Credential Issuance Transparency: A Privacy-preserving Audit Log of Credential Issuance

Anonymous Author(s)

Abstract Digital identity ecosystems are rapidly transforming the landscape of identity management. Self-Sovereign Identity (SSI) promises to enhance the individual's agency over their identity; and related concepts form core parts of the EU's upcoming eIDAS 2.0 regulation. Yet, this privacy-preserving technology must become *less* privacy-preserving for one overlooked party – credential issuers. Issuers are trusted to validate users' attributes and attest to them. Thus, a compromised or misbehaving issuer is an immense threat, being able to issue credentials that allow them to impersonate anyone.
We address this concern by introducing Credential Issuance Transparency (CIT), a transparency framework for the issuance of identifying credentials. We take concepts from the Web PKI's Certificate Transparency (CT), such as using public append-only logs, but adapt them to a privacy-preserving SSI world. In contrast to CT, the public logs of CIT disclose no information about a credential or its subject. Still, genuine subjects can monitor the log to discover mis-issued credentials that would allow an attacker to impersonate them; and empowered by non-interactive zero-knowledge proofs, verifiers can mandate correct logging.
CIT is practical. It adds a neglectable overhead of less than 2ms to credential showing. Daily monitoring for mis-issuance requires less than 1GB of data to be downloaded, and less than 10 seconds of computation to be invested. This makes CIT an important step towards SSI's organizational acceptance and real-world feasibility.

Keywords: self-sovereign identity · transparency logs · credential issuance · unlinkability · non-interactive zero-knowledge proofs · certificate transparency

1 Introduction

Digital identities play a pivotal role in our daily interactions. They facilitate a secure online environment and fosters trust in digital interactions. However, as Microsoft's Chief Identity Architect Kim Cameron, aptly identified almost 20 years ago – the internet was built without an identity layer [5]. This absence of a standardized framework for managing digital identities in the foundational design of the internet has led to a diverse range of fragmented solutions for digitally identifying and authenticating individuals. Today, centralized solutions that enable users to use a single set of credentials across multiple services are

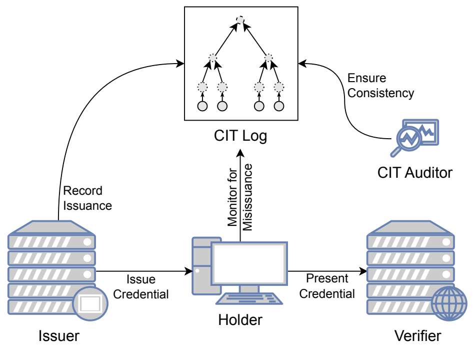

Figure 1: Credential Issuance Transparency architecture

commonly in use [33]. However, credential-based approaches under the banner of Self-Sovereign Identity (SSI) have emerged, promising to enhance privacy and restore user agency.

The European Union, recognizing this trend, is looking to incorporate SSI paradigms into the eIDAS 2.0 regulation [12]. This regulation thrusts many of the technology's challenges into the spotlight of forced practicability, with vision giving way to viability.

Challenge. A crucial open research area of such a privacy-preserving technology is the need for *transparency*. By allowing public visibility into both theories and practices of operation, corruption risks can be reduced, shortcomings can be noticed, and reputation-based penalties can be inflicted.

In computer science, such enforced transparency is not a novel concept; it is used to great effect in the Web PKI, where *Certificate Transparency* (CT) should serve as a safeguard against misbehavior by certificate authorities [20]. However, it is not clear how to translate such technologies to a privacy-preserving world. On the web, malicious actors commonly crawl certificate issuance audit logs – which are public by design – to discover target hosts for scanning campaigns [31,26,22]. In a credential ecosystem, which holds highly sensitive personal information, such information leakage must be avoided at all costs.

Contributions. We introduce *Credential Issuance Transparency* (CIT), a privacy-preserving audit logging mechanism for the issuance of identifying credentials. CIT takes CT's append-only log concept, but fully blinds the log entries, removing all sensitive information from them while retaining public auditability of the log's append-only nature as well as user discoverability. Instead of the publicly visible domain name, we rely on a randomly blinded commitment to obscure the credential subject's identity.

By also borrowing CT's trust model, we rely on truthful verifiers to enforce a logging mandate; in order for a credential to be accepted, it must have been logged in an appropriate manner. Here, our log entries' blinded nature introduce an additional validation challenge. We overcome this by employing non-interactive zero-knowledge proofs (NIZKPs) to demonstrate the claimed log entry's consistency with the presented credential.

CIT (illustrated in Figure 1) thus allows users to monitor for issuance of any identifying credentials that could serve to impersonate them. Such an auditing framework is crucial in bringing self-sovereign identity principles closer to acceptability in legislative circles.

Outline. We first introduce relevant concepts (Section 2). In Section 3, we list involved entities and concepts; formally describing our system's design goals and threat model. In Section 4, we introduce CIT, its building blocks, and the issuance, showing, and validation processes. We particularly emphasize the importance of the unlinkability of user identifiers present in the log and zero-knowledge proofs for binding credentials and maintaining user privacy. In Section 5, we evaluate CIT's performance and practicability. Finally, we discuss possible adjustments to the protocol and related work in Section 6.

2 Background

2.1 Credential Schemes

A *credential* is a declaration about a *user* (or *holder*) made by an *issuer*. It may encompass various *attributes*, such as the user's name, physical characteristics, group affiliations, or other personal details that the issuer has verified. Users holding a credential can present it to a *verifier*. If the verifier trusts the issuer, this convinces the verifier that the user possesses the claimed attributes. The combined procedures for issuing and presenting a credential constitute a *credential scheme*.

Traditional credential schemes often adopt a centralized structure, where the issuer's server stores all user information. In contrast, user-centric digital identity paradigms such as Self-Sovereign Identity (SSI) enable individuals to independently handle digital attestations of their identity attributes and cryptographic keys for authentication. This is often done on users' personal devices, such as mobile phones, through a so-called "wallet" application. Such attestations contain cryptographic evidence of integrity, commonly in the form of a digital signature created by the issuer. This signature renders the information verifiable by machines. This concept led to the common term "verifiable credentials" (VCs) being used for the associated attestations [32]. We further describe the VC presentation process in Appendix A.

SSI concepts are seeing adoption in the European Union's eIDAS 2.0 regulation that introduces a "European Digital Wallet" [12] capable of receiving, storing, and showing digital credentials with high levels of assurance.

2.2 Transparency Logs

Transparency logs are a reliable mechanism for storing and showcasing information. They ensure that all users can independently confirm they are viewing the same entries irrespective of their role. This functionality is pivotal in maintaining integrity and trust within a system as it significantly reduces the chances of tampering or misrepresenting data.

These logs were initially introduced in the Web PKI. Here, they serve to record all certificates issued by Certificate Authorities (CAs). This allows mistaken or malicious certificate issuance to be detected. The resulting system is called Certificate Transparency (CT) [20].

Recognizing the potential of these transparency logs, developers have started applying them in various other ecosystems. Notably, they are now used to offer tamper-proofing in diverse sectors such as binary transparency, supply chain transparency, and data access transparency [1,19,18]. Binary transparency logs can help in tracking software updates and ensuring that they are consistent across all users [24]. Supply Chain transparency can detect malicious software updates. Data Access transparency can prevent covert violations of a published data protection policy, and allows public verification of access statistics.

When incorporated effectively in a responsive cycle, transparency logs can significantly enhance the speed of detecting and responding to any wrongdoing. This is because any changes or additions to the log can be immediately observed and scrutinized, thereby enabling swift action against potential security threats.

2.3 Merkle Hash Trees

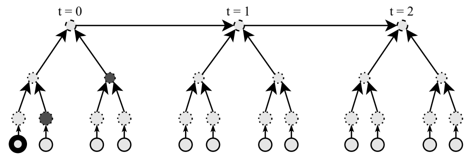

Figure 2: A time-based forest of merkle trees.

Transparency logs organize their entries in a cryptographically verifiable data structure, specifically in a Merkle Hash Tree [23]. A Merkle Hash Tree is a form of tamper-evident history tree [8] that stores data at the leaves of a binary tree. A forest consisting of three such trees is depicted in Figure 2. Each Merkle tree consists of leaf nodes (solid border), intermediate nodes (dotted border), and a root node (dashed border). Every non-leaf node is formed by concatenating the hash of its two child nodes, and hashing the result. Thus, each non-leaf node constitutes a commitment to the contents of its children, and transitively the children of its children. Consequently, each tree's root node is a commitment to the entire tree's contents.

The integrity of merkle threes is protected by two cryptographic proofs (as further described in Appendix B): An *inclusion proof* persuades external entities that a leaf is part of the log. A *consistency proof* proves that the log is append-only by demonstrating that the published root hashes are consistent.

Based on these proofs, the concept of *verifying a log* generally involves four steps: (i) confirming entry inclusion, (ii) ensuring consistency in log growth, (iii) validating uniformity across user views, and (iv) scrutinizing entries for potential malfeasance. The first three checks verify the correct conduct of the log operator, while the last check strives to guarantee the reliability of log entries and detect indications of malicious actions. *Ensuring the accuracy* of log entries is equally crucial as *recording* them to derive security benefits from transparency. Any system aiming to enhance security through transparency logs must incorporate one or more entities responsible for validating the correctness of log entries.

In the Certificate Transparency use case, Certificate Authorities (CAs) submit their newly issued certificates to the CT Log. Upon acceptance of a certificate, the log issues a signed timestamp that serves as future evidence of submission. TLS clients or browsers can subsequently mandate that all certificates must be accompanied by such a signed timestamp to be deemed valid. Domain owners can actively monitor the logs, regularly requesting all new entries to check for unexpectedly issued certificates for the domains under their responsibility.

It is worth noting that the fundamental infrastructure of applications where these transparency logs are used remains largely unchanged. Instead, this paradigm offers a mechanism to monitor and audit the activities to swiftly identify and exclude authorities engaging in improper practices.

2.4 Zero-Knowledge Proofs

Zero-Knowledge (zk) proofs are widely defined as cryptographic techniques that enable a *prover* to demonstrate their knowledge of specific information to a *verifier* without disclosing the actual information [15]. Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (zkSNARK) refers to short-proof constructions that do not require any back-and-forth interaction between the prover and the verifier.

3 Assumptions & Goals

3.1 Participating Actors

We now introduce our system's entities and their roles.

Credentials & Users. A *credential* is a data structure containing claims regarding a particular data subject. This data subject is commonly called the *user* or the *holder*. They operate a *user device* implementing a *wallet* application, which stores credentials on their behalf, and can present them to services.

Verifiers. The *(credential) verifier*, also known as the *service provider*, is a party requiring the user to prove or reveal certain attributes about themselves. The user achieves this by *presenting* a credential. There are many different service providers, which may offer wildly disparate services. An internet message board, a virtual storefront, a health provider's appointments software, a bank's online banking application, and a government bureau's web portal, are very different kinds of service providers, with different needs.

Many service providers share a need to (re-)identify users. If a user visits a message board, they should have the ability to edit messages they have previously sent; if they visit a storefront, they should be able to see their pending orders; if they visit their health provider, they should be able to see and cancel their appointments; and so forth. It is, therefore, commonly desirable that the credential presentation results in a *trusted persistent identifier* for the user.

Credential Issuers. The *credential issuer* is the party that issues credentials to users. They also store additional information that lets them authenticate users, such as password digests, shared secrets, or public keys.

Each service provider must decide whether it will trust a given credential. This decision almost always includes verifying that a trusted issuer issued the credential. In choosing this policy, the service provider determines to which issuers it will delegate the responsibility of actually verifying that the attributes contained in the credential accurately describe the user holding the credential.

We now outline actors not already involved in typical credential systems.

Log Provider. The *log provider* is an untrusted party that maintains a public append-only audit log containing chunks of information known as *log entries*. This role is primarily concerned with providing storage of information, and ensuring that this information remains publicly available; a responsibility that none of the traditional actors may wish to assume. The role is untrusted; it does not obtain any information that is not public, and can be cryptographically prevented from modifying or retracting information.

Auditor. *Auditors* ensure that the Log Provider strictly appends and maintains global consistency. In other words, Auditors are tasked with uncovering any misbehavior of the Log Provider. Any auditor can detect a cheating log provider, and can irrefutably prove that they cheated. Since any member of the public can operate an auditor, requiring no additional access or privileges, log providers must assume that the data they publish is monitored and audited at all times. This, combined with a robust gossip scheme of the observed log fingerprints, ensures they operate faithfully. Any interested party can act as an auditor; no particular access is required.

Monitor. *Monitors* routinely query the data published by the log provider and ensure internal consistency within individual trees. By verifying that the content of the log remains free from inconsistencies, monitors assist in identifying misconduct by the log provider. In doing so, monitors also process individual log entries and can detect mis-issuance.

Monitors may also act as mirrors of the log, and offer search and notification services.[1]

3.2 Mis-issuance

In this work, we address a particular type of misbehavior by a credential issuer: the issuance of *legitimate* credentials to *illegitimate* recipients. We are not con-

[1] See Section 6.2 for a discussion of the privacy implications of such delegation.

cerned with the issuer lying about someone's age; we are concerned about them attesting that someone is me even though they are not.

We thus limit ourselves to a particular class of credential, which we will call *identifying credential*. We define this as a credential which, when shown to a particular verifier, will result in that verifier learning some trusted, stable, persistent, unique identifier of that user towards that verifier. Note that these identifiers (or *pseudonyms*) are not necessarily global; the same user may have different pseudonyms towards different verifiers. However, multiple showings by the same user at the same verifier will result in the same pseudonym; this makes the pseudonym suitable for re-identification.

This is a common class of credential; it is used in both OpenID Connect [28] and SAML [25], the most widely used authentication protocols on today's internet. These stable pseudonyms are commonly derived from some private global user identifier (which we'll call userId) only known to the issuer. A naive example of this, commonly used in OpenID Connect's pairwise pseudonymous identifiers, is pseudonym derivation as $H(verifier||\mathsf{userId})$ for some cryptographic hash function H.

With these preliminaries in place, we now define *mis-issuance* in the context of this work as follows: an issuer surreptitiously issues a credential based on a victim's UID to some malicious collaborator, who is not the victim. The collaborator may be the issuer themselves. This credential allows the collaborator to impersonate the user towards verifiers, as presenting the credential will result in verifiers associating the victim's pseudonym with the collaborator.

Our work allows the victim to detect such missuance, allowing them to take appropriate reputation-based action against the issuer. This parallels the efforts of the certificate transparency system to allow website administrators to detect mis-issued TLS certificates. However, certificate transparency concepts do not trivially translate to the credential context; the content of credentials is private, and the user's global identifier is doubly so.

3.3 Threat Model

Users, issuers, auditors, and log providers may all act maliciously, attempting to subvert the protocol's intent. We identify the following challenges.

A malicious Log Provider might seek to present different views of the transparency log to different entities. This might serve to avoid disclosing a mis-issued credential to the user, while convincing a verifier that it was logged correctly.

A malicious or compromised issuer might cooperate with or incorporate an imposter acting as a user. It might try to issue a credential allowing this *malicious* user to impersonate another *genuine* user. This is the core challenge that our work mitigates; in the context of our work, the issuer might thus attempt to hide the issuance from the public log, such as by not submitting it, or not submitting it correctly.

A malicious auditor – a role that requires no particular authorization – might attempt to learn private information about the user. For example, it might try to enumerate a particular user's credentials.

Finally, collaborating verifiers might attempt to correlate multiple credentials issued to the same user; something that scoped pseudonyms schemes typically aim to prevent. In the scope of this work, we must not enable this via the logged information. The augmented version of any given credential scheme should be as unlinkable as the base version.

3.4 Goals

Certificate Transparency offers an inspirational example of such public audit logging being both practicable and practical. Yet, it cannot be translated directly to the credential context; there are multiple challenges, which we outline here.

Fundamentally, *a credential's contents are not public*. By contrast, in certificate transparency, certificates' contents are public. Thus, we cannot simply log the entire credential, as certificate transparency does.

The blinding mechanism we employ needs to be able to solve two challenges. First, *user discoverability*. Given an entry in the credential log, a user should be able to determine whether this entry corresponds to their userId. Second, *verifier verifiability*. Given a credential and a log entry, the verifier needs to be able to determine whether the log entry corresponds to this credential.

Additionally, the following privacy requirements need to be met. First, *credential privacy*. Given all public information, including the public log entry, it is infeasible to determine any claims contained within the credential.[2] Second, *credential unlinkability*. Given two credentials and their associated log entries, it should be infeasible to determine whether they correspond to the same userId.[3]

4 Credential Issuance Transparency

This section describes our Credential Issuance Transparency scheme. It is based on append-only logs, cryptographic blinding, and zero-knowledge proofs, and enables genuine users to detect credential mis-issuance while not compromising the privacy properties of the underlying credential scheme.

4.1 Building Blocks

We utilize some of the previously-described building blocks unchanged, and make some assumptions regarding the underlying credential scheme.

Append-Only Logs. We assume that an append-only, tamper-evident log, as described in Section 2.2, exists. It features efficient proofs of inclusion and consistency, allowing us to ensure the integrity of the recorded information in the log. We assume that auditors have the capability to identify log misbehavior, such as altered logs or split-view attacks. This may necessitate connecting the

[2] We assume that the entire credential contains sufficient entropy to prevent brute force attacks.

[3] We assume that the existing credentials, without logging, have this property.

public snapshots of the log to some form of transparency overlay [7]. We also assume that the log provider can somehow identify authorized credential issuers, preventing the log from being rendered unusable by garbage data.

Pseudonyms. We assume that users' stable pseudonyms can be deterministically computed with knowledge of the verifier's identity in addition to some secret global userId, which is only known to the user and issuer. In particular, we assume that this userId is known to the user. In a user-centric identity model, we find this assumption to be reasonable.

Zero-Knowledge Proof System. We assume that some trusted procedure has been used to globally initialize a non-interactive zero-knowledge proof system. This allows any user to prove zero-knowledge statements over disclosed or undisclosed inputs to any verifier. Such setup could employ multi-party computation, or some other scheme to ensure its compliant execution. Regardless of how this is done, it only needs to be done once, and the resulting parameters can be used globally.

Under these initial assumptions, we now introduce our protocol.

4.2 Protocol

To solve these challenges, we propose our *Credential Issuance Transparency* protocol for privacy-preserving credential issuance audit logging. Its log entries consist of the hash digest of the user's credential and a blinded commitment to the referenced userId. Knowing their userId, a user can then test whether a given entry corresponds to them by simple trial calculation. To satisfy verifiability, we supply verifiers with a non-interactive zero knowledge pre-image proof. Thus, they can verify that the blinded userId in the log entry equals the userId underlying their learned pseudonym, without learning that userId.

We now formalize this intuitive explanation.

Let $C(n, x)$ is a deterministic commitment function; n is a nonce, and x is the input. Given n and x, $C(n, x)$ is easy to obtain. At the same time, even given n and $C(n, x)$, x is hard to determine. Additionally, given n_1, $C(n_1, x_1)$, n_2, and $C(n_2, x_2)$, it is hard to decide whether $x_1 = x_2$.

Let $P(s, u)$ be the pseudonym derivation function; s is some service ID, and u is the user ID. Given s and u, $P(s, u)$ is easy to obtain. Yet, given s and $P(s, u)$, u is hard to determine. Additionally, given s, s', $P(s, u)$, and p', it is hard to determine whether $p' = P(s', u)$ or not.

Let H be a cryptographic hash function, with the standard properties.

We now augment the credential issuance process as stated in Protocol 1. Steps marked with + are newly introduced by our protocol. The user can then verify that $C(n, \mathsf{userId})$ and $H(x)$ are correct, and that the append proof is valid.

The showing process is augmented as stated in Protocol 2. Once again, steps marked with + are added by us. In this process, the verifier convinces itself that the credential has been correctly appended to the public log; due to its append-only nature, the credential cannot be removed once appended.

Finally, in Protocol 3 we introduce a discovery process by which the user can discover all credentials issued for its userId.

− The issuer creates credential x based on user ID userId.
+ The issuer samples a random nonce n, then sends a signed log entry $l := (n, C(n, \mathsf{userId}), H(x))$ to the log provider.
+ The log provider appends l to the append-only audit log, then returns a signed append-proof to the issuer.
− The issuer returns credential x, the log entry, and the append proof to the user.

Protocol 1: Issuance Process

− The verifier with verifier ID s prompts the user to prove their identity.
− In response, the user shows identifying credential x.
+ They also show the log entry $(n, C(n, \mathsf{userId}), H(x))$
  and its append proof.
− The verifier verifies the credential's provenance.
  It learns pseudonym $p := P(s, \mathsf{userId})$ as a result.
+ The user provides a NIZKP of the statement
  $c = C(n, \mathsf{userId}) \wedge p = P(s, \mathsf{userId})$. The user does not reveal userId.
+ If necessary, the verifier may query a consistency proof from the log provider to validate the inclusion proof.
+ The verifier checks that $H(x)$ in the log entry matches the provided credential.
+ The verifier checks the NIZKP, and compares the revealed inputs n, s, c and p against the expected values.
+ The verifier can now be confident that the issuance of x was correctly logged.

Protocol 2: Showing Process

+ The user requests all new audit log entries since the last time the process was run.
+ For each entry $(n, C(n, \mathsf{userId}'), H(x))$, the user trial calculates $C(n, \mathsf{userId})$ and compares this against $C(n, \mathsf{userId}')$.
+ If the values match, this entry corresponds to a credential that was issued for the user's identity
+ The user compares $H(x)$ against the values for its known pseudonyms. If no match is found, this credential was mis-issued.

Protocol 3: Discovery Process

4.3 Analysis

We now argue that our protocol solves the challenges outlined in Section 3.4, and thus also addresses the threats discussed in Section 3.3.

User discoverability. Assuming that the credential has been correctly appended to the log, the user will necessarily encounter its entry. Once the user encounters the entry, the trial calculation will succeed.

Verifier verifiability. In a scenario where the identity provider has mis-issued a credential, both it and the user the credential was issued to are not trustworthy. Therefore, it falls to the verifier to verify whether the credential has been correctly appended. In order to be *correctly appended*, three criteria need to be met. 1. The log entry needs to be contained in the log. 2. The hash digest in the log entry needs to match the credential. 3. The commitment in the log entry needs to match the userId used for the pseudonym.

After performing the verification steps in Section 4.2, the verifier can be assured of all three requirements. 1. It has performed the inclusion check as described in Section 2.2. Therefore, the log entry is contained in the log. 2. It has verified that the hash digest of the credential is included in the log entry. 3. It has verified the user-provided NIZKP. Thus, it is assured that the userId committed to in the log entry is the same userId used in the pseudonym derivation.

Credential privacy. The log entry makes $H(cred)$ publicly available. If $cred$ has sufficient entropy, it is infeasible to find $cred$ from this value.

Additionally, the log entry makes the pair $(n, C(n, \mathsf{userId}))$ publicly available. However, assuming that userId has sufficient entropy, it is infeasible to find userId from this pair. Additionally, we note that, if userId did not have sufficient entropy, the pseudonym derivation $P(s, \mathsf{userId})$ would not be privacy-preserving.

Credential unlinkability. Given that the credentials in question are already known, only the two pairs $(n_1, C(n_1, \mathsf{userId}_1))$ and $(n_2, C(n_2, \mathsf{userId}_2))$ are learned from the log entry. Due to our security assumptions regarding C, this does not allow an adversary to decide whether $\mathsf{userId}_1 = \mathsf{userId}_2$.

5 Evaluation

To demonstrate the feasibility and practicability of CIT we provide a proof-of-concept evaluation based on the Trillian framework.[4] Trillian is a generalization of the tamper-evident log employed by Certificate Transparency, and it accommodates logs of arbitrary data. We use SHA-256 hashes, and commit to userId using $C(n, \mathsf{userId}) := \mathsf{SHA}(n||\mathsf{userId})$. We similarly derive pseudonyms using $P(s, \mathsf{userId}) := \mathsf{SHA}(s||\mathsf{userId})$. In both cases, $||$ is concatenation.

Further, we evaluate the NIZKP proving the relation between the log entry l and the user's pseudonym p. For the implementation of the NIZKP we utilize the gnark zk-SNARK library[5] [4].

To ensure the practicability of CIT, two criteria must be met: (i) Logging of issued credentials should not introduce significant delays to the credential issuance and verification process, and (ii) the underlying log storage layer should not impede scalability for broader deployments.

5.1 ZKP Proof

While our CIT concept is described on a generic level, the concrete choice of ZKP proof system and implementation is important to assess the feasibility and to evaluate the performance and security. Thus, we provide a concrete instantiation of CIT's NIZKP. To do so, we implement the statement $c = C(n, \mathsf{userId}) \land p = P(s, \mathsf{userId})$ in the form of a gnark circuit [4]. Paraphrased, this states that the commitment c (in the log entry) and pseudonym

[4] https://github.com/google/trillian

[5] https://github.com/consensys/gnark

p (which the verifier will use) correspond to the same userId. This circuit is compiled into a Rank-1 Constraint Systems (R1CS) by gnark. As the proving scheme, we utilize Groth16 [16] on the BN254 elliptic curve. We choose Groth16 for its better performance over, e.g., PLONK [13]. Groth16 uses a circuit-specific trusted setup,[6] but this is not a downside as our system uses the same circuit for all entities in the system. BN254 provides 102 bits of security [17, Table 8]. We benchmark three different commitment functions, demonstrating their impact on the performance. Specifically, we apply the SHA-256, SHA3, MiMC hash functions [9,10,2]. While the SHA family is commonly used in practice, we additionally chose MiMC as representative of a family of ZKP-friendly hash functions. Implementing a ZKP proof on a ZKP-friendly function results in a smaller circuit and, thus, better performance.

5.2 Performance Results

We analyze the overhead of employing CIT in a real-world scenario and conclude that doing so would be practical.

Logging. We identify three key areas of concern when analyzing CIT's impact on performance.

First, a monitor – i.e., the user – must download all log entries that were appended since the last check. In Web PKI Certificate Transparency, a solution that is deployed in practice, Bingyu Li et al. report an average size of 5.93 KB for log entries, and an average growth rate of 7,778,870 records per day, resulting in 43.99 GB of new log content per day [21]. This makes it practically challenging for anyone except well-supported enterprises to run a CT monitor. In stark contrast, due to only storing blinded data, Certificate Issuance Transparency log entries are constant size – 96 *bytes* in our implementation.[7] Even assuming the monumental growth rate of CT, this results in only 746 MB of log content per day; certainly feasible for users to keep up with on a home internet connection.

Second, the user must go through all downloaded entries and trial calculate to test the userId commitment. In our implementation using SHA-256, hardware acceleration allows for a hash rate of upwards of 3 million hashes per second on consumer hardware [11]. Even discounting various operating system overhead, performing such verification on the 8 million records per day reported by Bingyu Li et al. would clearly be feasible for home users.

Third, the entry must be submitted to the log, and an append proof must be obtained. Both involve negligible transfers of data. Merkle Tree append proof size is logarithmic in the number of elements in the tree; assuming SHA-256 hashes, day-long epochs, and the CT append rate reported by Bingyu Li et al., append proofs would be less than 1 KB each.

We conclude that, even at the scale reached by a potential worldwide deployment, Certificate Issuance Transparency would still be feasible to monitor for end

[6] i.e., the trusted setup of verifier- and prover-key must be redone if the circuit changes
[7] 32 bytes SHA-256 userId commitment, 32 bytes random blind, 32 bytes SHA-256 credential hash

users on home hardware. Additionally, we conclude that the logging operations imposed by CIT do not impose noticeable overhead.

Proofs of Logging Correctness. To evaluate our ZKP implementation, we perform several benchmarks on an office laptop with an Intel i7-8550U CPU. The results of this evaluation are given in Table 1 and visualized in Figure 3. We divide the steps into three phases: i) the initiation phase is only performed once for the whole system and comprises the (trusted) setup and the compilation of the circuits, ii) the preparation phase is performed once for each nonce-userID-spID tuple and can be pre-computed by the user, and iii) the showing phase is performed by the verifier on each credential showing. We note that the showing phase is fast enough that caching at the verifier is not useful, but possible.

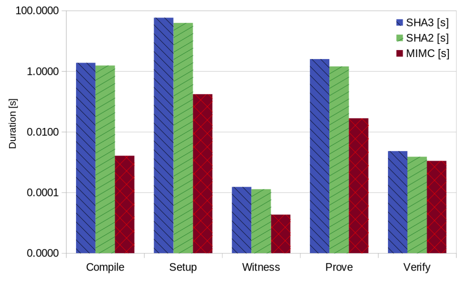

Figure 3: NIZKP Evaluation

		BN254		
		MiMC	SHA256	SHA3-256
	security		102 bit	
	#constraints	1322	170157	235032
initiation	compile policy [s]	0.002	1.541	1.891
	setup (gen. keys) [s]	0.176	39.134	58.189
prepare	gen. witness [s]	0.00002	0.0001	0.0002
	prove statement [s]	0.028	1.444	2.515
show	verify proof [s]	0.001	0.002	0.002

Table 1: Evaluation Results

6 Discussion

In a given application scenario, it might be desirable to tweak the functionality of our protocol. We discuss potential avenues for doing so; allowing the trading of privacy for performance alongside various axes.

6.1 Identifying Credentials

Our work is limited to auditing the issuance of *identifying credentials*, which produce a stable pseudonym for the user at a particular verifier. We have done this because for this class of credential, the intuitive concept of "mis-issuance" can be defined rather plainly, by tying it to the pseudonym in question.

Of course, not all credentials are identifying credentials. I might be issued a credential that only includes my date of birth. We might also want to audit mis-issuance of such credentials. However, I almost certainly share a date of birth with some other legitimate credential subject. If a date-of-birth credential for my date of birth is issued to them, this is not mis-issuance. Yet, from my point of view, the two cases are indistinguishable; it is not even clear how to define "mis-issuance" in this context.

Finally, we turn to a third case: identifying credentials being used in contexts where the identifying information – the pseudonym – is not used by the verifier. For example, this may be achieved by not selectively disclosing the pseudonym.

At first, a path forward seems apparent in this scenario: perform issuance audit logging as usual, but instead of disclosing the pseudonym, extend the NInowZKP in Section 4.2 to prove consistency with the undisclosed pseudonym contained in the credential. While the details might vary based on the pseudonym derivation method used, this seems intuitively possible. Yet, this hope, too, is fleeting. After all, consider a scenario where the issuer, cooperating with some malicious user, issues a modified credential to that user containing otherwise identical claims, but a bogus pseudonym derived from a non-existent userId. The issuer then faithfully records this credential in the audit log. Since the only information the verifier makes use of in this scenario is non-identifying information – not the pseudonym – this credential is *functionally equivalent* to a genuine credential. Clearly, the modified credential was also *mis-issued*; yet, since the identifying attributes are not used by the verifier, this mis-issuance is functionally equivalent yet hard to capture in a rigorous definition.

Other uniquely identifying information. In this work, we have focused on "a pseudonym" as the unique identifier of a user towards a given verifier. One might reasonably argue that no specific pseudonym is necessary to define "mis-issuance"; after all, if someone else is issued a credential containing my first name, last name, and date of birth, isn't this also clearly mis-issuance? It is not clear that this is the case; *identity* is a hard concept to pin down. "Data twins", people sharing both the same name and the same date of birth, are not unheard of. However, if one is convinced that they have identified such a uniquely identifying combination of attributes, the concepts of this work can be extended seamlessly by substituting the combination of these attributes for a pseudonym.

6.2 Delegating Log Monitoring

Our base scheme, described in Section 4.2, requires interested users to monitor all new additions to the audit log, and trial calculate to test whether the addition matches their own userId. This may not be desirable for some users. While the effort expended to perform this monitoring is not excessive, it may nevertheless be desirable for users to delegate this duty to professional *monitors*. Such a monitor could process the log just once, and notify each user only of the log entries that concern them.

This delegation can be supported by a slight adjustment to our protocol, and at a trade-off in NIZKP performance. Our current protocol publicly logs a commitment to userId, in the form of $C(n, \mathsf{userId})$. Knowing userId, the user can test each log entry. However, delegating monitoring to a third party would then also require userId to be disclosed to the third party. This allows the user to be de-anonymized at SPs globally. Even if the monitor is trusted to perform monitoring, disclosing such sensitive data is not desirable.

Thus, if delegation should be supported, instead of logging a commitment to userId, a commitment to $H(\mathsf{userId})$[8], i.e., $C(n, H(\mathsf{userId}))$, can be logged instead. This allows a third-party monitor to operate only with knowledge of $H(\mathsf{userId})$.

[8] or, in practice, $H(”LOG\_KEY”||\mathsf{userId})$ to provide domain separation

This knowledge permits the monitor to determine whether a given log entry corresponds to the underlying user, as desired; but it does not allow it to perform any other operations on the userId, nor does it allow it to learn the userId. Of course, the NIZKP provided to the verifier also needs to incorporate this additional operation. This corresponds to a 50% increase in system setup and proof generation time compared to our existing benchmarks in Section 5.

6.3 Anonymity versus Monitoring Efficiency

Another venue for improving monitoring efficiency is trading off some of the user's anonymity for a corresponding decrease in entries to be monitored. For n-bit userIds, this is achieved by choosing a parameter p from 0 to n, and treating the first p bits of the user's userId as public information. The audit log can then effectively be fragmented into 2^p different audit logs. Each user only needs to monitor one of these logs, which contains 2^{-p} times as many entries as the unparameterized log. The correct assignment of the entry to a particular log can be verified as part of the existing NIZKP by simply revealing the first p bits of the userId input. This does not induce significant overhead. However, such a parametrization comes with a commensurate reduction in anonymity, as the user base is split into 2^p disjoint anonymity sets.

6.4 Related Work

We examine research associated with transparency overlays and digital (verifiable) credentials regarding the underlying data structures and sanitization mechanisms employed.

In *Certree*, Saramago et al. suggest a system that adds transparency to certification systems of academic credentials [29]. Their approach enhances transparency in the issuance process by leveraging metadata about each individual certification stage, such as a course completed towards a degree, recorded in a blockchain. However, at the same time, their approach requires all credentials to be made public. This allows tracking all users' credentials and related activities through on-chain data, as user addresses (hash of their public key) are included in the metadata for each credential.

Zk-Cert maintains the transparency of certifying academic credentials systems provided by Certree while significantly enhancing privacy by implementing zero-knowledge proofs [30]. Their approach employs a blockchain as a verifiable data registry with robust timestamping capabilities. A group of smart contracts systematically records cryptographic commitments of all issued credentials in an incremental Merkle Tree, and zero-knowledge proofs enable subjects to demonstrate ownership over these commitments without disclosing them.

Chase et al. propose and formalize their *credential transparency framework (CTS)* for remote (possibly cloud-based) credential management systems (CMS) [6]. CTS adds transparency guarantees by logging every time a credential management service provider *presents* a credential on behalf of a user to an honest verifer, in such a way that the user can then audit all the presentations made on their behalf, without the remote CMS being able to manipulate or omit any presentation without detection. This cloud-based CMS maintains a Strong Accumulator (SA) data structure that stores commitments of credential presentations' contexts and a zero-knowledge set (ZKS) of the number of uses of the user's credential. The framework is compatible with credential management systems of varying degrees of privacy-preservation. By taking this approach and abstracting each credential presentation as a "showing", they ensure that incorporating logging guarantees does not introduce substantial additional privacy leakage beyond what the credential presentation already discloses.

Goldwasser et al. address the lack of accountability of anonymous credentials that restricts their practical applications in real-world scenarios [14]. They present an *anonymous verifiable logging scheme* that allows decentralized inspectors to enforce accountability for anonymous credential users. This scheme guarantees that the user's activities are recorded and encrypted using a key that is unique to the user. A third-party auditor that possesses the user's log decryption key, can subsequently access and assess the user's actions to ascertain compliance to regulations. They adapt the issuing protocol by including a log encryption key chosen by the user and the auditor in the credential. They associate the log entries of anonymous credential presentations with the credential user identified during issuance so that a comprehensive log of all the user's activities at honest verifiers can be retrieved. Importantly, this retrieval doesn't compromise the privacy of other users.

Zk-creds, besides employing general-purpose zero-knowledge proofs as a basis for anonymous credentials, proposes to eliminate the requirement for credential issuers to secure signing keys by issuing credentials to a public bulletin board [27]. By doing so, every issued credential is observable and the issuance process is auditable. A user can use her credential by showing a membership proof that the credential is present in the issuance list. Membership proofs are realized through Merkle Forests rather than a single Merkle Tree for an improved tradeoff between proving and verification time.

6.5 Future Work: Multi-show Unlinkability

Our approach described in this work has one log line correspond to one issued credential. The user then provides this log line, and a traditional inclusion proof, to verifiers when showing the credential. This prevents *multi-show unlinkability*, where multiple instances of showing the same credential cannot be correlated. Current eIDAS 2.0 implementations do not provide this property. However, more advanced cryptographic schemes such as BBS+ do [3].

It may be possible to adopt our approach to work in such schemes without compromising multi-show unlinkability. We believe this might be achieved by a two-step process: first, by replacing the credential hash $H(x)$ with a blinded commitment; and second, by showing a NIZKP that merely proves that a correct log line exists in a given tree instead of disclosing the particular line. We do not pursue this idea further in this work.

References

1. Al-Bassam, M., Meiklejohn, S.: Contour: A practical system for binary transparency. CoRR **abs/1712.08427** (2017)
2. Albrecht, M.R., Grassi, L., Rechberger, C., Roy, A., Tiessen, T.: Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In: ASIACRYPT (1). LNCS, vol. 10031 (2016)
3. Au, M.H., Susilo, W., Mu, Y.: Constant-size dynamic k-taa. In: SCN. LNCS, vol. 4116. Springer (2006)
4. Botrel, G., Piellard, T., Housni, Y.E., Kubjas, I., Tabaie, A.: Consensys/gnark: v0.9.0 (feb 2023), https://doi.org/10.5281/zenodo.5819104
5. Cameron, K.: The laws of identity (2005)
6. Chase, M., Fuchsbauer, G., Ghosh, E., Plouviez, A.: Credential transparency system. In: International Conference on Security and Cryptography for Networks (2022)
7. Chase, M., Meiklejohn, S.: Transparency overlays and applications. In: Proceedings of the 2016 acm sigsac conference on computer and communications security (2016)
8. Crosby, S.A., Wallach, D.S.: Efficient data structures for tamper-evident logging. In: USENIX security symposium (2009)
9. Dang, Q.: Secure hash standard (shs) (2012-03-06 2012)
10. Dworkin, M.: Sha-3 standard: Permutation-based hash and extendable-output functions (2015-08-04 2015)
11. ECRYPT VAMPIRE: Measuremants of hash functions, index by machine. https://bench.cr.yp.to/results-hash.html (2024)
12. European Parliament, Council of the European Union: Regulation (EU) 2024/1183 of the European Parliament and of the Council of 11 April 2024 amending Regulation (EU) No 910/2014 as regards establishing the European Digital Identity Framework (eIDAS 2) (2024)
13. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. IACR Cryptol. ePrint Arch. (2019)
14. Godtschalk, L.: Accountability and access control using anonymous credentials (2022)
15. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems (2019)
16. Groth, J.: On the size of pairing-based non-interactive arguments. In: EUROCRYPT (2). LNCS, vol. 9666. Springer (2016)
17. Guillevic, A., Singh, S.: On the alpha value of polynomials in the tower number field sieve algorithm. IACR Cryptol. ePrint Arch. (2019)
18. Hicks, A., Mavroudis, V., Al-Bassam, M., Meiklejohn, S., Murdoch, S.J.: VAMS: verifiable auditing of access to confidential data. CoRR **abs/1805.04772** (2018)
19. Hof, B., Carle, G.: Software distribution transparency and auditability (2017)
20. Laurie, B., et al.: Certificate transparency v2.0. RFC **9162** (2021)
21. Li, B., Lin, J., Li, F., Wang, Q., Li, Q., Jing, J., Wang, C.: Certificate transparency in the wild: Exploring the reliability of monitors. In: CCS. ACM (2019)
22. Marquardt, F., Schmidt, C.: Don't stop at the top: Using certificate transparency logs to extend domain lists for web security studies. In: LCN. IEEE (2020)
23. Merkle, R.C.: A digital signature based on a conventional encryption function. In: CRYPTO. LNCS, vol. 293. Springer (1987)

24. Nikitin, K., Kokoris-Kogias, E., Jovanovic, P., Gailly, N., Gasser, L., Khoffi, I., Cappos, J., Ford, B.: CHAINIAC: proactive software-update transparency via collectively signed skipchains and verified builds. USENIX Association (2017)
25. OASIS Security Services TC: Security assertion markup language (saml) v2.0 technical overview. https://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0-cd-02.html (2008)
26. Roberts, R., Levin, D.: When certificate transparency is too transparent: Analyzing information leakage in HTTPS domain names. In: WPES@CCS. ACM (2019)
27. Rosenberg, M., White, J., Garman, C., Miers, I.: zk-creds: Flexible anonymous credentials from zksnarks and existing identity infrastructure. In: IEEE Symposium on Security and Privacy (2023)
28. Sakimura, N., Bradley, J., Jones, M.B., de Medeiros, B., Mortimore, C.: Openid connect core 1.0. https://openid.net/specs/openid-connect-core-1_0.html (2014)
29. Saramago, R.Q., Jehl, L., Meling, H., Estrada-Galiñanes, V.: A tree-based construction for verifiable diplomas with issuer transparency. In: IEEE International Conference on DAPPS (2021)
30. Saramago, R.Q., Meling, H., Jehl, L.N.: A privacy-preserving and transparent certification system for digital credentials. In: International Conference on Principles of Distributed Systems (OPODIS) (2023)
31. Scheitle, Q., Gasser, O., Nolte, T., Amann, J., Brent, L., Carle, G., Holz, R., Schmidt, T.C., Wählisch, M.: The rise of certificate transparency and its implications on the internet ecosystem (2018)
32. Sporny, M., Longley, D., Chadwick, D.: Verifiable credentials data model v1.1. https://www.w3.org/TR/vc-data-model/ (2022)
33. Zwattendorfer, B., Zefferer, T., Stranacher, K.: An overview of cloud identity management-models. In: WEBIST (1). SciTePress (2014)

A Verifiable Presentations

A Verifiable Credential (VC) showing process begins with a verifier sending a credential presentation request to the user. This request generally includes a nonce to prevent replay attacks and asks for the disclosure of certain attributes from one or several of the user's credentials. It also includes additional parameters and constraints, such as a timestamp for expiration and revocation-related requirements, or a list of trusted issuers for each identity attribute.

Verifiable Presentation (VP) refers to a process involving individuals, or "users", who use their credentials to disclose identity attributes to "verifiers" or relying parties. Upon receiving the presentation request, the user's digital wallet application automatically searches for stored credentials that include the requested attributes and meet the requirements specified in the request. With the user's consent, the wallet app creates a cryptographic proof of the correctness of these attributes and sends them, along with the proof, to the verifier.

To avoid showing excessive information, VPs with a privacy emphasis selectively disclose specific attributes to the verifier. VPs also contain some form of blinded cryptographic evidence originating from the credential, indicating that the displayed subset of attributes is genuinely derived from a credential endorsed by the corresponding issuer. These proofs may be built using a variety of technologies, such as Merkle hash trees (Section 2.3) or zero-knowledge proofs (see also Section 2.4). The verifier can then automatically check the proof, and verify that the credential was issued by an issuer that it trusts. It thus trusts the authenticity of the identity attributes claimed by the user, allowing them to be used to provide services to them.

B Merkle Hash Tree Security

It is possible to link multiple hash trees trees together to form an incrementally growing forest. Each append batch consists of a single Merkle Tree. The root hash of the previous Merkle tree forms part of the input to the new tree's root hash. Thus, the new root hash also doubles as a commitment to the previous root hash; and, transitively, as a commitment to the previous tree's contents, as well as any other previous trees' root hashes and contents. This gives us two categories of cryptographic proofs: *inclusion proofs* and *consistency proofs*.

An *inclusion proof* persuades external entities that a leaf is part of the log. It consists of a path within the Merkle Tree, extending from the leaf containing the relevant data to the root hash embedded in the published fingerprint. A *Merkle audit path* is computed for an inclusion proof and consists of the set of missing nodes that are essential for calculating the tree's root. If the computed root from the audit path aligns with the actual root, it serves as evidence that the leaf in question is present in the tree. In Figure 2, the inclusion proof for the bolded node is depicted with a dark background.

On the other hand, a *consistency proof* proves that the log is append-only by demonstrating that the published root hashes are consistent. Consistency proofs enable external entities to confirm that the log strictly appends, ensuring that the log reflected by a fingerprint at an earlier point in time is a precursor of the log indicated by a fingerprint at a subsequent time. A consistency proof contains a subset of intermediary nodes in the Merkle Tree that are essential for linking the two root hashes. Entities within the system share the observed log snapshots using so-called "gossiping" mechanisms that ensure that all parties have the same view of the log at the point of time of the snapshot.