Enhancing Privacy in Decentralized Identity Management

Anonymous Verifiable Credentials for Privacy-Preserving Digital Identity

Master Thesis

Cirkovic Marko

Faculty of Science at the University of Bern

Mariarosaria Barbaraci
Dr. Jayamine Alupotha
Prof. Christian Cachin
Cryptology and Data Security Group Institute of Computer Science
University of Bern, Switzerland
June 27, 2025

Abstract

Digital surveillance has become widespread in modern society, with governments, corporations, and malicious actors tracking individuals across online platforms. This surveillance apparatus violates fundamental privacy rights while creating comprehensive behavioral profiles that go far beyond the original purpose of digital interactions. Traditional identity systems exacerbate the problem by requiring persistent identifiers that allow correlation across services, making every digital transaction a potential privacy violation.

The Swiss electronic provisional driving license (eLFA) system [3] exemplifies this privacy challenge in action. Despite implementing robust standards-based verifiable credentials using OpenID protocols [17], the system relies on Decentralized Identifiers (DIDs) [20] to create linkable interactions across different verifiers. When learner drivers present their credentials to multiple instructors or authorities, the accesses to those become linkable, allowing for a level of profiling that goes beyond simple verification requirements violating user privacy

This thesis introduces a privacy-preserving framework called Anonymous Verifiable Credentials (AVC) that eliminates surveillance capabilities while still providing strong verification guarantees. AVC combines Verifiable Credentials [21] and User-issued Unlinkable Single Sign-On (U2SSO) [2], assigning credentials to service-specific pseudonyms instead of global identifiers. This approach ensures that even colluding service providers are unable to correlate user interactions, effectively preventing commercial tracking and state surveillance while maintaining cryptographic integrity and selective disclosure capabilities. Our contributions demonstrate practical privacy enhancements for existing identity infrastructure: we define the AVC protocol architecture, which achieves unlinkability without sacrificing Sybil resistance, implement pseudonym-based credential binding, which prevents cross-service correlation, and validate our approach by integrating with Switzerland's operational eLFA in the form of a prototype. The AVC framework demonstrates that strong privacy protection and regulatory compliance are compatible, providing a viable path to surveillance-resistant digital identity that empowers users while meeting institutional verification requirements.

Contents

1 Introduction

In today's increasingly digital society, access to both public and private services requires proof of identity or specific characteristics such as age, license status, or academic qualifications. As our daily interactions shift online, traditional identity verification methods face significant challenges in terms of privacy, security, and user control. Users want easy access to services without compromising their privacy, while service providers require dependable verification of user attributes to meet regulatory requirements and manage access. This creates a difficult balancing act: how to provide adequate proof of identity or attributes while minimizing unnecessary data disclosure and preventing tracking across services.

Verifiable Credentials (VCs) [21] are an effective paradigm for addressing these challenges. They allow individuals to obtain digitally signed claims from trusted issuers and selectively present them to verifiers. Unlike traditional identity systems, VCs give users more control over their personal data, encourage cross-platform interoperability, and enable privacy-preserving attribute disclosure. This aligns with modern data protection principles, including those outlined in the European Union's General Data Protection Regulation GDPR [1]. To better understand VCs, consider a university that issues digital degree credentials to its graduates. A student finishes their degree requirements and requests a credential from the university (the issuer). After verifying the student's academic record, the university issues a digitally signed credential with information such as the degree name, major, graduation date, and grade. This credential is cryptographically linked to the student's digital identity, usually via a Decentralized Identifier (DID) [20]. When applying for a job, the graduate can present only relevant attributes (for example, degree name and graduation date) to potential employers (verifiers) without disclosing their entire transcript or other personal information. Employers can cryptographically verify that the credential was legitimately issued by the university without contacting the university directly. This ensures both privacy and trust in the verification process.

Switzerland is among the countries testing digital identity solutions. One notable example is the electronic provisional driving license "elektronischer LernFahrAusweis" (eLFA) [3] program, which enables learner drivers to receive and present digital driving credentials via mobile applications. The eLFA infrastructure consists of several key components that work together to create a secure and user-friendly credential ecosystem. The Holder Wallet (mobile apps for Android [6] and iOS [8]) enables users to manage their digital identities and store credentials. The Swiss Road Traffic Office operates the Issuer Agent, which verifies user attributes and issues credentials according to the OpenID4VCI [17]

specification's pre-authorized flow. Verifier Agents, such as police officers and driving instructors, compare these digital credentials to the Base Registry, which keeps accurate records of issuer keys. A separate Revocation Registry maintains credential status information, allowing for revocation or suspension as needed. The Road Traffic Office is the system's primary issuer, with law enforcement officers and driving schools acting as verifiers.

While the Swiss eLFA infrastructure provides a robust and standards-based mechanism for issuing verifiable credentials, it inherits the significant privacy limitation of a persistent identifier DID, that uniquely links a credential to its holder across services. When the same DID is used for multiple service interactions, verifiers or external observers can compare the results and track user behavior over time. For example, when a user presents their driving license to multiple instructors using the same DID, the interactions become linkable, potentially revealing patterns in their learning process beyond what is required for verification.

Contributions

This thesis proposes a novel framework called *Anonymous Verifiable Credentials (AVC)*, to address the abovementioned critical privacy issue. Our approach combines Verifiable Credentials with the User-Issued Unlinkable Single Sign-On (U2SSO) protocol [2], a decentralized authentication scheme that enables users to authenticate anonymously across services and protect against Sybil attacks. By associating VCs with service-specific pseudonyms rather than global identifiers like DIDs, AVC preserves the cryptographic integrity and selective disclosure benefits of VCs while eliminating the risk of cross-service linking.

Our AVC framework includes a comprehensive workflow for privacy-preserving attribute verification. First, users must register their master identity with an Identity Registry, which then stores it in an anonymity set. When interacting with a service provider, users request verifiable credentials from reputable issuers and use service-specific pseudonyms based on their master identity. The issuer validates the user's attributes and issues credentials using pseudonyms rather than persistent identifiers. Users then create service-specific profiles for service providers, which include their pseudonyms and verifiable credentials. Service providers can continue using their existing systems and processes for handling VCs, with no need for major changes. Finally, service providers validate these profiles by confirming with the Identity Registry that users have valid master identities, even though they do not know which specific identities belong to them. This approach strikes a good balance between privacy protection and authentic verification requirements, filling a significant gap in existing digital identification frameworks while adhering to data protection regulations. AVC allows users to present the same credential to different service providers under different pseudonyms, preventing them from linking the interactions even if they collude. The AVC framework was developed by combining U2SSO's pseudonym generation with standard VC workflows. To assess the feasibility of our solution, we developed a proof-of-concept within the Swiss eLFA infrastructure, extending OID4VCI protocols to support pseudonym-based credential issuance and presentation.

This thesis makes the following key contributions to the field of privacy-preserving digital identity:

- **Novel Framework Design:** We introduce the Anonymous Verifiable Credentials (AVC) framework, which combines unlinkable authentication with verifiable attribute disclosure in a unified system. This framework addresses a fundamental limitation in existing digital identity solutions by enabling both unlinkability and verification simultaneously.

- **Prototype Integration:** We use the AVC framework to extend Switzerland's electronic provisional driving license (eLFA) system, demonstrating its compatibility with the existing government identity infrastructure. This integration demonstrates how privacy enhancements can be added to operational systems while maintaining core functionality.

The remainder of this thesis is organized as follows: Chapter 2 reviews existing research in privacy-preserving digital identity. Chapter 3 introduces the two key components of our framework: U2SSO and Verifiable Credentials. Chapter 4 presents the detailed design of our AVC framework. Chapter 5 describes our implementation within the Swiss eLFA infrastructure. Chapter 6 reflects on the practical challenges encountered during implementation and the lessons learned from integrating AVC with real-world infrastructure. Finally, Chapter 7 summarizes our findings and discusses implications for the future of privacy-preserving digital identity.

2
Related Work

This chapter examines existing research in the field of privacy-preserving digital identity, with a particular emphasis on systems that attempt to combine unlinkability, Sybil resistance, and credential verification mechanisms similar to our Anonymous Verifiable Credentials (AVC) framework.

2.1 Blockchain-based privacy-preserving identity systems

Blockchain-backed identity systems have emerged as a prominent solution for balancing privacy, Sybil resistance, and decentralized trust. One common approach is to use blockchains as passive Identity Registries (IdRs), where users create and store unique pseudonyms linked to their identity. Before accepting an authentication request, verifiers query the blockchain to check the existence or validity of a user's pseudonym. This strategy is used in systems like Sovrin [11], based on Hyperledger Indy [15] and AnonCreds [14], which supports selective disclosure and zero-knowledge proofs for verifiable credential presentations. Users can prove attributes without revealing their full identity, enhancing privacy across services.

Another family of solutions leverages Ethereum-based [9] identity frameworks [12, 18], where users register decentralized identifiers (DIDs) and associated public keys directly on-chain. While this offers strong integrity guarantees and decentralized control, the use of public blockchains like Ethereum introduces inherent traceability risks. Financial transactions and identity operations can be publicly observed and correlated, undermining multi-verifier unlinkability even if advanced cryptographic credential formats like AnonCreds are used.

Projects [13, 16] based on self-sovereign identity (SSI) principles have also adopted blockchain-backed identity storage to mitigate dependence on centralized Identity Providers (IdPs). However, these solutions often impose economic costs (e.g., gas fees on Ethereum) to create Sybil-resistant barriers, making identity creation economically costly but not necessarily privacy-preserving.

Despite these advances, most blockchain-based systems still require continuous interaction with on-chain registries and often expose linkable metadata through identity anchoring or payment traces. In contrast, our AVC framework takes a more lightweight and privacy-centric approach by avoiding persistent blockchain dependencies altogether. AVC achieves Sybil resistance and unlinkability through service-specific pseudonyms and nullifiers, eliminating the need for users to publish their identifiers or

credential proofs on-chain. This makes AVC more suitable for privacy-sensitive use cases where regulatory or technical constraints limit blockchain deployment.

2.2 Threshold cryptography for anonymous credentials

In addition to hardware-based approaches, Rabaninejad et al. [19] proposed a novel credential system that uses attribute-based threshold issuance tokens to achieve privacy-preserving and Sybil-resistant identity. Their system introduces anonymous counting tokens, which enable services to enforce a one-user-per-service policy while maintaining user unlinkability across services. By distributing the credential issuance authority across multiple parties via threshold cryptography, they prevent any single entity from tracking users while also preventing multiple registrations. The authors were also able to prevent Sybil attacks, since they prevent double registration without allowing tracking. However, AVC provides a more comprehensive framework by directly integrating with standardized protocols such as OpenID and providing a complete implementation that is compatible with existing identity infrastructure such as Switzerland's eLFA system. While Rabaninejad et al.'s work establishes solid theoretical foundations, AVC bridges theory and practice through its modular architecture and compatibility with existing identity systems, demonstrating the practical viability of unlinkable credentials in real-world applications.

2.3 Cryptographic Accumulators for Unlinkable Credentials

Yang et al. [23] investigated unlinkable verifiable credentials with dynamic accumulators. Their system enables users to present credentials multiple times without being tracked by constantly refreshing accumulator-based proofs. When a user needs to prove possession of attributes, they generate a one-time proof from the accumulator that does not reveal the original credential or create linkable identifiers. This ensures strong theoretical unlinkability using sophisticated cryptographic techniques. While dynamic accumulators provide an elegant mathematical solution to the unlinkability problem, credential revocation becomes significantly more complex, and proof regeneration is computationally intensive. AVC achieves similar unlinkability benefits with a more straightforward approach that employs service-specific pseudonyms derived from a master key. This lowers the cryptographic burden on clients while maintaining privacy across service interactions, making it more suitable for resource-constrained devices and everyday use cases.

2.4 Algebraic MACs for Anonymous Credentials

Chase et al. [5] pioneered the use of algebraic Message Authentication Codes (MACs) in efficient anonymous credential systems. Their cryptographic constructions enable unlinkable credential presentations based on zero-knowledge proofs, allowing for attribute verification while protecting privacy. While these systems laid the mathematical groundwork for anonymous credentials, they lacked practical integration with widely used identity protocols, limiting their real-world application. AVC builds on these cryptographic principles, but expands them into a comprehensive, standards-compliant framework that can be used with existing identity infrastructures. By combining these privacy-enhancing techniques with established protocols such as OpenID, AVC makes anonymous credentials practical for use in today's digital ecosystems, closing the gap between theoretical privacy models and usable identity systems.

2.5 Digital Signatures in Verifiable Credentials

Verifiable Credentials [4] fundamentally rely on digital signature schemes to establish cryptographic integrity and authenticity. When a credential issuer creates a VC, they generate a digital signature over the credential's contents using their private signing key. This cryptographically binds the credential data to the issuer's identity, prevents tampering after issuance, and enables verifiers to authenticate the credential's origin without contacting the issuer directly. Verifiers validate credentials using the issuer's corresponding public key to check the digital signature against the credential content. This ensures credentials cannot be forged without access to the issuer's private key, maintaining trust relationships essential for credential ecosystems. However, standard digital signature schemes create linkability challenges when the same credential is presented to multiple verifiers, as the digital signature remains constant across presentations. Advanced schemes like BBS+ digital signatures [22] enable selective disclosure and unlinkable presentations, but they are not yet available in production systems like the Swiss eLFA infrastructure.

3 Preliminaries

This chapter introduces two key components of our privacy-preserving identity framework: the User-issued Unlinkable Single Sign-On (U2SSO) protocol and Verifiable Credentials (VC).

First, we introduce the U2SSO protocol [2], which provides an alternative to traditional federated identity management solutions by eliminating the need for a centralized trusted identity provider (IdP). Instead, it uses an immutable identity registry (IDR) and cryptographic techniques to ensure both Sybil resistance for service providers and unlinkability for users across different services. Next, we introduce Verifiable Credentials, a framework for privacy-preserving attribute verification. Verifiable Credentials allow users to selectively disclose verified attributes without compromising their privacy, while U2SSO focuses on authentication. These preliminary steps lay the technical groundwork for our future integration of these complementary technologies into a comprehensive solution for private and secure digital identity management.

3.1 Single-Sign-On with Unlikable User-Issued Identities U2SSO

Traditional Single Sign-On (SSO) systems rely on centralized identity providers, posing risks such as single points of failure and privacy concerns. In contrast, U2SSO allows users to manage their own identities while maintaining the convenience of SSO. This method meets two critical criteria for modern identity systems: (1) it reduces Sybil attacks, in which malicious individuals create multiple identities to exploit services, and (2) it protects user privacy by ensuring unlinkability across services. U2SSO represents a significant advancement in decentralized identity management by allowing these seemingly contradictory properties to exist concurrently.

3.1.1 System Model

The U2SSO system comprises three main entities:

- **User (user)**: Creates a master identity and produces unlinkable service-specific pseudonyms and keys. The user maintains complete control over their identity without relying on a centralized authority or IdP

- **Service Provider (SP)**: Provides services and enforces Sybil-resistant authentication for users. Each SP has a unique identifier sname. Service providers can confirm that users are members of a valid identity set without knowing which specific identity belongs to them.
- **Identity Registry (IdR)**: A public, append-only registry for storing master public keys in anonymity sets and managing registered snames. The IdR is not trusted to issue credentials or store secrets; it only supports public membership proofs and Sybil detection.

3.1.2 Functionalities

We formally define the U2SSO system as a tuple of five algorithms. These capture the creation of a master identity, the derivation of unlinkable service-specific keys, the generation of Sybil-resistant proofs, and authentication mechanisms.

Definition 3.1.1 (U2SSO Scheme)**.** *A **U2SSO** scheme is defined by the tuple of algorithms:*

- $\mathsf{pp} \leftarrow$ ***U2SSO.Setup***$(1^\lambda, \mathsf{params})$*: Takes as input a security parameter λ and a set of global system parameters* params*; outputs a set of public parameters* pp*.*
- $(\mathsf{mpk}, (\mathsf{msk}, \mathsf{kdfkey})) \leftarrow$ ***U2SSO.CreateID***(pp)*: On input* pp*, it outputs a master public key* mpk*, a master secrete key* msk *and a key derivation key* kdfkey*.*
- $(\mathsf{cpk}, \mathsf{csk}) \leftarrow$ ***U2SSO.DeriveKey***$(\mathsf{kdfkey}, \mathsf{sname})$*: Takes as input a key derivation key* kdfkey *and a service name* sname*; outputs a pseudonym* cpk *and a corresponding child secret key* csk*.*
- $(\pi, \mathsf{nul}) \leftarrow$ ***U2SSO.Prove***$(\mathsf{pp}, \mathsf{msk}, \mathsf{cpk}, \Lambda, j)$*: Takes as input the public parameters* pp*, a master secret key* msk*, a pseudonym* cpk*, an anonymity set $\Lambda = \{\mathsf{mpk}_1, \ldots, \mathsf{mpk}_N\}$, and an index $j \in \{1, \ldots, N\}$ corresponding to the user's own mpk_j. Outputs a non-interactive zero-knowledge proof π and a nullifier* nul*.*
- $0/1 \leftarrow$ ***U2SSO.Verify***$(\mathsf{pp}, \mathsf{sname}, \mathsf{cpk}, \pi, \mathsf{nul}, \Lambda)$*: Takes as input the public parameters* pp*, a service name* sname*, a pseudonym* cpk*, a proof π, a nullifier* nul*, and an anonymity set Λ; outputs 1 if the the proof is valid, otherwise 0.*
- $\sigma \leftarrow$ ***U2SSO.AuthProve***$(\mathsf{csk}, \mathsf{chal})$*: Takes as input a service-specific secret key* csk *and a challenge* chal*; outputs a response σ.*
- $0/1 \leftarrow$ ***U2SSO.AuthVerify***$(\mathsf{cpk}, \sigma, \mathsf{chal})$*: Takes as input a pseudonym* cpk*, a challenge* chal*, and a response σ; outputs 1 if the the proof is valid, otherwise 0.*

3.1.3 Workflows

While the formal specification establishes the algorithmic foundation for U2SSO, this section describes how these algorithms interact in practice to form complete protocols. The U2SSO system is comprised of four main protocols:

3.1.3.1 Setup

During configuration, the IdR allows service providers to register their service identifiers (sname) and preferred credential generators. The system guarantees the uniqueness of all service names and incorporates them into the public parameters pp generated by **U2SSO.Setup**.

3.1.3.2 Master Identity Registration

To join the U2SSO system, a user first uses the **U2SSO.CreateID** algorithm to generate their master credentials. This process generates three essential cryptographic components: a public master identity (mpk) that will be stored on the IdR, a private master secret key (msk) that the user will keep, and a key derivation key (kdfkey) for generating service-specific credentials.
The user registers the master identity mpk with the IdR (Step 1 in Figure 3.1), and it is stored in an anonymity set. An anonymity set is a collection of N master identities that allows users to demonstrate membership without revealing which identity belongs to them.

3.1.3.3 Pseudonym Generation and Registration

Figure 3.1 illustrates the three-step process for registering with a service provider. The user first creates a service-specific key pair with **U2SSO.DeriveKey**:

$$(\mathsf{cpk}_l, \mathsf{csk}_l) := \textbf{U2SSO.DeriveKey}(\mathsf{kdfkey}, \mathsf{sname}_l).$$

They then use **U2SSO.Prove** to generate a proof and nullifier:

$$(\pi, \mathsf{nul}) := \textbf{U2SSO.Prove}(\mathsf{pp}, \mathsf{msk}, \mathsf{cpk}_l, \Lambda, j),$$

where j is the master identity index in the anonymity set. Finally, the user sends the entire registration package $(\mathsf{cpk}_l, \pi, \mathsf{nul}, \Lambda)$ to the service provider (Step 2).
Then, in Step 3, the service provider uses **U2SSO.Verify** and a full validation process to check the information. The service provider gets the anonymity set Λ from the IdR to make sure it has the most up-to-date information about its members. It then checks the proof π to make sure the user has a valid master identity in the set and that the nullifier nul has not been used before. This allows U2SSO to prevent duplicate registrations. If the verification is successful, the service provider registers the pseudonym and keeps the nullifier to stop people from registering with the same master identity again.

3.1.3.4 Authentication to Service

After registration, the user can authenticate to the service using a simpler four-step process (shown in Figure 3.1). The service provider starts the authentication process by sending the user a random challenge, chal. The user makes an authentication proof with $\sigma := \textbf{U2SSO.AuthProve}(\mathsf{csk}_l, \mathsf{chal})$ and sends it back to the service provider (Step 4) after getting the challenge. Finally, the service provider checks the authentication response with $\textbf{U2SSO.AuthVerify}(\mathsf{cpk}_l, \sigma, \mathsf{chal})$.
This authentication method makes it easy to authenticate users multiple times without needing complicated zero-knowledge proofs or interactions with the IdR. This makes the user experience smoother while keeping the U2SSO system's privacy and security features intact.

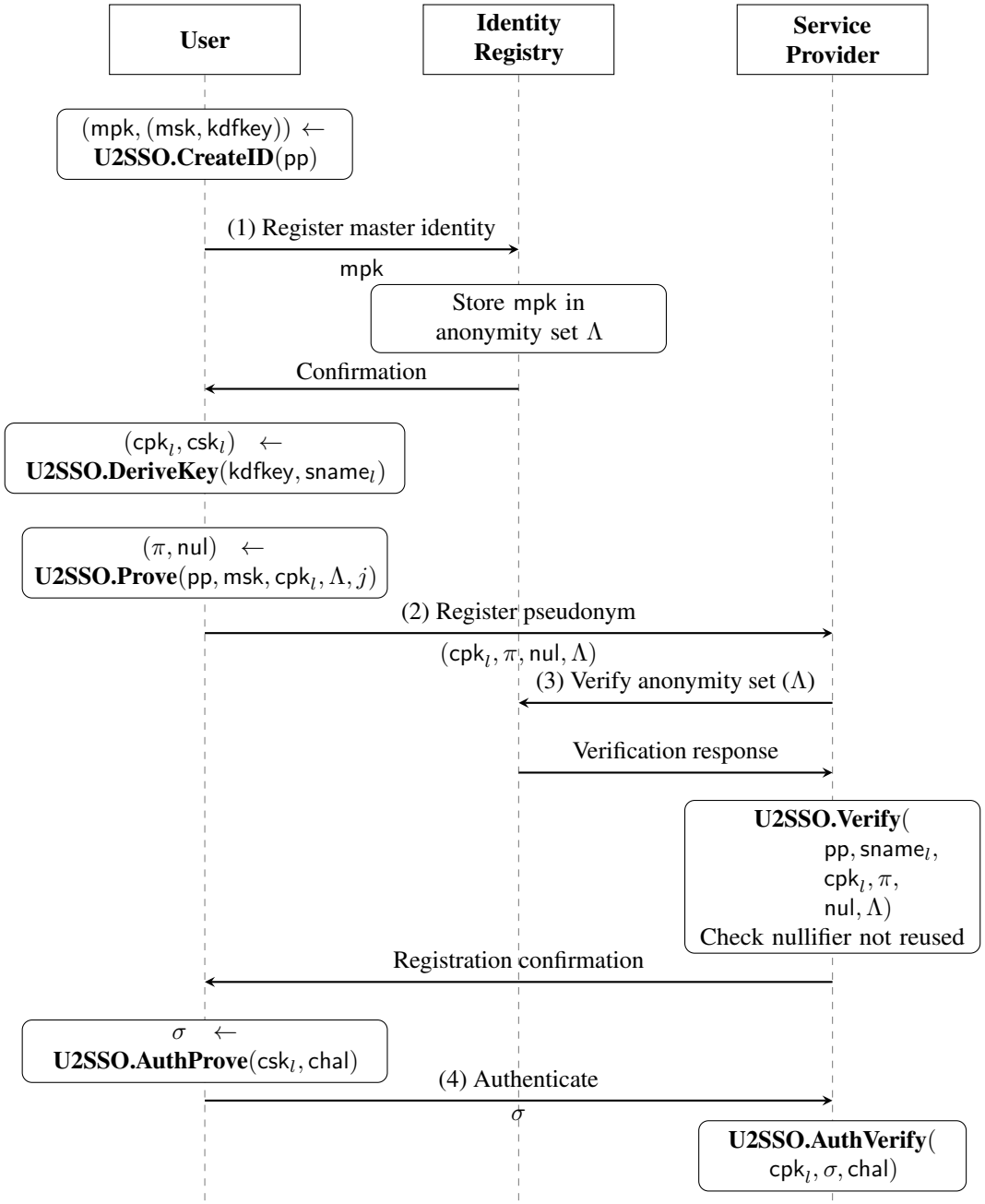


Figure 3.1: Complete U2SSO workflow. The numbers in parentheses show the most important steps: (1) registering your identity with the Identity Registry, (2) registering your pseudonym with the Service Provider, (3) checking against the anonymity set, and (4) authenticating without the IdR's help. The diagram shows the whole process, from making a master identity to registering it and then authenticating it.

3.1.4 Security Properties

The U2SSO protocol achieves three fundamental security properties through its cryptographic design and zero-knowledge proof construction.

Sybil Resistance. The system offers Sybil resistance because the nullifier nul is computed deterministically from both the master secret key msk and the service name sname_l. This ensures that for any given master identity and service provider pair, the same nullifier is generated. Because service providers store and verify nullifiers before accepting new registrations, a service provider can prevent users from registering multiple pseudonyms using the same master identity, as subsequent registration attempts would result in the same nullifier and be rejected.

Unforgeability. Unforgeability is guaranteed because generating a valid proof π requires knowledge of the master secret key msk, which corresponds to one of the master public keys mpk_j in the anonymity set Λ. The zero-knowledge proof construction ensures that without the secret key, an adversary cannot produce a proof that passes the verification algorithm **U2SSO.Verify**. Additionally, the binding be the pseudonym cpk_l and the master identity is cryptographically enforced through the key derivation process, preventing pseudonym forgery.

Robustness. The system remains robust because zero-knowledge proof verification only requires the prover to know the secret key associated with at least one valid master public key in the anonymity set. Even if some entries in Λ are malformed or invalid, honest users can still generate valid proofs by using their legitimate master identities. The verification algorithm can distinguish between valid and invalid proofs even when malformed entries are present, ensuring that legitimate users can continue to use the system.

Unlinkability. The U2SSO protocol ensures unlinkability across services because pseudonyms are derived deterministically from the master secret key and service name using **U2SSO.DeriveKey**$(\mathsf{kdfkey}, \mathsf{sname}_l)$, but appear statistically independent to external observers. Even if service providers collude and compare their records, they cannot determine whether different pseudonyms $\mathsf{cpk}_1, \mathsf{cpk}_2, \ldots$ belong to the same user. The key derivation function ensures that knowledge of one service-specific pseudonym provides no information about pseudonyms used with other services, preventing cross-service tracking and user profiling while maintaining the cryptographic binding to the user's master identity.

Anonymity. The U2SSO protocol provides anonymity because the zero-knowledge proof π demonstrates membership in the anonymity set Λ without revealing which specific master public key mpk_j belongs to the prover. The cryptographic construction ensures that even with access to all public information (anonymity set, service names, pseudonyms, and proofs), an adversary cannot determine which master identity generated a particular pseudonym cpk_l. This property holds as long as the anonymity set contains multiple legitimate identities and the underlying zero-knowledge proof system maintains its hiding properties.

3.2 Verifiable Credentials

Verifiable Credentials (VCs) [4] provide a framework for users to comply with specific requirements without disclosing private information.

This section introduces the Verifiable Credentials framework, a key component of our comprehensive Anonymous Verifiable Credentials (AVC) architecture. Standard VCs provide significant advantages over

traditional identity systems through selective disclosure, but they frequently rely on personal IDs.

3.2.1 System Model

The Verifiable Credentials system comprises three main entities:

- **User (user)**: Possesses verifiable credentials and provides them to service providers as needed. The user retains control over the attributes disclosed in various scenarios.
- **Verifiable Credential Issuer (VCI)**: A trusted entity (e.g., a governmental body, a financial organization, or an academic institution) that verifies user characteristics with real-world evidence and issues digitally signed credentials.
- **Service Provider (SP)**: Provides services contingent on specific user characteristics for access or compliance requirements. Each service provider verifies the authenticity of submitted credentials without knowing the user's true identity.

3.2.2 Functionalities

The Verifiable Credentials system is formalized as a set of algorithms. The components consist of system configuration, credential issuance, and credential verification procedures.

Definition 3.2.1 (Verifiable Credentials Scheme)**.** *A Verifiable Credentials scheme is defined by the tuple of algorithms:*

- $\mathsf{pp} \leftarrow$ ***VC.Setup***$(1^\lambda, \mathsf{params})$*: Takes as input a security parameter* λ *and a set of global system parameters* params*; outputs a set of public parameters* pp*.*
- $(\mathsf{sk}_{VCI}, \mathsf{pk}_{VCI}) \leftarrow$ ***VC.KeyGen***(pp)*: Takes as input the public parameters* pp*; outputs a signing key pair* $(\mathsf{sk}_{VCI}, \mathsf{pk}_{VCI})$ *for the credential issuer.*
- $VC_{\mathsf{PID}} \leftarrow$ ***VC.Issue***$(attr, \mathsf{PID}, \mathsf{sk}_{VCI})$*: Takes as input a set of attributes* $attr$*, a personal identifier* PID*, and the issuer's signing key* sk_{VCI}*; outputs a verifiable credential VC bound to the identifier* PID*.*
- $0/1 \leftarrow$ ***VC.Verify***$(attr, \mathsf{PID}, VC_{\mathsf{PID}}, \mathsf{pk}_{VCI})$*: Takes as input a set of attributes* $attr$*, a personal identifier* PID*, a verifiable credential VC bound to* PID*, and the issuer's public key* pk_{VCI}*; outputs 1 if the credential is valid for the given attributes and identifier, otherwise 0.*

Our definitions are intentionally protocol-agnostic, prioritizing functionality over specific cryptographic frameworks. This approach allows implementers to select appropriate cryptographic primitives based on their requirements, provided implementations satisfy the security principles detailed in Section 3.2.4.

3.2.3 Workflows

This section describes how the formal VC algorithms interact in practice to form complete protocols. The VC system comprises two main workflows that enable users to obtain and present verifiable credentials for attribute verification.

3.2.3.1 Credential Issuance

The credential issuance process allows users to obtain verifiable credentials from trusted issuers via a structured interaction (see Figure 3.2). The process starts with the service provider determining the required attributes $attr$ for access to their service (Step 1). The user requests information about the requirements from the service provider (Step 2), who responds by specifying the necessary attributes $attr$ (Step 3).

The user submits a credential request to the Verifiable Credential Issuer, providing their personal identifier PID and a set of attributes $attr$ for verification (Step 4). Upon receiving the request, the VCI confirms the legitimacy of the personal identifier PID. This verification usually entails traditional identity proofing methods like document inspection or in-person verification, which are deemed out of scope for this thesis. After validating the user's identity, the VCI uses **VC.Issue**$(attr, \mathsf{PID}, \mathsf{sk}_{\mathrm{VCI}})$ to create a verifiable credential that cryptographically binds the specified attributes to the user's personal identifier (Step 5). Finally, the VCI sends the credential $\mathrm{VC}_{\mathsf{PID}}$ to the user via a secure channel (Step 6).

This process ensures that credentials are only issued to verified users and cryptographically linked to their identifiers, preventing unauthorized use or transfer.

3.2.3.2 Credential Verification

The credential verification process allows users to demonstrate possession of verified attributes to service providers through the remaining steps of the protocol (shown in Figure 3.2). After obtaining the necessary credential, the user presents their personal identifier PID and the corresponding credential $\mathrm{VC}_{\mathsf{PID}}$ to the service provider (Step 7). This presentation should only include the attributes requested by the service provider, following the principle of minimal disclosure.

The service provider uses **VC.Verify**$(attr, \mathsf{PID}, \mathrm{VC}_{\mathsf{PID}}, \mathsf{pk}_{\mathrm{VCI}})$ to verify that the credential was issued by a trusted authority, is bound to the presented identifier, and contains the required attributes (Step 8). Based on the verification results, the service provider grants or denies access to the requested service, communicating the decision to the user (Step 9).

This verification mechanism enables attribute-based access control while protecting user privacy through selective disclosure, ensuring that service providers only learn the information required to meet their specific needs.

3.2.4 Security Properties

The Verifiable Credentials system provides four essential security and privacy properties through its cryptographic design and protocol structure.

Correctness. The Verifiable Credentials system achieves correctness in the sense that the credential issuance algorithm **VC.Issue**$(attr, \mathsf{PID}, \mathsf{sk}_{\mathrm{VCI}})$ cryptographically binds the specified attributes to the user's identifier using the issuer's signing key. When the same attributes and identifier are later presented for verification, the algorithm **VC.Verify**$(attr, \mathsf{PID}, \mathrm{VC}_{\mathsf{PID}}, \mathsf{pk}_{\mathrm{VCI}})$ can cryptographically confirm this binding using the corresponding public key. The deterministic nature of the cryptographic operations ensures that legitimate credentials will always verify successfully when presented with the correct attributes and identifier.

Unforgeability. Unforgeability means that credential generation requires knowledge of the issuer's private signing key $\mathsf{sk}_{\mathrm{VCI}}$, which is computationally infeasible to derive from the public key $\mathsf{pk}_{\mathrm{VCI}}$. The cryptographic digital signature scheme used in **VC.Issue** ensures that any attempt to forge or modify a credential without the private key will result in verification failure during **VC.Verify**. This property holds under the assumption that the underlying digital signature scheme is existentially unforgeable under chosen message attacks.

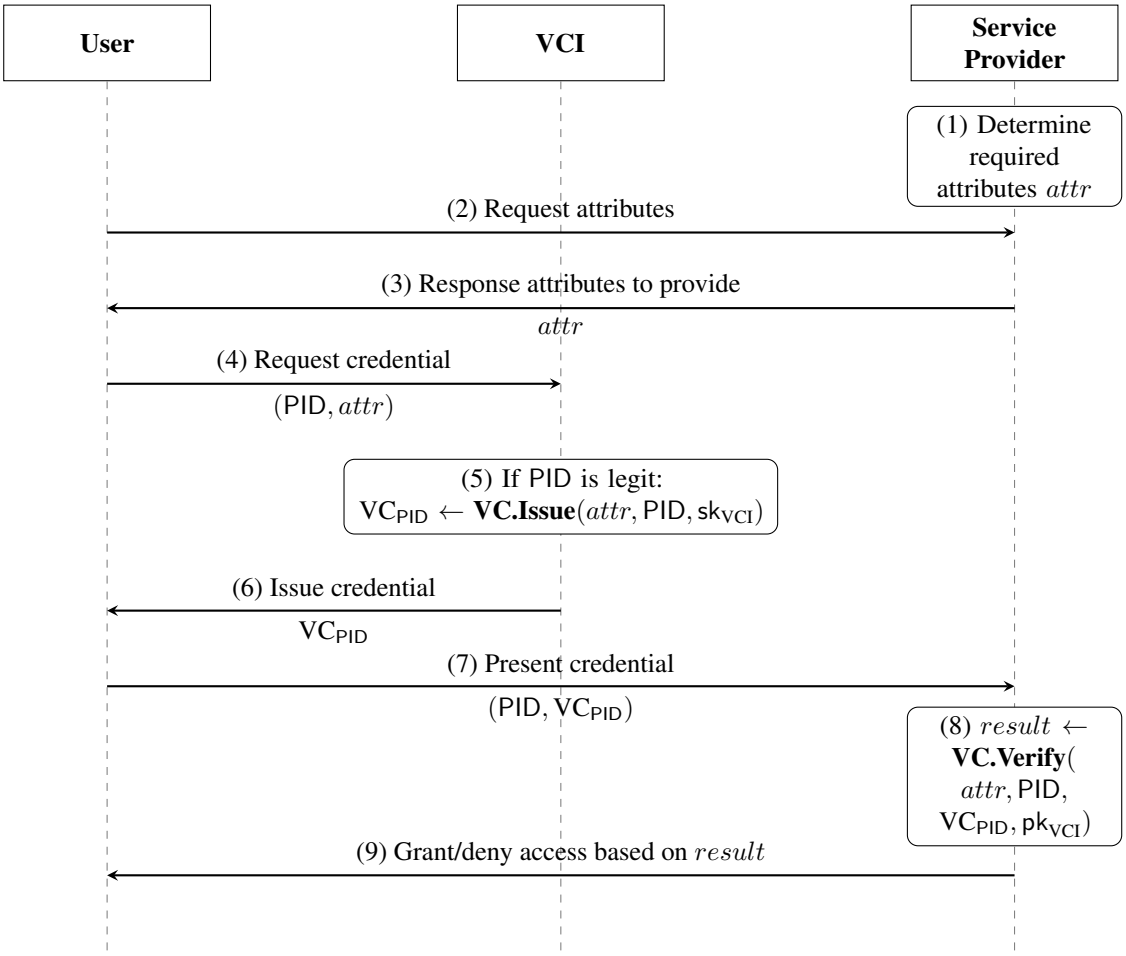


Figure 3.2: Verifiable Credentials System Workflow. The diagram illustrates the complete lifecycle: (1) The Service Provider determines required attributes; (2-3) The User and Service Provider exchange information about required attributes; (4) The User requests a credential from the Verifiable Credential Issuer; (5) The VCI verifies the user's identity legitimacy (this step involves traditional identity verification methods and is considered out of scope for this thesis) and issues a credential; (6) The VCI delivers the credential to the User; (7) The User presents the credential to the Service Provider; (8) The Service Provider verifies the credential's validity; and (9) The Service Provider grants or denies access based on the verification result.

Identifier-binding. Identifier-binding ensures that each verifiable credential is cryptographically tied to a specific user identifier, preventing credentials from being transferred or used by unauthorized parties. This property guarantees that a credential issued to one user cannot be presented by another user, even if the credential data is somehow obtained by an attacker. The verification algorithm **VC.Verify**$(attr, \mathsf{PID}, \mathrm{VC}_{\mathsf{PID}}, \mathsf{pk}_{\mathrm{VCI}})$ checks that the identifier presented matches the one embedded in the credential's cryptographic structure. Any attempt to use a credential with a different identifier will cause verification to fail, as the cryptographic binding cannot be altered without invalidating the issuer's digital signature.

Minimal Disclosure. The system supports minimal disclosure in the sense that verification protocol allows users to selectively present only the required attributes $attr$ to the service provider, rather than revealing the entire credential contents. The cryptographic structure of the credential enables verification of individual attributes without exposing additional information. The service provider's verification process using **VC.Verify** confirms only the presented attributes, ensuring that users maintain control over their personal data and comply with privacy principles such as data minimization mandated by frameworks like GDPR [1].

4
Anonymous Verfiable Credentials AVC

Traditional Verifiable Credentials systems, while offering significant privacy advantages over centralized approaches, have a fundamental limitation: they rely on persistent personal identifiers (PIDs). These identifiers enable service providers to track and profile users across multiple services, posing a significant privacy risk. When a user uses the same PID-bound credential with multiple service providers, these providers can work together to link the user's activities, effectively undermining the selective disclosure benefits that VCs strive to provide.

In previous chapters, we talked about two different frameworks: User-issued Unlinkable Single Sign-On (U2SSO) for privacy-preserving authentication and Verifiable Credentials (VC) for attribute verification. Each addresses a different aspect of the digital identity challenge: U2SSO enables authentication across services without tracking, whereas VCs provide dependable attribute verification. However, neither solution fully addresses the issue of privacy-preserving, attribute-based authorization. This chapter introduces our Anonymous Verifiable Credentials (AVC) framework, which combines these two approaches into a single system with their respective capabilities. The AVC framework addresses the PID linkability issue by associating verifiable credentials with service-specific pseudonyms rather than persistent personal identifiers. This approach ensures that even if a user provides credentials to multiple service providers, the interactions are not linked. Even when colluding, service providers cannot tell if they are interacting with the same people.

Instead of creating entirely new cryptographic primitives, we demonstrate how existing algorithms and protocols can be combined to form a comprehensive approach to privacy-preserving digital identity management. By combining U2SSO's unlinkability features with VCs' attribute verification capabilities, our AVC framework achieves Sybil resistance and cross-service unlinkability while allowing for reliable attribute verification.

4.1 System Components

The integrated AVC system combines the components of both U2SSO and VC while maintaining their original roles and security properties. This section describes the architectural components and setup of the integrated system.

- **User (user)**: Generates a master identity following the U2SSO protocol and derives service-specific pseudonyms as needed. Additionally, a user obtains verifiable credentials from trusted issuers and presents them to service providers.
- **Identity Registry (IdR)**: Continues to function as in U2SSO, storing master public keys in anonymity sets and facilitating the unlinkable authentication process. It remains unaware of the credentials associated with any pseudonym.
- **Verifiable Credential Issuer (VCI)**: Verifies user attributes through real-world proofs and issues credentials bound to specific service-specific pseudonyms rather than to personal identifiers.
- **Service Provider (SP)**: Verifies both the U2SSO authentication proof and the bound verifiable credentials, ensuring both that the user is legitimately registered and possesses the required attributes.

The integration does not require introducing additional entities beyond those already defined in the component frameworks, maintaining a lean architecture that minimizes potential privacy leakage points.

4.2 Workflow

The AVC framework integrates the authentication workflow of U2SSO with the credential verification features of VC systems. Figure 4.0 illustrates the complete workflow, from master identity creation to pseudonym-bound credential verification. Notably, this integrated approach replaces the persistent personal identifier (PID) from traditional VC systems with service-specific pseudonyms (cpk_l), thereby eliminating cross-service tracking.

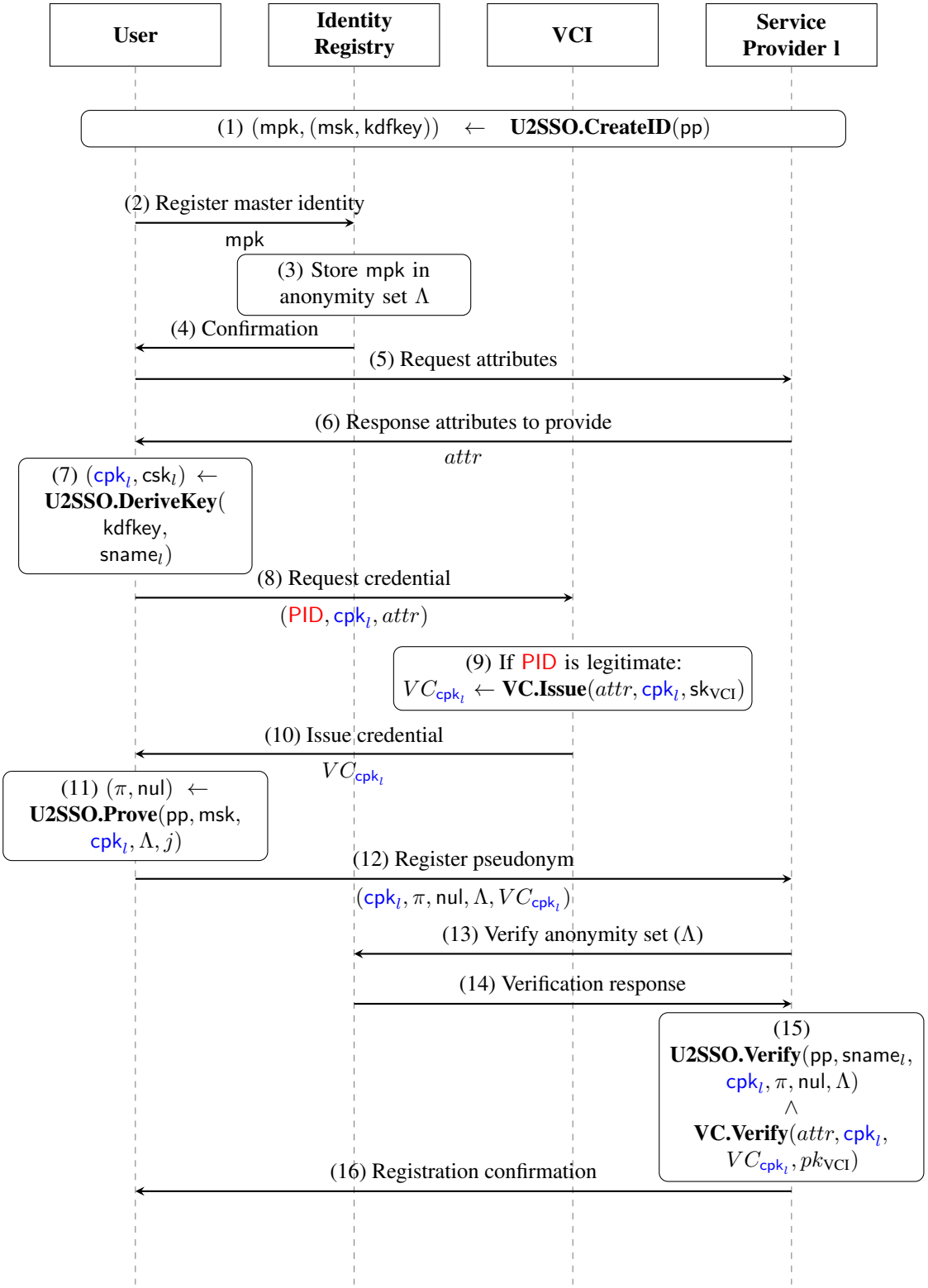
4.2.1 Workflow Description

The AVC workflow represents a fundamental departure from traditional identity systems by integrating unlinkable authentication with verifiable credential presentation (as shown in Figure 4.0). The key innovation lies in binding credentials to service-specific pseudonyms rather than persistent identifiers, enabling privacy-preserving attribute verification across multiple services.

The workflow can be conceptually divided into three phases. The initialization phase (steps (1)–(4) in Figure 4.0) establishes the user's master identity within the anonymity set of the Identity Registry, providing the cryptographic foundation for all subsequent interactions. The credential acquisition phase (steps (5)–(10)) demonstrates how users can obtain verifiable credentials bound to service-specific pseudonyms rather than global identifiers, with the VCI issuing credentials to the pseudonym after verifying the user's real-world identity through traditional methods.

The registration and verification phase (steps (11)–(16)) showcases the dual verification mechanism that distinguishes AVC from existing approaches. The Service Provider must verify both the U2SSO proof (confirming the user possesses a legitimate master identity) and the verifiable credential (confirming the required attributes), creating a comprehensive authentication and authorization framework. This dual verification ensures that users can prove both their legitimacy within the system and their possession of specific attributes without revealing their global identity or enabling cross-service tracking.

Once registered, the system enables efficient recurring authentication (steps (17)–(19)) through a lightweight challenge-response protocol that maintains the privacy guarantees while providing practical usability for everyday interactions.


User
Identity Registry
VCI
Service Provider l
(1) (mpk, (msk, kdfkey)) ← U2SSO.CreateID(pp)
(2) Register master identity
mpk
(3) Store mpk in anonymity set Λ
(4) Confirmation
(5) Request attributes
(6) Response attributes to provide
attr
(7) (cpk_l, csk_l) ← U2SSO.DeriveKey(kdfkey, sname_l)
(8) Request credential
(PID, cpk_l, attr)
(9) If PID is legitimate:
VC_cpk_l ← VC.Issue(attr, cpk_l, sk_VCI)
(10) Issue credential
VC_cpk_l
(11) (π, nul) ← U2SSO.Prove(pp, msk, cpk_l, Λ, j)
(12) Register pseudonym
(cpk_l, π, nul, Λ, VC_cpk_l)
(13) Verify anonymity set (Λ)
(14) Verification response
(15) U2SSO.Verify(pp, sname_l, cpk_l, π, nul, Λ) ∧ VC.Verify(attr, cpk_l, VC_cpk_l, pk_VCI)
(16) Registration confirmation

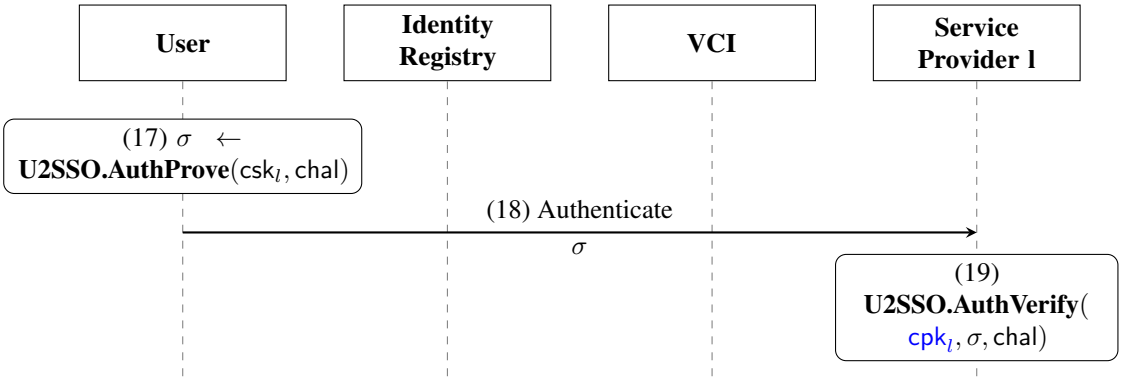


Figure 4.0: Anonymous Verifiable Credentials (AVC) Workflow. The diagram illustrates the integration of U2SSO authentication with verifiable credentials: ((1)–(4)) the User creates and registers a master identity with the Identity Registry; ((5)–(6)) the User determines required attributes with the Service Provider; (7) the User derives a service-specific pseudonym for Service Provider l; ((8)–(10)) the User requests and receives a credential bound to their pseudonym rather than their personal identifier; ((11)–(16)) the User registers the pseudonym and associated credential with the Service Provider, which verifies both the authentication proof and the credential; ((17)–(19)) for subsequent authentications, the User can employ a simpler authentication protocol without re-verifying the credential. The key innovation is binding credentials to pseudonyms (cpk_l, shown in blue) rather than to personal identifiers (PID, shown in red), preventing cross-service tracking. Color coding throughout the diagram highlights this key difference where VCs are bound to pseudonyms instead of PID.

4.2.2 Design Flexibility

Although our workflow assumes a verifiable credential is provided during initial registration with the Service Provider, the AVC framework allows for flexible implementation. The system can be easily customized to meet operational needs while maintaining its core security features. Instead of presenting credentials during registration, users can register their pseudonyms and only present credentials when accessing services that require attribute verification. This increases privacy by allowing basic authentication without revealing attributes until necessary. The AVC framework is adaptable to different usage scenarios while maintaining its core privacy and security properties. It combines the privacy benefits of U2SSO with the attribute verification capabilities of VC systems to create a comprehensive framework for privacy-preserving digital identity management, addressing the limitations of each system.

4.2.3 Security Properties

The integrated AVC system inherits and enhances the security properties of both U2SSO and VC frameworks, creating a comprehensive privacy-preserving digital identity solution with six fundamental security guarantees, as follows.

Unlinkability. The AVC framework maintains unlinkability by cryptographically generating service-specific pseudonyms using **U2SSO.DeriveKey**($\mathsf{kdfkey}, \mathsf{sname}_l$), resulting in statistically independent outputs for various services. Credential binding to pseudonyms using **VC.Issue**($attr, \mathsf{cpk}_l, \mathsf{sk}_{\mathrm{VCI}}$) does not generate additional linkability vectors because each credential is associated with a unique pseudonym. The zero-knowledge proof π created in step (11) demonstrates membership in the anonymity set without revealing which specific master identity generated the pseudonym. This prevents colluding service

providers from determining whether different pseudonym-credential pairs belong to the same user, unless the attributes themselves contain linkable information that could enable correlation across services.

Sybil resistance. AVC's nullifier mechanism, inherited from U2SSO, ensures Sybil resistance by computing nul deterministically from both the master secret key msk and the service name sname_l. This prevents users from registering multiple pseudonym-credential combinations with the same service provider while utilizing the same master identity. Credential binding does not violate this property because credentials are issued to pseudonyms that are already subject to Sybil resistance constraints, ensuring one-to-one correspondence between master identities and service registrations.

Unforgeability. The AVC system guarantees dual unforgeability: users cannot forge U2SSO proofs without a valid master secret key msk, and credentials cannot be forged without access to the issuer's private key $\mathsf{sk}_{\mathrm{VCI}}$. The integration preserves both properties because the U2SSO proof verifies legitimate system membership and the credential digital signature ensures attribute authenticity. Neither proof can be generated nor modified without the corresponding private keys, preventing identity spoofing and attribute falsification.

Credential integrity. AVC's modified issuance process, **VC.Issue**$(attr, \mathsf{cpk}_l, \mathsf{sk}_{\mathrm{VCI}})$, keeps credentials cryptographically bound to their designated pseudonyms. The verification algorithm ensures that the presented pseudonym matches the one embedded in the credential, preventing credential transfer between pseudonyms, even if they belong to the same user. This binding is maintained throughout the integrated verification process, ensuring that credentials cannot be used in multiple service contexts.

Minimal disclosure. The system preserves VCs' selective disclosure capabilities while improving privacy via pseudonym-based presentation. Users can only provide required attributes $attr$ to service providers, and the pseudonym-based binding ensures that this selective disclosure cannot be linked across services. The verification process using
VC.Verify$(attr, \mathsf{cpk}_l, \mathrm{VC}_{\mathsf{cpk}_l}, \mathsf{pk}_{\mathrm{VCI}})$ confirms only the presented attributes and does not reveal additional information or enable cross-service correlation.

Robustness. AVC ensures system robustness by ensuring that invalid entries in the anonymity set do not disrupt legitimate users' ability to obtain and use pseudonym-bound credentials. Even when there are invalid anonymity set entries, the integrated verification process can distinguish between valid U2SSO proofs and authentic credentials, allowing honest users to continue using the system while preserving all privacy and security guarantees.

5 Implementation

This chapter describes a practical implementation of our Anonymous Verifiable Credentials (AVC) framework. Moving from theoretical models to operational systems necessitates careful consideration of current infrastructure, technical standards, and real-world constraints. Our implementation not only shows the feasibility of the suggested framework, but it also provides useful insights into its practical application. We selected to evaluate our AVC architecture in the context of a major national digital identification effort, the Swiss Electronic Provisional Driving License Program (eLFA). This implementation is a realistic proof of concept, demonstrating how our theoretical contributions can improve privacy in existing identity systems without requiring a total overhaul of the underlying infrastructure. The next sections go over the system architecture, component relationships, and specific implementation choices taken to connect the AVC framework with the eLFA infrastructure. We focus on the cryptographic operations that enable pseudonym-based credential binding because they represent the primary innovation of our work. In addition, we discuss how conventional protocols such as OpenID for Verifiable Credentials (OID4VC) were expanded to support our privacy-enhancing methods. By using the AVC framework in the real-world setting, we demonstrate that it is feasible technically and that it works with other digital identity projects. This opens the door for identity providers and service providers who want to improve user privacy while keeping strong verification capabilities to use it.

5.1 Project Context: The Swiss Electronic Provisional Driving License Program (eLFA)

The implementation of our Anonymous Verifiable Credentials (AVC) framework was developed within the context of the Swiss electronic Provisional Driving License Program, known as eLFA (elektronischer LernFahrAusweis). This project serves as a proof of concept for Self-Sovereign Identity (SSI) using OpenID for Verifiable Credentials (OID4VC) with Selective Disclosure JSON Web Tokens (SD-JWT) as the authentication mechanism between issuers and the base registry [7].

5.1.1 Architectural Components

The eLFA infrastructure consists of five primary components, as illustrated in Figure 5.1:

- **Holder Wallet**: Apps for Android and iOS that let users manage their digital identities and save their credentials. These wallets use the OpenID for Verifiable Credentials (OID4VC) protocols to issue and show credentials [6, 8]. The wallet keeps the user's credentials safe and lets them choose which attributes to share when they talk to verifiers.
- **Issuer Agent**: The Swiss Road Traffic Office is the trusted authority that checks user information and gives out verifiable credentials [7]. The issuer specifically implements the pre-authorized flow of the OpenID4VCI specification and supports both standard JWT and SD-JWT [10] credential formats depending on configuration.
- **Verifier Agent**: Services that check credentials, like the police and driving instructors who need to check digital driving licenses. The Verifier checks credentials against the base registry and uses caching to speed things up [7].
- **Base Registry**: A central part that handles the public keys needed for ecosystem interactions and serves as the basis for verifying identities within the system [7]. The base registry is very important for building trust in the ecosystem because it keeps official records of issuer keys.
- **Revocation Registry**: Split from the Base Registry to allow for different optimizations, keeps track of credential status information so that issuers can revoke, suspend, or restore credentials as needed [7]. This separation makes it easier to manage and check the status list. This thesis does not cover credential revocation mechanisms.

5.1.2 Current System Workflow

Figure 5.1 illustrates the complete workflow of our AVC implementation, showing both the credential issuance and verification processes. The workflow is based on the OpenID for Verifiable Credentials (OID4VC) specifications and has been implemented in the eLFA project [7].

5.1.2.1 Credential Issuance

The credential issuance process follows the pre-authorized flow of the OpenID4VCI specification and consists of the following steps:

1. **Metadata Request**: The Holder Wallet initiates the process by requesting metadata from the Issuer Agent to understand its capabilities and requirements.
2. **Metadata Response**: The issuer provides configuration details, including supported credential types and formats (JWT or SD-JWT).
3. **Token Redemption**: The Holder Wallet redeems a one-time token that was previously provided as part of a credential offer, identifying the specific credential to be issued.
4. **Bearer Token Issuance**: The issuer validates the one-time token and provides a bearer token for subsequent authentication.
5. **Credential Request**: The Holder Wallet requests the verifiable credential, including proof of holder binding to ensure the credential is cryptographically bound to the holder's key.
6. **Credential Issuance**: The issuer generates and signs the verifiable credential (using either JWT or SD-JWT format) and returns it to the Holder Wallet, which stores it securely for future use.

5.1.2.2 Credential Verification

When a holder needs to prove possession of certain attributes to a verifier, the verification process occurs as follows:

7. **Request Object Request**: The Holder Wallet requests verification parameters from the Verifier Agent.
8. **Request Object Response**: The Verifier provides a request object containing a presentation definition that specifies which credential types and attributes are required.
9. **VP-Token Submission**: The Holder Wallet creates a Verifiable Presentation (VP) containing the requested credentials and attributes, and submits this VP-Token to the Verifier.
10. **Issuer Key Verification**: The Verifier contacts the Base Registry to retrieve the public key of the credential's issuer, essential for verifying the credential's digital signature.
11. **Verification Result**: After validating the VP-Token's digital signature, checking the credential's status, and verifying the presented attributes, the Verifier sends the verification result back to the Holder Wallet.

This workflow ensures both the privacy of the holder and the security of the verification process. By using the Base Registry to verify issuer keys, the system maintains a decentralized trust model while preventing cross-service tracking of users [7].

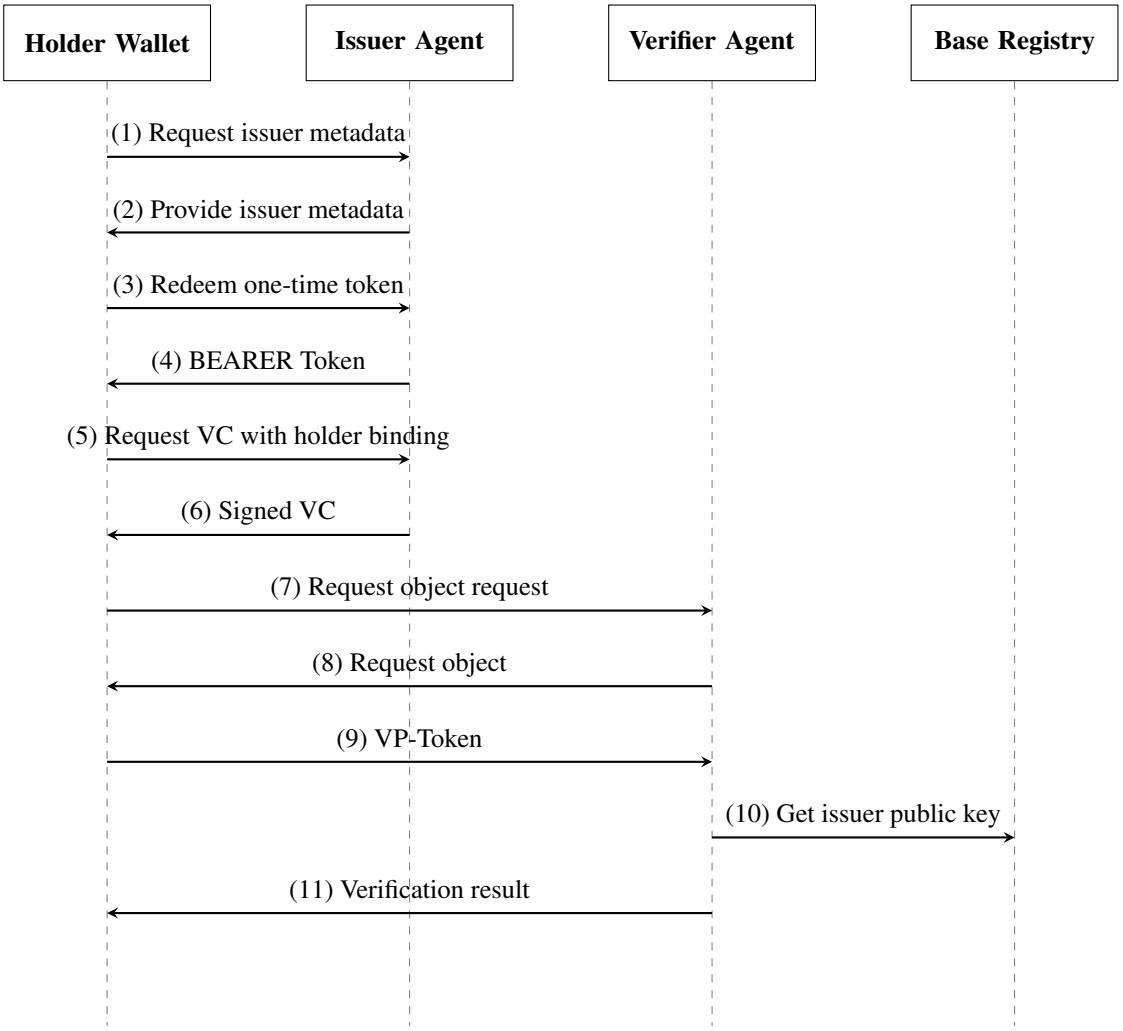


Figure 5.1: AVC System Flow Diagram showing the sequence of interactions during credential issuance and verification based on the eLFA implementation

5.2 AVC-eLFA Integration: Technical Implementation

This section explains how our Anonymous Verifiable Credentials (AVC) framework works with the eLFA infrastructure on a technical level. Building upon the basic eLFA architecture described in the previous section, we now detail the specific modifications and enhancements required to enable AVC's privacy-preserving authentication and attribute verification systems work. We use standard OpenID protocols and custom cryptographic operations to allow pseudonym-based credential binding without having to make major changes to the existing infrastructure.

5.2.1 Service Provider Registration with Identity Registry

Before participating in the AVC ecosystem, each Service Provider (Verifier) must register itself with the Identity Registry (IdR) following the U2SSO protocol. This registration is a prerequisite setup phase that occurs before step (1) of Figure 4.0, establishing the Verifier's identity within the system and enabling it to later verify user pseudonyms.

```
function registerService(servicename):
    url ← registry_identity_url + "/identity/service"
    headers ← generateRegistryHeaders()
    params ← {"name": servicename}
    response ← sendPostRequest(url, headers, params)
    if response.status_code is not ok:
        raise Error("Registration failed")
    return servicename
end function
```

Listing 1: Service Provider Registration to Identity Registry

As shown in Figure 5.2, the registration API endpoint accepts a service name and returns a unique identifier. The service provider sends its name to the Identity Registry through a secure API endpoint, where the registry records this identifier in its database or confirms the service is already registered. This registration establishes the Service Provider as a trusted entity within the U2SSO framework that underlies our AVC system.

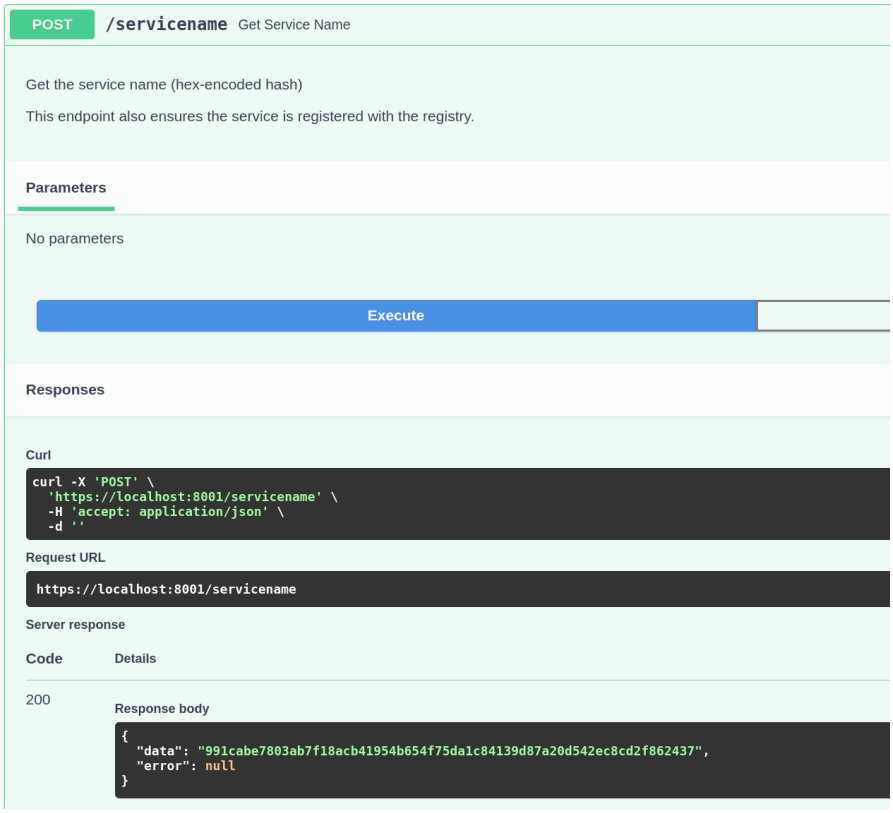


Figure 5.2: Service Provider Registration API endpoint demonstrating the process of registering a service identity. The endpoint returns a unique cryptographic hash identifier that establishes the Verifier's identity in the AVC trust framework and enables subsequent pseudonym verification.

5.2.2 Credential Format Definition for Verification

Before a verification process can begin, the Verifier must define the format and requirements for the Verifiable Credentials it will accept. This corresponds to steps (5)–(6) in Figure 4.0 (user requests attributes from Service Provider, Service Provider responds with required attributes), representing a critical preparatory step in the AVC workflow.

```
function createVerification(presentationDefinition):
    // Create verification management
    verificationId = generateUniqueIdentifier()

    // Store presentation requirements
    storePresentationDefinition(presentationDefinition)

    // Generate verification link
    verificationLink = generateVerificationLink(verificationId)

    return verificationLink
end function
```

Listing 2: Creating Verification Request

The Verifier makes a Presentation Definition object that lists the type of credential needed, the cryptographic algorithms that should be used for signatures, the specific attributes that must be present, and any other validation rules that must be followed. Listing 2 shows how the system makes a unique verification ID, saves the presentation requirements, and makes a verification link that users can share.
An example of a Presentation Definition for a university degree credential demonstrates the structure and specificity required for effective credential verification:

```
{
  "input_descriptors": [
    {
      "id": "Masters Degree",
      "format": {
        "jwt_vc": {
          "alg": "ES512"
        }
      },
      "constraints": {
        "fields": [
          {
            "path": ["$.vc.type[*]"],
            "filter": {
              "type": "string",
              "pattern": "UniversityDegreeCredential"
            }
          },
          {
            "path": ["$.vc.credentialSubject.degree.average_grade"]
          }
        ]
      }
    }
  ],
  "client_metadata": {
    "client_name": "Dummy University",
    "logo_uri": "Dummy Logo Uri"
  }
}
```

Listing 3: Example Presentation Definition for Degree Verification

The completed verification request is stored in the Verifier's cache with a unique identifier, which is then used to generate a verification URI following the format `https://verifier/request-object/{authorizationRequestId}/avc`. This approach ensures that the Verifier clearly specifies its requirements upfront, enabling selective disclosure of required attributes.

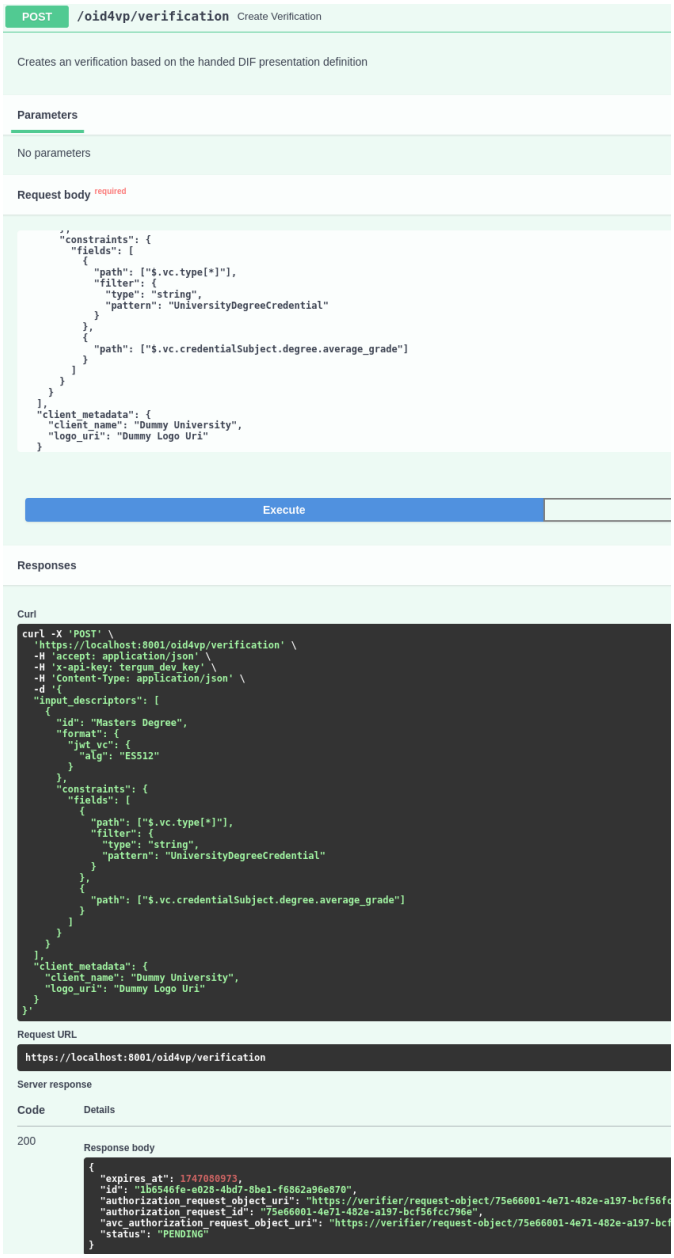


Figure 5.3: The Credential Verification Request API endpoint shows how to make a verification request with a full Presentation Definition. This definition says that the credential must be of type “UniversityDegreeCredential”, use ES512 for signatures, and have certain attributes, such as “average_grade”. Users will get the verification URI that comes out of this process so they can start the credential presentation process.

5.2.3 Master Identity Creation

Before interacting with Service Providers, users must first establish a master identity in the Identity Registry (IdR) using the U2SSO protocol. This implementation corresponds to steps (1)–(4) in Figure 4.0 (master identity creation, registration with IdR, storage in anonymity set, and confirmation), creating the cryptographic foundation for all subsequent privacy-preserving interactions.

```
function createIdentity(identity):
    if not passkeyExists(identity):
        createPasskey(identity)
    msk_bytes ← loadPasskey(identity)
    service_list ← fetchServices()
    id_bytes ← createId(msk_bytes, service_list)
    owner_id ← generateUniqueIdentifier()
    addIdentityToRegistry(id_bytes, owner_id)
    return {
        "identity": identity,
        "passkey": hexEncode(msk_bytes)
    }
end function
```

Listing 4: User Identity Creation in IdR

This step creates a master identity that will be used for all future AVC system interactions that keep your information private. Listing 4 shows that the system first looks for a passkey for the identity in question and makes one if it does not find one. The next step is to load or make the master secret key (msk) and then get a list of services that are available from the Identity Registry. Using the msk and the list of services, one can derive a cryptographic identity. After that, a random owner ID is created, and the whole identity is put in the Identity Registry.

Service Providers never see this master identity directly, so it stays private. Instead, it is used to create U2SSO service-specific pseudonyms that let people authenticate across different services without being able to link them.

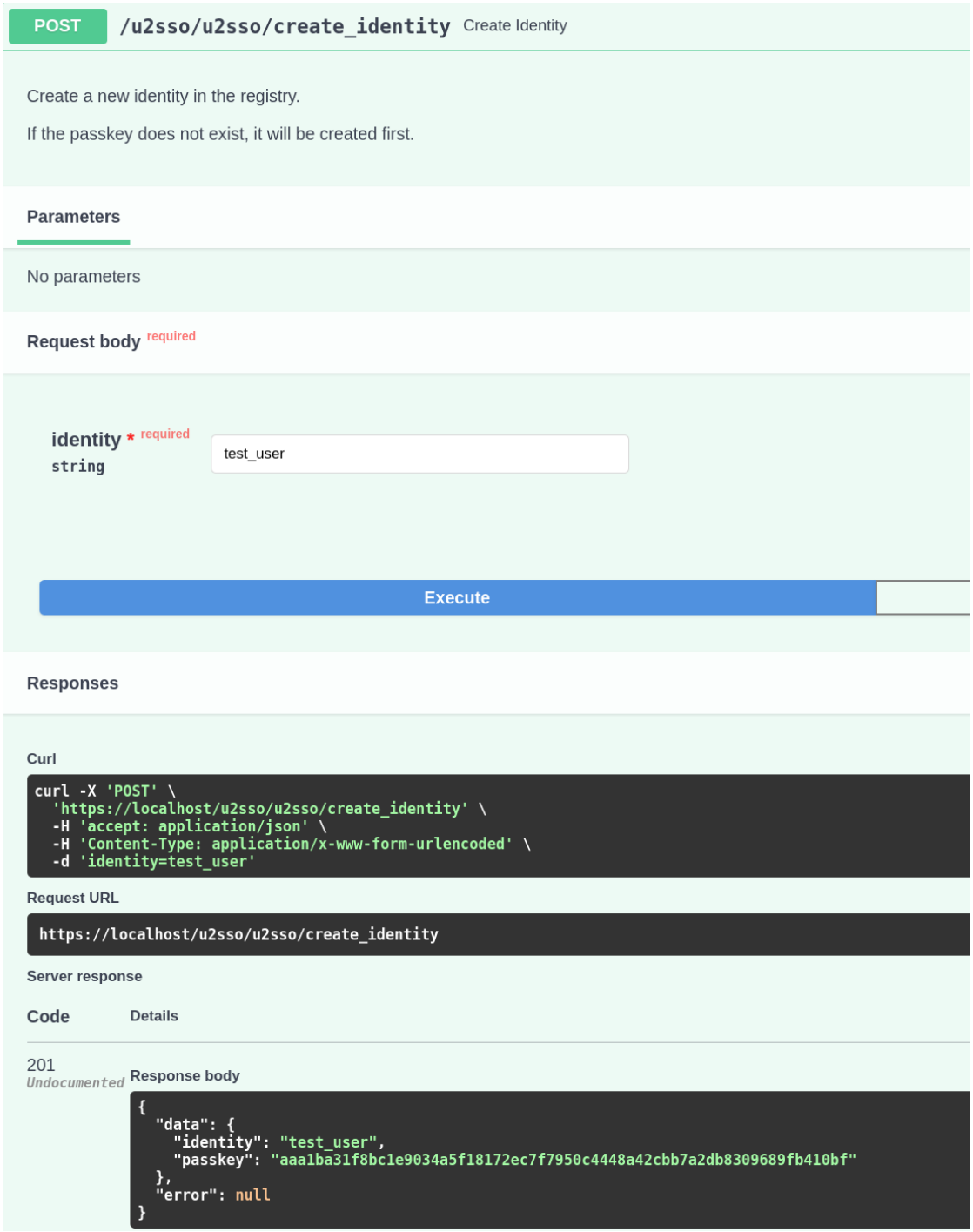


Figure 5.4: Master Identity Creation API endpoint. The screenshot shows the process of creating a new identity with a simple username (“test_user”) which generates both a master identity and a cryptographic passkey. This master identity forms the foundation for all pseudonym derivations while remaining private and never directly exposed to Service Providers.

5.2.4 Service-Specific Pseudonym Derivation

After creating a master identity in the IdR, the user must derive a service-specific pseudonym (child public key or cpk) before receiving verifiable credentials. This implementation corresponds to step (7) in Figure 4.0 (derive service-specific pseudonym), enabling privacy-preserving interactions while preventing cross-service tracking through U2SSO's unlinkability guarantees.

```
function registerWithService(servicename, challenge, identity):
    // Convert input parameters to required format
    service_name_bytes ← hexDecode(servicename)
    challenge_bytes ← hexDecode(challenge)

    // Load the user's master secret key
    msk_bytes ← loadPasskey(identity)

    // Fetch registered services from IdR
    service_list ← fetchServices()

    // Find service index in the registry
    service_index ← fetchServiceIndex(servicename)

    // Create user ID from master secret key
    id_bytes ← createId(msk_bytes, service_list)
    id_index ← fetchIdentityIndex(id_bytes)

    // Verify identity exists in registry
    if id_index == -1:
        raise Error("Identity not registered")

    // Get ID list for zero-knowledge proof
    id_list_bytes ← fetchIdList()

    // Calculate parameters for the proof
    current_m, ring_size ← calculateRingParameters(length(id_list_bytes))

    // Generate the proof and pseudonym
    proof, cpk, nullifier ← generateProof(
        id_index, current_m, service_name_bytes,
        challenge_bytes, msk_bytes, id_list_bytes,
        service_list, service_index
    )

    // Verify the proof (additional safety check)
    if not verifyProof(proof, cpk, nullifier):
        raise Error("Proof verification failed")

    return {
        "proof": proof,
        "cpk": cpk,
        "nullifier": nullifier,
        "ring_size": ring_size
    }
end function
```

Listing 5: Deriving Service-Specific Pseudonym

Figure 5.5 demonstrates the service-specific pseudonym derivation process through the API endpoint. Listing 5 shows the complete U2SSO protocol implementation: input parameters are converted to the required format, the user's master secret key is loaded, and registered services are fetched from the IdR. The system then locates the service index, creates a user ID from the master secret key, and verifies the identity exists in the registry. The core U2SSO innovation occurs in the zero-knowledge proof generation,

where the system fetches the complete identity list, calculates ring parameters for anonymity, and generates a proof that demonstrates membership in the anonymity set without revealing which specific identity belongs to the user. The nullifier prevents double registration while maintaining unlinkability.
This process creates three important parts that make it possible to interact with services while keeping your privacy: a zero-knowledge proof that shows valid identity membership, a service-specific pseudonym that is only used for this service, and a nullifier that stops Sybil attacks. The issuer later uses the derived cpk to link verifiable credentials to this pseudonym instead of a permanent identifier. This lets verification happen without giving away personal information and stops tracking across services.

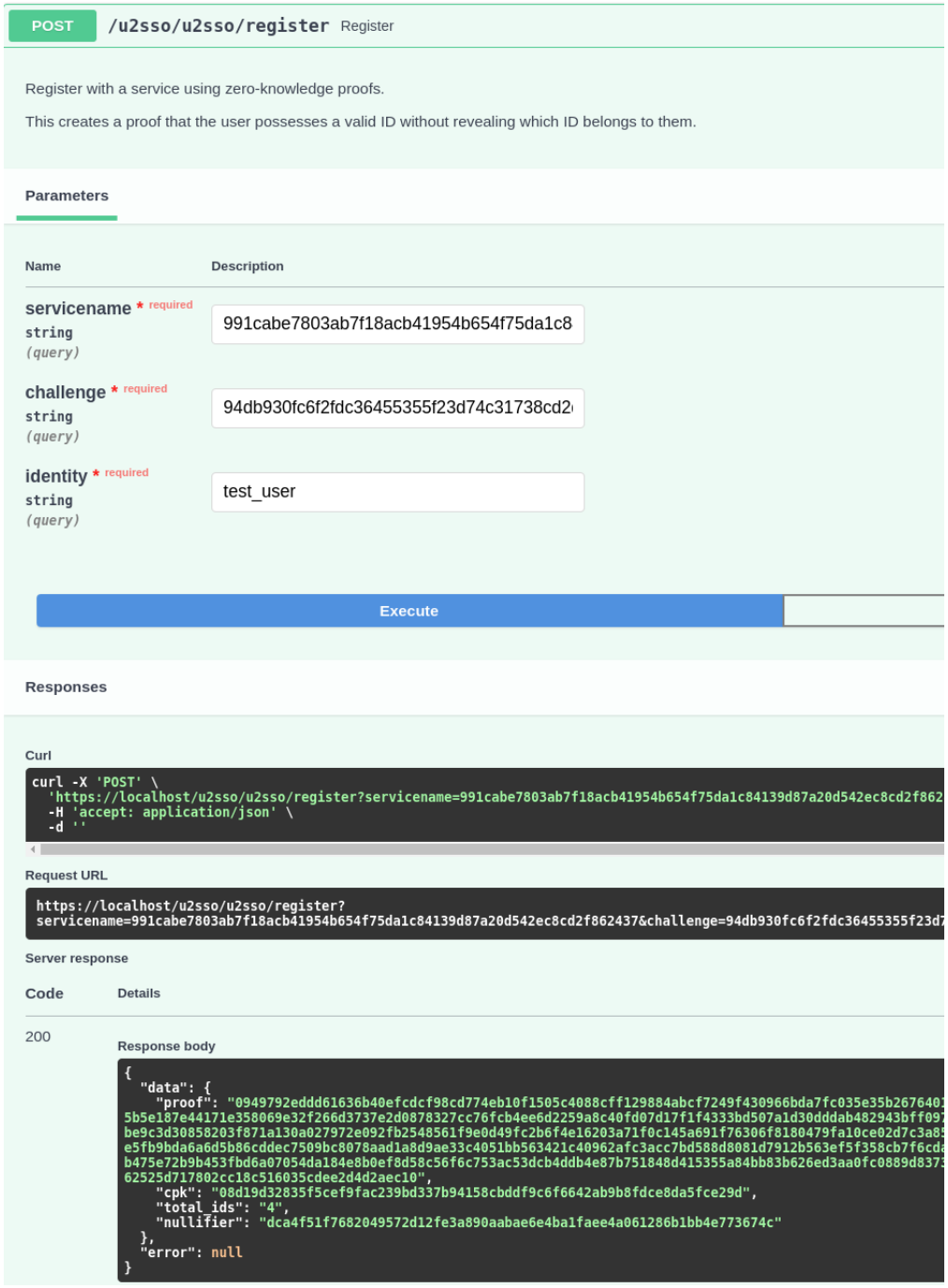


Figure 5.5: Service-Specific Pseudonym Derivation API endpoint. The screenshot demonstrates the registration process with a service using zero-knowledge proofs. Three critical parameters are provided: the service name identifier, a challenge for fresh proof generation, and the user identity. The endpoint returns a privacy-preserving pseudonym cpk, a zero-knowledge proof of legitimate identity membership, and a nullifier that prevents double registration.

5.2.5 Credential Issuance Request

To obtain a verifiable credential bound to their service-specific pseudonym, the user must first request a credential offer from the issuer. This implementation corresponds to step (8) in Figure 4.0 (request credential from VCI), where the user initiates the credential acquisition process.

```
function requestCredentialOffer(credentialMetadata, subjectData):
    // Create the credential offer request
    offerRequest = {
        "metadata_credential_supported_id": credentialMetadata,
        "credential_subject_data": subjectData,
        "offer_validity_seconds": validityPeriod
    }

    // Send request to the Issuer
    response = sendPostRequest(issuerUrl +
                               "/oid4vc/credential/offer", offerRequest)

    // Extract the credential offer deeplink
    offerDeeplink = response.offer_deeplink

    return offerDeeplink
end function
```

Listing 6: Requesting a Credential Offer

Figure 5.6 shows how to make a credential issuance offer using the API endpoint. Listing 6 shows how the system makes a credential offer request with the right metadata and subject data, sends it to the issuer according to the OpenID4VCI protocol, and gets a credential offer deeplink from the response. This deeplink has encoded information about the endpoint of the credential issuer, the type of credential being offered, a code that has already been approved for redeeming the credential, and optional authentication parameters.
An example credential offer request demonstrates the structure required for university degree credentials:

```
{
  "metadata_credential_supported_id": "tergum_dummy_jwt",
  "credential_subject_data": {
    "degree": {
      "type": "MasterDegree",
      "name": "Master of Science",
      "average_grade": 5
    }
  },
  "offer_validity_seconds": 2592000
}
```

Listing 7: Example Credential Offer Request

The issuer responds with a credential offer deeplink that follows the OpenID4VCI protocol:

```
{
  "offer_deeplink": "openid-credential-offer://?credential_offer=..."
}
```

Listing 8: Credential Offer Response

The user will later use this deeplink in conjunction with their derived `cpk` to fetch the credential bound to their service-specific pseudonym, ensuring that the credential can be used privately without enabling cross-service tracking.

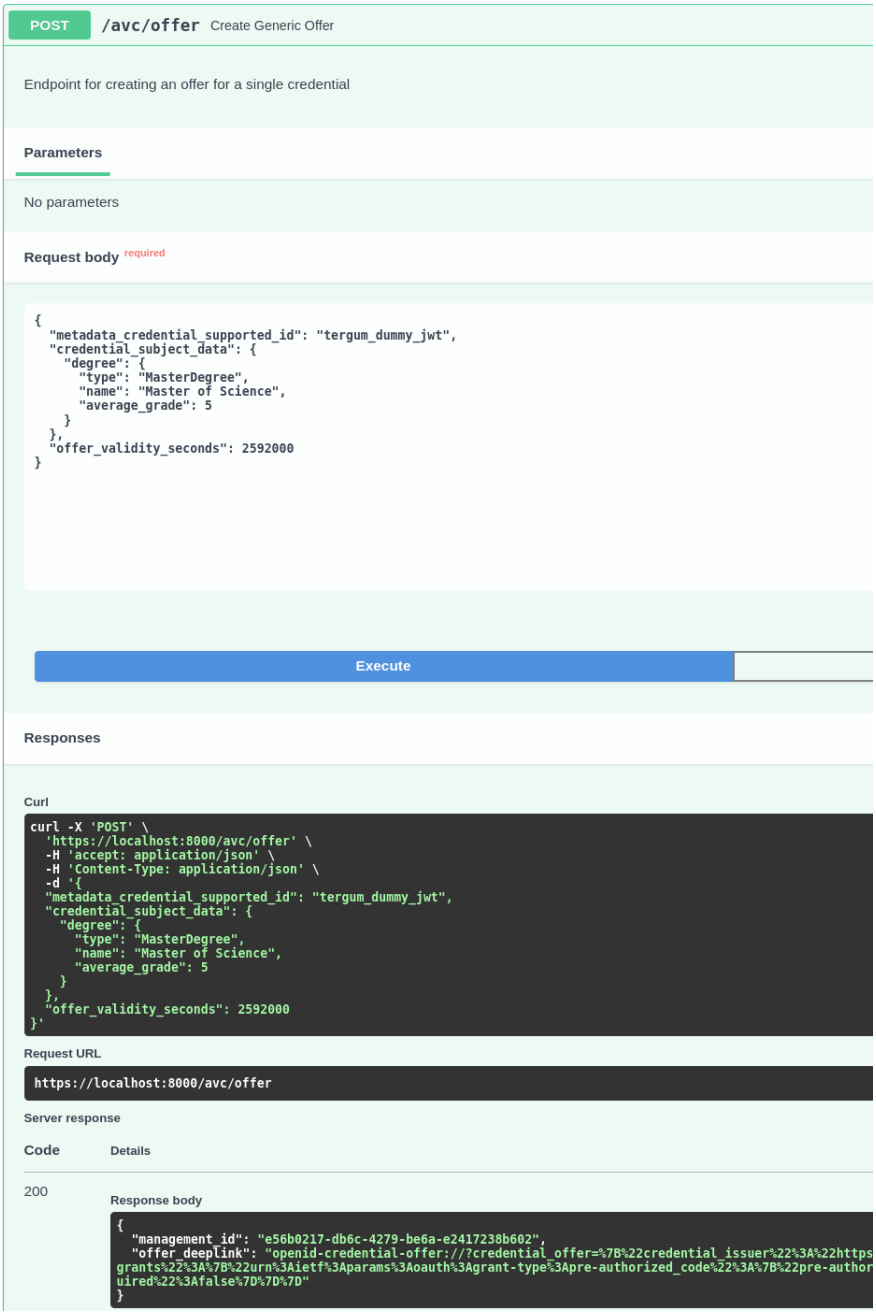


Figure 5.6: Credential Issuance Offer API endpoint. The screenshot shows the creation of a credential offer for a university degree with specific attributes (Master of Science with grade 5). The endpoint generates a credential offer deeplink following the OpenID for Verifiable Credential Issuance (OpenID4VCI) protocol, which will later be used with the derived pseudonym to fetch a privacy-preserving credential.

5.2.6 Binding Verifiable Credentials to Service-Specific Pseudonyms

After obtaining a credential offer deeplink from the Issuer and deriving a service-specific pseudonym (cpk), the user must retrieve the actual credential bound to this pseudonym. This implementation corresponds to steps (9)–(10) in Figure 4.0 (VCI issues credential bound to pseudonym, credential delivery to user), representing the critical innovation in our AVC framework.

```
function retrieveCredentialWithPseudonym(username, cpk, deeplink):
    // Decode the credential offer from the deeplink
    offer ← parseCredentialOfferDeeplink(deeplink)

    // Exchange pre-authorized code for access token
    tokenResponse ← requestToken(offer.issuerUrl,
                                 offer.preAuthorizedCode)
    accessToken ← tokenResponse.accessToken
    nonce ← tokenResponse.nonce

    // Create proof of possession for the pseudonym
    proofJwt ← createProof(nonce, offer.issuerUrl)

    // Build credential request with the pseudonym
    credentialRequest ← {
        "cpk": cpk,
        "format": "jwt_vc_json",
        "proof": {
            "proof_type": "jwt",
            "jwt": proofJwt
        },
        "credential_definition": {
            "types": offer.credentialTypes
        }
    }

    // Request credential bound to the pseudonym
    credential ← sendCredentialRequest(
        offer.issuerUrl + "/avc/credential",
        accessToken,
        credentialRequest
    )

    // Store the credential for the user
    storeCredential(username, credential)

    return credential
end function
```

Listing 9: Retrieving Credential Bound to Pseudonym

Figure 5.7 shows the credential retrieval process with pseudonym binding via the API endpoint. Listing 9 depicts the entire AVC credential binding process: the system decodes the credential offer from the deeplink, exchanges the pre-authorized code for an access token, generates proof of possession for the pseudonym, and creates a credential request that includes the service-specific pseudonym rather than a persistent identifier. This is the key innovation, in which credentials are linked to U2SSO-derived pseudonyms rather than DIDs.

Our AVC framework introduces an innovative approach to binding credentials to service-specific pseudonyms. The key innovation lies in how the issuer creates the credential:

```
function issueCredentialWithPseudonymBinding(credentialRequest,
                                             accessToken, nonce):
    // Verify the access token matches a valid offer
    offer ← findOfferByAccessTokenAndNonce(accessToken, nonce)

    // Extract the pseudonym from the request
    pseudonym ← credentialRequest.cpk

    // Build the verifiable credential with pseudonym binding
    credential ← buildVerifiableCredential(
        credentialType: offer.credentialType,
        credentialSubject: offer.subjectData,
        validFrom: offer.validFrom,
        validUntil: offer.validUntil,
        pseudonym: pseudonym  // Bind credential to the pseudonym
    )

    // Sign the credential with the issuer's key
    signedCredential ← signCredential(credential, issuerPrivateKey)

    // Mark the offer as used
    updateOfferStatus(offer, "ISSUED")

    return signedCredential
end function
```

Listing 10: Issuer Binding Credential to Pseudonym

Listing 10 shows how the issuer checks that the access token matches a valid offer, gets the U2SSO-derived pseudonym from the request, and makes a credential that can be verified with pseudonym binding instead of DID binding. After that, the issuer signs the credential and marks the offer as used. This method is different from how OpenID is usually used because it ties credentials directly to service-specific pseudonyms. This makes sure that the credential is cryptographically linked to the user's pseudonym while keeping U2SSO's unlinkability features.

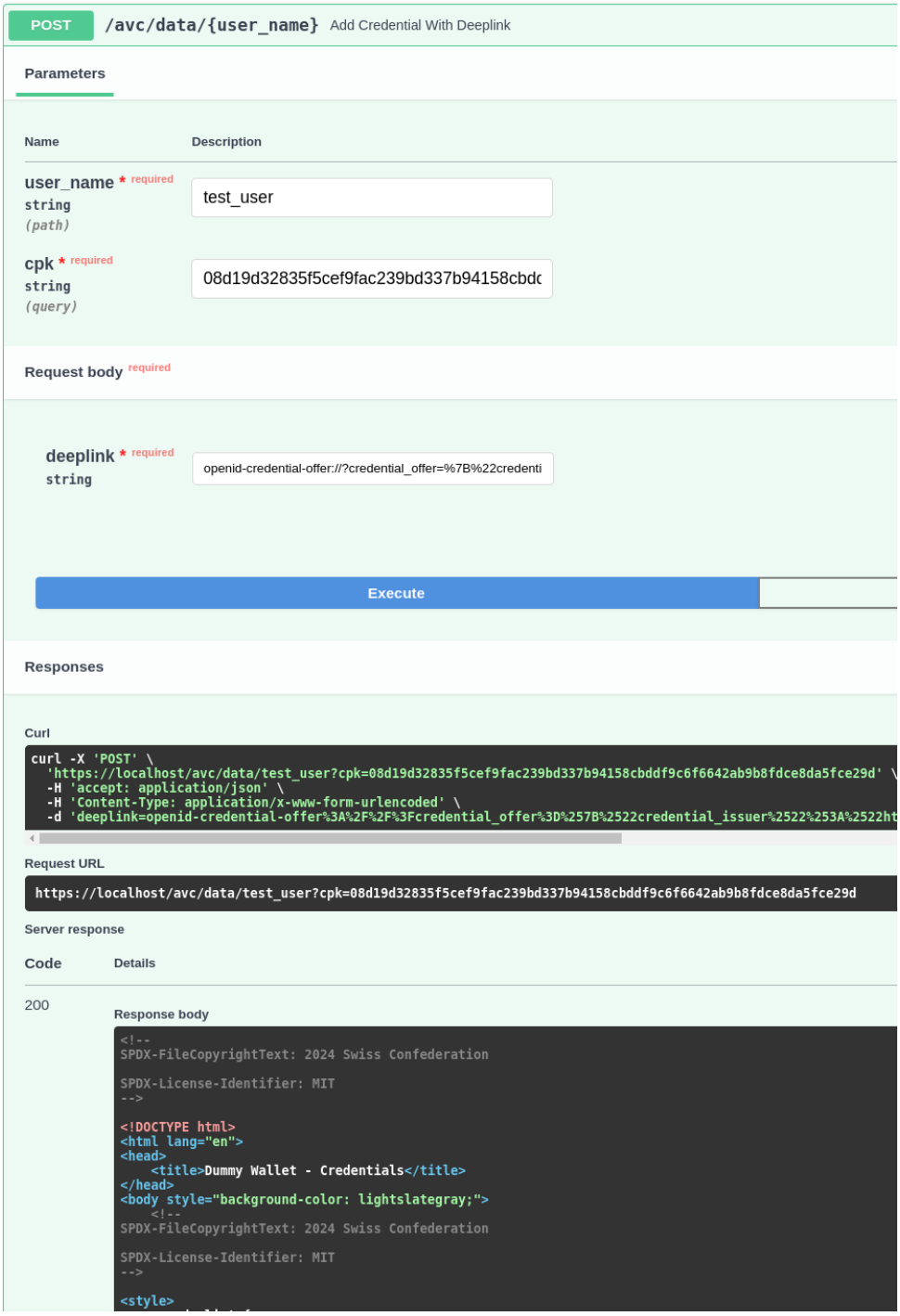


Figure 5.7: The Credential Retrieval API endpoint shows how to link a verifiable credential to a pseudonym that is only used for that service. The user gives their username, the derived pseudonym cpk, and the deeplink for the credential offer. This binding makes sure that credentials cannot be linked across services, but they can still be verified cryptographically. This is the main privacy innovation of the AVC framework.

5.2.7 Privacy-Preserving Credential Presentation

Once a user has obtained a verifiable credential bound to their service-specific pseudonym (cpk), they can use it to prove attributes to a Service Provider while preserving their privacy. This implementation corresponds to steps (12)–(16) in Figure 4.0 (register pseudonym with Service Provider, verify anonymity set, dual verification of U2SSO proof and credential, registration confirmation).

```
function presentCredential(username,
                           authRequestUri,
                           cpk,
                           nullifier,
                           proof,
                           ringSize):
    // Fetch the verifier's request object containing
    // presentation requirements
    requestObject ← fetchRequestObject(authRequestUri)

    // Extract credential types required by the verifier
    credentialTypes ← extractRequiredTypes(
                              requestObject.presentationDefinition
                              )

    // Retrieve matching credential from user's wallet
    credential ← retrieveMatchingCredential(username, credentialTypes)

    // Create a verifiable presentation containing the credential
    vp ← createVerifiablePresentation(credential)

    // Bundle the presentation with the pseudonym proof components
    verifiableToken ← {
        "vp": vp,
        "cpk": cpk,
        "nullifier": nullifier,
        "proof": proof,
        "ring_size": ringSize
    }

    // Create presentation submission mapping
    presentationSubmission ← createPresentationSubmission(credential)

    // Send the presentation to the verifier
    response ← sendPresentation(
        username,
        authRequestUri,
        presentationSubmission,
        verifiableToken
    )

    return response
end function
```

Listing 11: Privacy-Preserving Credential Presentation

Figure 5.8 demonstrates the credential presentation and verification process through the API endpoint. The listing 11 shows how the system fetches the verifier's presentation requirements, extracts required credential types, retrieves matching credentials from the user's wallet, and creates a verifiable presentation. The critical innovation is bundling the presentation with U2SSO proof components (cpk, nullifier, proof, ring_size) that enable privacy-preserving verification.
The verification process combines standard verifiable credential verification with the AVC framework's zero-knowledge proof verification:

```
function verifyPresentation(requestObject, presentation):
    // Extract verifiable token and username
    verifiableToken ← presentation.verifiableToken
    username ← presentation.username

    // 1. Extract and parse the credential JWT
    credential ← extractCredential(verifiableToken)

    // 2. Verify the credential's cryptographic integrity
    verifyJwtSignature(credential)

    // 3. Check credential expiration
    checkExpiration(credential)

    // 4. Verify credential matches presentation definition requirements
    validateAttributes(credential, requestObject.presentationDefinition)

    // 5. Verify pseudonym binding
    cpk ← verifiableToken.cpk
    credentialCpk ← extractCpkFromCredential(credential)

    if cpk not equal credentialCpk:
        raise Error("Pseudonym in credential
                     does not match presentation")

    // 6. Verify zero-knowledge proof components
    verifyZeroKnowledgeProof(
        username,
        cpk,
        verifiableToken.nullifier,
        verifiableToken.proof,
        verifiableToken.ringSize
    )

    return {
        "status": "success",
        "attributes": validatedAttributes,
        "cpk": cpk
    }
end function
```

Listing 12: Verifier Credential Verification

Listing 12 shows how AVC's dual verification works. The system first does standard VC checks like checking the JWT signature, expiration, and attributes. Then it does AVC-specific pseudonym binding checks and U2SSO zero-knowledge proof checks. This makes sure that the user has a valid identity in the Identity Registry, that the pseudonym is correctly derived from this identity, that the credential is linked to the same pseudonym, and that the user cannot be tracked across different services.

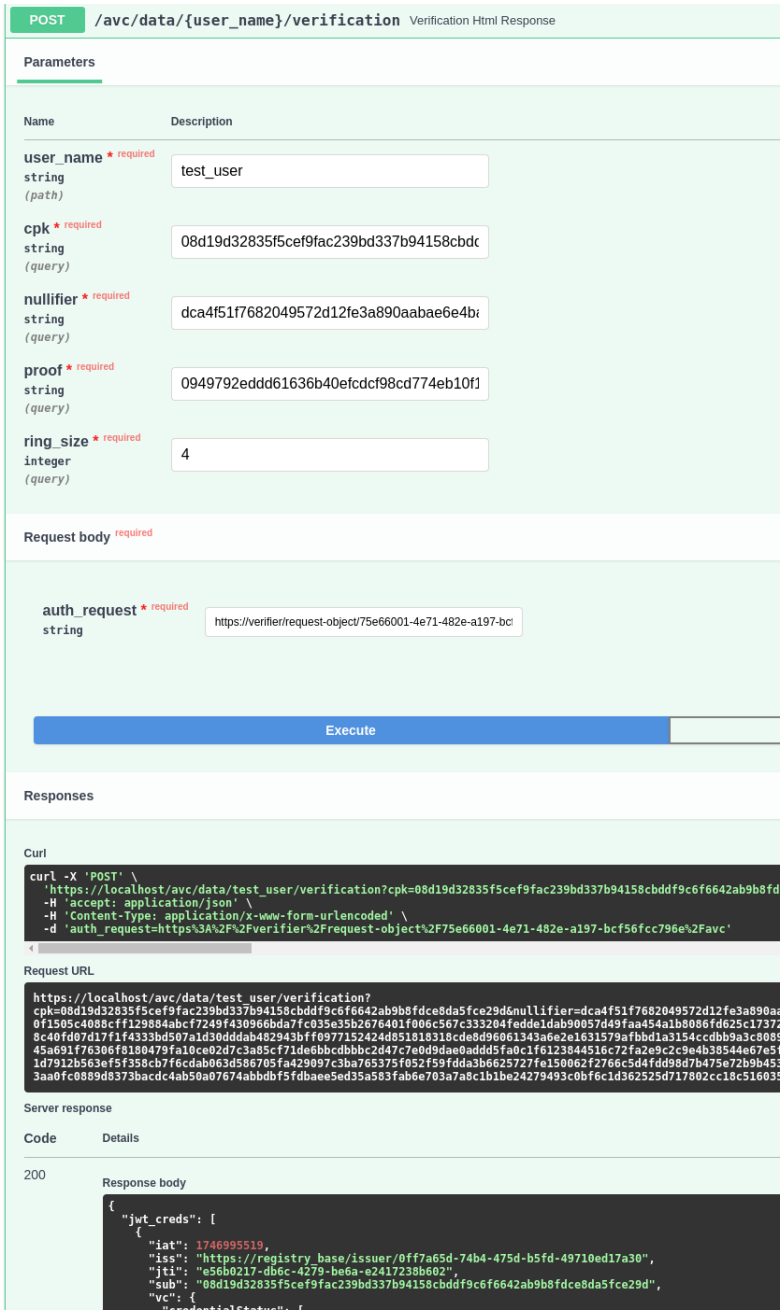


Figure 5.8: Credential Presentation API endpoint demonstrating the verification process. The endpoint receives multiple privacy-preserving components: the username, the service-specific pseudonym cpk, the nullifier preventing Sybil attacks, the zero-knowledge proof of legitimate identity, and the ring size for anonymity. These components collectively enable attribute verification without revealing the user's global identity or enabling cross-service tracking.

5.2.8 Privacy-Preserving Authentication

After initial registration, users can authenticate to the service using their derived pseudonym (cpk) without requiring computationally intensive zero-knowledge proof generation. This implementation corresponds to steps (17)–(19) in Figure 4.0 (generate authentication signature, authenticate with Service Provider, verify authentication), leveraging U2SSO's efficient challenge-response protocol.

```
function authenticateWithService(identity, servicename, challenge):
    // Convert input parameters to required format
    service_name_bytes ← hexDecode(servicename)
    challenge_bytes ← hexDecode(challenge)

    // Load the user's master secret key
    msk_bytes ← loadPasskey(identity)

    // Fetch registered services from IdR
    service_list ← fetchServices()

    // Generate authentication digital signature
    signature ← generateAuthProof(
        service_name_bytes,
        challenge_bytes,
        msk_bytes,
        service_list,
        length(service_list)
    )

    if not signature:
        raise Error("Authentication proof generation failed")

    return signature
end function
```

Listing 13: User Authentication Process

Figure 5.9 depicts the authentication digital signature generation process via the API endpoint. Listing 13 demonstrates U2SSO's lightweight authentication process: input parameters are converted to the required format, the user's master secret key is loaded, registered services are retrieved from the IdR, and an authentication digital signature is generated based on the service name, challenge, and master secret key. This digital signature demonstrates knowledge of the master secret key without disclosing it and is tied to the service-specific pseudonym.
The authentication digital signature is cryptographically bound to the service-specific pseudonym and the challenge provided by the Verifier. When received by the Verifier, it is verified against the stored user account through a comprehensive validation process:

```
function verifyAuthentication(username, cpk, signature):
    // Convert input parameters
    cpk_bytes ← hexDecode(cpk)

    // Find user account using username and cpk
    account ← findAccount(username, cpk_bytes)

    if not account:
        raise Error("Account not found")

    // Fetch services for verification
    service_list ← fetchServices()

    // Get service name bytes
    service_name_bytes ← getServiceNameBytes()
```

```
16
17      // Verify authentication digital signature
18      verified ← verifyAuthSignature(
19          signature,
20          service_name_bytes,
21          challenge,
22          cpk_bytes,
23          service_list,
24          length(service_list)
25      )
26
27      if not verified:
28          raise Error("Authentication failed")
29
30      return "Authentication successful"
31  end function
```

Listing 14: Verifier Authentication Process

Listing 14 shows how the verifier converts input parameters, finds the user account using the username and pseudonym, fetches services for verification, and verifies the authentication digital signature using U2SSO's verification algorithm. The process confirms the user's legitimate ownership of the pseudonym without revealing their global identity.

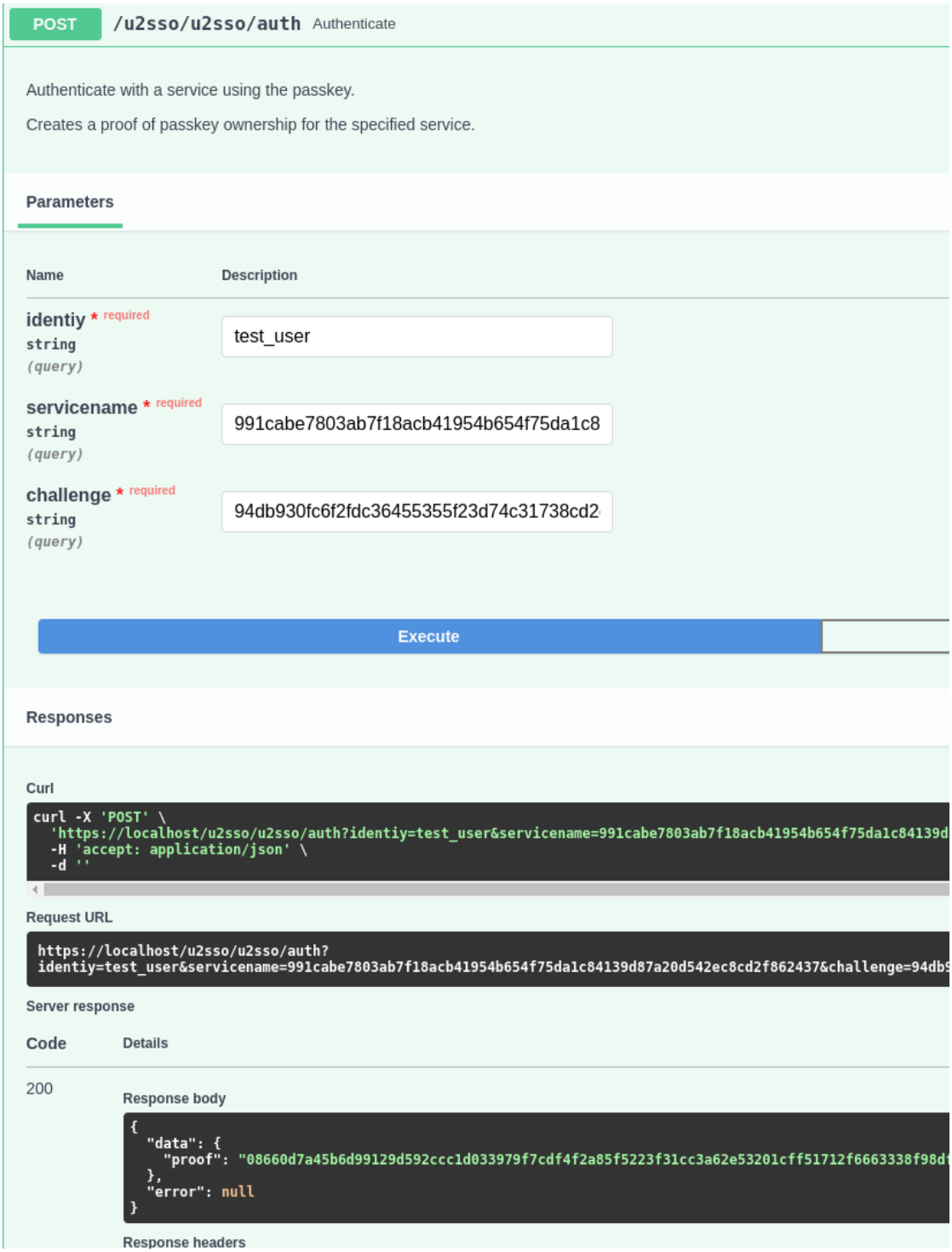


Figure 5.9: Authentication DigitalSignature Generation API endpoint. The screenshot shows how a user generates an authentication digital signature using their identity, the service name, and a challenge from the Verifier. This lightweight authentication proof is cryptographically bound to the service-specific pseudonym and unique challenge, ensuring both security and privacy without requiring repeated zero-knowledge proof generation.

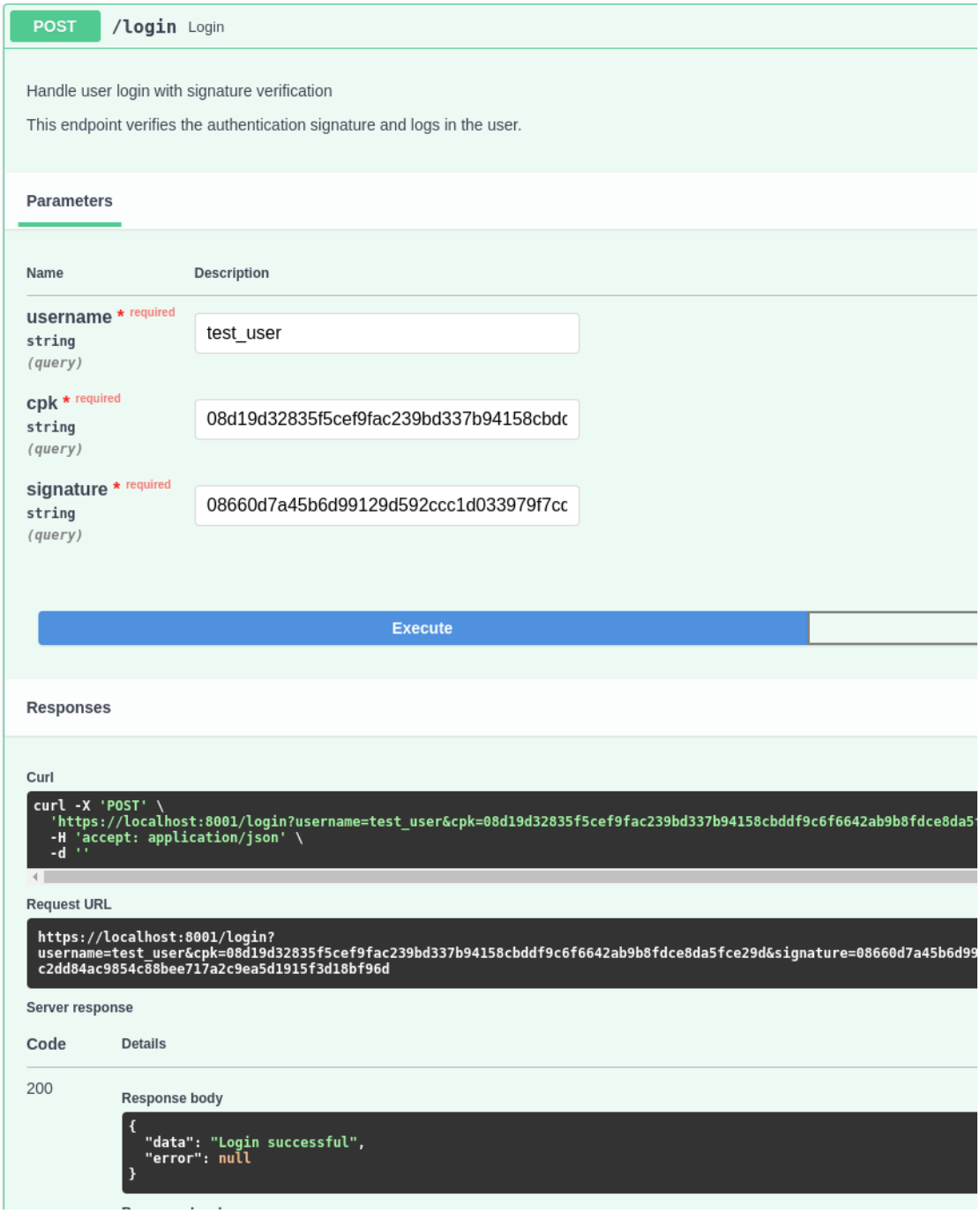


Figure 5.10: Authentication Verification API endpoint showing the process of verifying a user's authentication digital signature. The endpoint takes the username, service-specific pseudonym cpk, and the authentication digital signature as input. The successful verification confirms the user's legitimate ownership of the pseudonym without revealing their global identity, combining strong security with privacy preservation for recurring interactions.

This authentication mechanism maintains the privacy properties of the AVC system through several key guarantees provided by U2SSO. By separating the computationally intensive registration process (using zero-knowledge proofs) from the simpler authentication process, the AVC framework delivers both strong security and practical efficiency while maintaining unlinkability across different services.

6 Experience

The implementation of the Anonymous Verifiable Credentials (AVC) framework in the Swiss eLFA infrastructure posed numerous technical challenges that provided valuable learning opportunities. This chapter discusses the major difficulties encountered, solutions developed, and insights gained during the development process.

The initial challenge was to gain a thorough understanding of the existing eLFA system. Despite being a proof of concept, the codebase was large and complex, containing several components such as the Issuer Agent, Verifier Agent, Base Registry, and various wallet implementations. The most significant challenge was a lack of comprehensive documentation for the proof-of-concept implementation, which required extensive reverse engineering to understand the system's workflows and component interactions. This reverse engineering process entailed not only identifying which endpoints and functions needed to be executed, but also determining the proper order of operations.

The system's heavy reliance on OpenID protocols proved both beneficial and challenging. On the plus side, the extensive OpenID documentation provided guidance on expected workflow sequences, allowing me to navigate the codebase more methodically. When I encountered a specific step in the implementation, I could refer to the OpenID specifications to determine what steps should be taken next, which helped me locate the corresponding code sections. However, the tight integration with OpenID caused complications. Because my goal was to integrate the AVC framework as seamlessly as possible into the existing proof of concept, I had to examine numerous small parameters and intricate details in the OpenID specifications. This thorough examination of OpenID's technical nuances was time-consuming, but necessary for maintaining protocol compliance while introducing our privacy-enhancing changes.

One of the most significant technical challenges was allowing the Issuer Agent to associate issued Verifiable Credentials with the pseudonyms generated by the U2SSO protocol. The main issue was that the pseudonyms generated by U2SSO were incompatible with the standard OpenID for Verifiable Credentials (OID4VC) data structures and workflows. The existing eLFA implementation relied on traditional Decentralized Identifiers (DIDs) for credential binding, rather than the cryptographically derived pseudonyms from our privacy-preserving framework. The solution required developing a custom extension for the OpenID credential issuance process. Instead of attempting to force U2SSO pseudonyms into existing OpenID fields, which would have compromised protocol compliance, I added a new parameter designed specifically for pseudonym-based credential binding. This approach enabled the Issuer to

process AVC requests while remaining backward compatible with standard OID4VC implementations. While this solution may appear simple in retrospect, the development process was complicated and iterative, necessitating careful analysis of the protocol's extension mechanisms as well as extensive testing to ensure protocol compliance. Integrating the U2SSO library, which is written in C, into the Python-based eLFA infrastructure presented new challenges. I needed to develop a Python interface that could seamlessly convert data formats between Python and C representations while ensuring proper memory management and type conversions. When executing C methods from the Python framework, one particularly difficult issue arose: the C library generated new threads that did not inherit the parameters specified at initialization. Finding and resolving this threading problem necessitated extensive debugging and a thorough understanding of both Python's Global Interpreter Lock and C's threading model. The solution entailed implementing proper thread-local storage and ensuring that all relevant context was explicitly passed to C library functions. Furthermore, I gained extensive experience with containerization and networking configurations. Since each entity in the eLFA system operates within its own Docker container, introducing new AVC components necessitated the creation of additional containers while maintaining proper network connectivity between all services. This provided an opportunity to learn more about container orchestration, network routing, and service discovery mechanisms. Understanding how containers communicate, managing port configurations, and ensuring proper DNS resolution between services became critical for successful integration.

The implementation process was challenging, but ultimately rewarding. The challenges required me to gain a thorough understanding of complex distributed systems, protocol design principles, and the intricate details of modern identity frameworks. Working with real-world government infrastructure gave me invaluable insights into the practical constraints and requirements that theoretical research must consider. The experience demonstrated that, while privacy-enhancing technologies such as AVC can be successfully integrated into existing systems, compatibility, standard compliance, and operational requirements must all be carefully considered.

7 Conclusion

This thesis introduced Anonymous Verifiable Credentials (AVC), a privacy-preserving framework that combines User-issued Unlinkable Single Sign-On (U2SSO) with Verifiable Credentials (VC) to achieve unlinkability, Sybil resistance, and attribute verification all at once. The main new idea is to link credentials to service-specific pseudonyms instead of permanent identifiers. This stops tracking across services while still allowing verification. We have made the following contributions: (1) a unified framework that keeps the security features of both U2SSO and VC systems while fixing their problems; (2) a new way of binding credentials that stops linkability across services; (3) a working version of this in Switzerland's electronic Provisional Driving License Program (eLFA); and (4) an expansion of existing standards like OpenID for Verifiable Credentials. The AVC framework has a big effect on how we manage our digital identities. It gives people back control of their personal data while still allowing service providers to follow the rules. Our work fills in the gaps between centralized and decentralized identity models by showing that privacy and verification can work together. The fact that the Implementation of AVC in Swiss eLFA program worked shows that privacy-enhancing methods can be used in real-world identity systems. As society aims to create rules to stress minimizing data, frameworks like AVC provide a way to comply with verification requirements while still protecting user privacy. The future of digital identity is not about choosing between privacy and usefulness; it's about making designs that do both, so that people can be part of the digital world without giving up their privacy or freedom.

Bibliography

[1] Regulation (eu) 2016/679 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation), April 2016.

[2] Jayamine Alupotha, Mariarosaria Barbaraci, Ioannis Kaklamanis, Abhimanyu Rawat, Christian Cachin, and Fan Zhang. Anonymous self-credentials and their application to single-sign-on. *IACR Cryptol. ePrint Arch.*, page 618, 2025.

[3] Bundesamt für Justiz (BJ), Bundesamt für Strassen (ASTRA), Vereinigung der Schweizerischen Strassenverkehrsämter (asa), and Strassenverkehrsamt Appenzell Ausserrhoden. Elektronischer lernfahrausweis (elfa). `https://www.eid.admin.ch/de/pilotprojekte`, 2025. Zugriff am 18. April 2025.

[4] Jan Camenisch, Bruno Crispo, Simone Fischer-Hübner, Ronald Leenes, and Giovanni Russello, editors. *Privacy and Identity Management for Life - 7th IFIP WG 9.2, 9.6/11.7, 11.4, 11.6/PrimeLife International Summer School, Trento, Italy, September 5-9, 2011, Revised Selected Papers*, volume 375 of *IFIP Advances in Information and Communication Technology*. Springer, 2012.

[5] Melissa Chase, Sarah Meiklejohn, and Greg Zaverucha. Algebraic macs and keyed-verification anonymous credentials. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 1205–1216. ACM, 2014.

[6] e-id-admin. eidch-pilot-android-wallet. `https://github.com/e-id-admin/eidch-pilot-android-wallet`, 2024. Open Source Repository of Android wallet application for eLFA pilot. Deprecated.

[7] e-id-admin. eidch-pilot-elfa-base-infrastructure. `https://github.com/e-id-admin/eidch-pilot-elfa-base-infrastructure`, 2024. OpenID4VC Proof of Concept. Please be aware that this repository serves as example and not as reference code.

[8] e-id-admin. eidch-pilot-ios-wallet. `https://github.com/e-id-admin/eidch-pilot-ios-wallet`, 2024. Open Source Repository of iOS wallet application for eLFA pilot. Deprecated.

[9] Ethereum Foundation. Welcome to ethereum. `https://ethereum.org/en/foundation/`, 2024.

[10] Daniel Fett, Kristina Yasuda, and Brian Campbell. Selective Disclosure for JWTs (SD-JWT). Internet-Draft draft-ietf-oauth-selective-disclosure-jwt-22, Internet Engineering Task Force, May 2025. Work in Progress.

[11] Sovrin Foundation. Sovrin: A protocol and token for self-sovereign identity and decentralized trust, 2018. Available at: `https://sovrin.org/library/sovrin-protocol-and-token-white-paper/`.

[12] Le Gao, Jiaxin Yu, Junzhe Zhang, Yin Tang, and Quansi Wen. AASSI: A self-sovereign identity protocol with anonymity and accountability. *IEEE Access*, 12:58378–58394, 2024.

[13] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In Michael D. Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 519–535. USENIX Association, 2021.

[14] Hyperledger Foundation. Announcing hyperledger anoncreds: Open source, open specification privacy preserving verifiable credentials, November 2022. Accessed 2025-06-26.

[15] Hyperledger Indy Contributors. Hyperledger Indy Documentation. `https://hyperledgerindy.readthedocs.io/en/latest`, 2025.

[16] Dimitrios Kasimatis, Sam Grierson, William J. Buchanan, Chris Eckl, Pavlos Papadopoulos, Nikolaos Pitropakis, Christos Chrysoulas, Craig Thomson, and Baraq Ghaleb. DID:RING: ring signatures using decentralised identifiers for privacy-aware identity proof. In *IEEE International Conference on Cyber Security and Resilience, CSR 2024, London, UK, September 2-4, 2024*, pages 866–871. IEEE, 2024.

[17] Torsten Lodderstedt, Kristina Yasuda, and Tobias Looker. Openid for verifiable credential issuance 1.0. `https://openid.net/specs/openid-4-verifiable-credential-issuance-1_0.html`, December 2024. OpenID Foundation, Digital Credentials Protocols Working Group.

[18] Nitin Naik and Paul Jenkins. uport open-source identity management system: An assessment of self-sovereign identity and user-centric data platform built on blockchain. In *IEEE International Symposium on Systems Engineering, ISSE 2020, Vienna, Austria, October 12 - November 12, 2020*, pages 1–7. IEEE, 2020.

[19] Reyhaneh Rabaninejad, Behzad Abdolmaleki, Sebastian Ramacher, Daniel Slamanig, and Antonis Michalas. Attribute-based threshold issuance anonymous counting tokens and its application to sybil-resistant self-sovereign identity. *IACR Cryptol. ePrint Arch.*, page 1024, 2024.

[20] Manu Sporny, Dave Longley, Markus Sabadello, Drummond Reed, Orie Steele, and Christopher Allen. Decentralized identifiers (dids) v1.0. Recommendation DID-1.0, W3C, 2022.

[21] Manu Sporny, Dave Longley, Kristina Yasuda, and Orie Steele. Verifiable credentials data model v2.0. `https://www.w3.org/TR/vc-data-model-2.0/`, April 2024. W3C Candidate Recommendation.

[22] Stefano Tessaro and Chenzhi Zhu. Revisiting BBS signatures. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part V*, volume 14008 of *Lecture Notes in Computer Science*, pages 691–721. Springer, 2023.

[23] Liang Yang, Jie Zhang, Rui Chen, and Shujie Gao. Unlinkable verifiable credentials using dynamic accumulators. In *2021 IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2021.