIZK system. As we bind the proofs π to (uid, n, pub_U, ep), we ensure that the proof cannot be used in a different context. Most importantly, any proof returned by the O.AReqRP oracle can never be used as forgery (by the winning condition of the RP Accountability game). Thus, no simulated proof can be used as forgery, which is important to ensure the desired extractability.

$\mathcal{A}$ eventually outputs its forgery $auth_{RP}{}^*$ for fresh tuple $(uid^*, n^*, pub_U{}^*, cep)$ that is accepted by the IdP, while there are no corrupt RPs with a credential in epoch cep. Fresh refers to the requirement that $(uid^*, n^*, pub_U{}^*, cep)$ must not have been queried to an honest RP via O.AReqRP. We then use the knowledge extractor of the proof system to obtain rid and σ_{IdP} from $\pi := auth_{RP}{}^*$ in epoch cep and will use $(\overrightarrow{m}^*:= (rid, cep), \sigma_{IdP})$ as MMS forgery. It is easy to see that (rid, cep) is fresh and thus valid in the MMS game.

In conclusion, this shows that a forgery in Case (2) is infeasible, based on the unforgeability of the MMS scheme and the zero-knowledge property as well as simulation soundness of the NIZK.

□

D.2 RP Session Binding

Before proving the following Theorem D.2, let us recap the relevant part of the construction: the IdP employs a standard signature scheme SIG to issue identity tokens, while a user utilizes a commitment scheme COM to commit to the rid she wants to authenticate to. The IdP signs the commitment, together with the other user and context information, if it received a valid NIZK that proofs that the request stems from a properly authenticated RP. The latter relies on the MMS signature the RP must own on its rid and current epoch.

Session Binding holds, if the IdP's standard signature scheme and MMS signature are secure, the commitment scheme COM is binding, and the NIZK proof system used is special sound.

In the RP Session Binding game, all RPs are assumed to be corrupt from the beginning, which leads to a simpler proof than RP Accountability, as we do not have to handle adaptive corruptions of initially honest RPs.

Theorem D.2. *$\mathsf{AIF}_{\mathsf{ZKP}}$ is RP Session Binding if* SIG *(used by the IdP) is SIG-EUF-CMA secure,* COM *is binding,* MMS *is MMS-EUF-CMA secure, and the* NIZK *is special sound.*

Proof. An adversary in the RP Session Binding experiment must output a finalized token $\tau_{id}{}^*$ that is valid for an honest user session $(rid^*, uid^*, ctx^*, n^*, cep)$. By the winning condition of the game, the adversary wins if the session either

(a) is fresh, i.e., $(rid^*, uid^*, ctx^*, n^*, cep)$ was never used by O.AResIdP-AFin
(b) or this session was intended by an honest user, but then rid^* must belong to some RP that does not own a credential in epoch cep.

Forgery under condition (a). Recall that the finalized token $\tau_{id}{}^* := (\tau, c, o)$ contains an IdP's SIG signature τ on the combined message $(c||uid^*||ctx^*||n^*||cep)$ and a correct opening o for the commitment c to rid^*.

As we know that $(rid^*, uid^*, ctx^*, n^*, cep)$ must be fresh, there are two sub-cases in which $\mathcal{A}$ can win under condition (a): either the "public" session part (uid^*, ctx^*, n^*, cep) is fresh and $\mathcal{A}$ forged an IdP's signature, or that session part is not fresh, which in turn means that rid^* must be different than in the honest query to O.AInit. The latter breaks the binding property of COM. The first sub-case immediately yields a valid forgery for the IdP's (standard) signature scheme SIG.

Forgery under condition (b). To succeed under the second winning condition of the RP Session Binding game, the adversary provided an honest user session in which rid^* does not belong to a corrupt RP with a credential for epoch cep. Recall that in this game all RPs are corrupt. Therefore, the adversary must have provided an RP authentication request $auth_{RP} := \pi$ accepted by the IdP or it re-used a credential $cred$ of another RP $\overline{rid} \neq rid^*$ of epoch $\overline{ep} \neq cep$ and used it to authenticate rid^* of the honest user session to the IdP in epoch cep. In the first case, it must have forged π, which contradicts the soundness of the proof system. In the second case, it must have forged the signature σ contained in the credential $cred := (\sigma, \overline{ep})$, which is the IdP's MMS signature on $(\overline{rid}, \overline{ep})$.

The reduction to the unforgeability to MMS is straightforward, as we simply use the O.MMS.Sign oracle to answer any oracle call that request membership credentials for rid in epoch ep. $\mathcal{A}$'s final output contains the NIZK π from which we then extract the MMS signature σ, for the fresh (rid^*, cep) combination. The freshness of (rid^*, cep) immediately follows from the winning condition (b). □

E NIZK

In this section, we describe the concrete $\mathsf{AIF}_{\mathsf{ZKP}}$ NIZK instantiation, which corresponds to our implementation [22].

E.1 Building Blocks, Setup, and Instantiation

In the context of our NIZK instantiation in $\mathsf{AIF}_{\mathsf{ZKP}}$ (see Figure 16), let us first recap the setting. An issued credential by an IdP is represented by a MMS signature σ on (rid, ep). When the user initiates the protocol, it generates a commitment/opener (c, o) for the desired rid authentication. In the subsequent protocol, the RP proves, in zero-knowledge, its possession of a credential in epoch ep and knowledge of the commitment opening:

$$\mathsf{NIZK}\{(\sigma, rid, o) : \mathsf{Vf}(pk, (rid, ep), \sigma) = \mathsf{Open}(rid, c, o) = 1\}(ep, c).$$

This proof validates the possession of a valid MMS signature σ on (rid, ep) with respect to pk, along with knowledge of a valid opening o for the commitment to rid, without revealing (σ, rid, o).

Constructions, pairings, and hashing. We use Pedersen commitments [37] for the COM scheme, instantiated as $c \leftarrow g^m h^o$, where $o \leftarrow_{\mathrm{R}} \mathbb{Z}_p$ and $(g, h) \in \mathbb{G}_1^2$ with the discrete logarithm $\log_g h$ being unknown. The opening is verified by checking $g^m h^o = c$. For the MMS scheme, we utilize PS signatures [38], summarized in Figure 15, which rely on asymmetric pairings. Note that both schemes have the same message space, denoted as $\mathcal{S}_{\mathsf{Com}} = \mathcal{S}_{\mathsf{MMS}} = \mathbb{Z}_p$. The construction setup is outlined in the next paragraph.

To define asymmetric pairings, we consider the cyclic groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_{\mathrm{T}}$ of order p with the respective generators g_1, g_2, g_{T}. Moreover, let $e : \mathbb{G}_1 \times \mathbb{G}_2 \mapsto \mathbb{G}_{\mathrm{T}}$ be an efficiently computable non-degenerate function such that $\forall a, b \in \mathbb{Z}_p : e(g_1^a, g_2^b) := g_{\mathrm{T}}^{ab}$. Then e is called an asymmetric pairing. It must hold that $\mathbb{G}_1 \neq \mathbb{G}_2$ and that no efficient homomorphism $\phi : \mathbb{G}_2 \mapsto \mathbb{G}_1$ exists, which is a type-3 pairing. This asymmetric pairing is instantiated using the elliptic curve BLS12 – 381 [9].

Public parameters. The AIF public parameters pp include fixed generators $(g, h) \in \mathbb{G}_1^2$ for Pedersen commitments, where the discrete logarithm $\log_g h$ remains unknown and also provide the bilinear group description $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ for PS signatures.

```
$\underline{\mathsf{Setup}(1^\kappa) \to pp}$
Return $pp := (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$

$\underline{\mathsf{KGen}(pp, \ell) \to (sk, pk)}$
$(x, y_1, ..., y_\ell) \leftarrow_{\mathrm{R}} \mathbb{Z}_p^{\ell+1}; \widetilde{g} \leftarrow_{\mathrm{R}} \mathbb{G}_2$
$(X, Y_1, ... Y_\ell) \leftarrow (\widetilde{g}^x, \widetilde{g}^{y_1}, ..., \widetilde{g}^{y_\ell})$
$sk := (x, y_1, ..., y_\ell); pk := (\widetilde{g}, X, Y_1, ... Y_\ell)$
Return $(sk, pk)$

$\underline{\mathsf{Sign}(sk, \vec{m}) \to \sigma}$
Parse $sk$ as $(x, y_1, ..., y_j), \vec{m}$ as $(m_1, ..., m_j)$
$h \leftarrow_{\mathrm{R}} \mathbb{G}_1 \setminus \{1_{\mathbb{G}_1}\}; \sigma \leftarrow (h, h^{x+\Sigma y_j \cdot m_j})$
Return $\sigma$

$\underline{\mathsf{Vf}(pk, \vec{m}, \sigma) \to 0/1}$
Parse $pk$ as $(X, Y_1, ... Y_j), \vec{m}$ as $(m_1, ..., m_j), \sigma$ as $(\sigma_1, \sigma_2)$
If $(\sigma_1 = 1_{\mathbb{G}_1})$ abort
Return $e(\sigma_1, X \cdot \prod Y_j^{m_j}) = e(\sigma_2, \widetilde{g})$
```

Figure 15: Construction of PS signatures [38].

```
Prover
Inputs: $ipk, c, uid, n, ep, cred, rid, o$
Parse $ipk$ as $(\cdot, mpk)$, $mpk$ as $(\widetilde{g}, X, Y_1, Y_2)$
Parse $cred$ as $(\sigma_1, \sigma_2)$
If $(c \neq g^{rid} h^o)$ abort
$(r, t) \leftarrow_{\mathrm{R}} \mathbb{Z}_p^2; \sigma' \leftarrow (\sigma_1^r, (\sigma_2 \cdot \sigma_1^t)^r)$
$\pi \leftarrow \mathsf{NIZK}\{(rid, o, t)$ :
    $c = g^{rid} h^o \wedge$
    $e(\sigma_1', Y_1)^{rid} \cdot e(\sigma_1', \widetilde{g})^t =$
    $e(\sigma_2', \widetilde{g}) \cdot e(\sigma_1', X \cdot Y_2^{ep})^{-1}\}$
    $(uid, n, c, ep)$
Return $(\sigma', \pi)$

Verifier
Inputs: $ipk, uid, n, ep, c, \sigma', \pi$
Parse $\sigma'$ as $(\sigma_1', \sigma_2')$
If $(\sigma_1' = 1_{\mathbb{G}_1})$ abort
Return 1 if ($\pi$ verifies w.r.t. $(uid, n, c, ep)$)
```

Figure 16: The NIZK in $\mathsf{AIF}_{\mathsf{ZKP}}$.