AffixIO Technical Paper · WP-014
June 2026
affix-io.com

AFFIXIO WHITE PAPER · WP-014

Double-Spend Prevention for Zero-Knowledge Proofs: Proof Consumption Ledger and Session-Nonce Binding

One eligibility proof should not work for ten people.

AffixIO | United Kingdom | affix-io.com | June 2026

ABSTRACT

Zero-knowledge credentials fail operationally when the same proof gets shared or replayed. AffixIO tracks proof consumption digests and binds proofs to session nonces so each eligibility check fires once, without storing identity attributes.

CONTENTS

SECTION 1

Introduction

The double-spend problem was first formalised in the context of digital cash systems by David Chaum in the 1980s. A digital token that represents a monetary value has a fundamental problem: it is a string of bits that can be copied. If the same token can be presented multiple times to different recipients, the token's value is undermined. Chaum's blind signature schemes and their successors addressed this problem in digital cash through various mechanisms, ultimately leading to the blockchain-based approach in Bitcoin where the shared ledger prevents any token from being spent twice.

Zero-knowledge proofs face an analogous problem in access control and eligibility contexts. A ZK proof that an individual satisfies a condition (is over 18, has passed a KYC check, has received a governed AI response) is a valid cryptographic object. The cryptographic validity of the proof does not depend on who presents it. If a user who passed age verification shares their ZK eligibility proof with a friend, the friend can present a cryptographically valid proof of age verification without having undergone age verification. The proof is genuine; the claim that the presenting party was verified is fraudulent.

AffixIO's double-spend prevention combines two mechanisms. Session-nonce binding ensures that a proof is generated specifically for the session requesting it, binding the proof to the service's context rather than to the individual's credentials. The proof consumption ledger records every proof that has been accepted, ensuring that any proof can be presented at most

once regardless of session binding. The combination provides comprehensive replay prevention without storing any information about the individual whose credentials were proved.

SECTION 2

The ZK Proof Replay Problem

In a ZK proof system without replay prevention, the same proof bytes can be presented to a verifier multiple times. The verifier checks the proof against the verification key and the public output (for example, age threshold satisfied = 1). Both checks pass on every presentation: the proof bytes are unchanged, the verification key is unchanged, and the public output is unchanged. The verifier has no cryptographic means to distinguish the first presentation of a proof from the tenth.

This is not a bug in the ZK proof system. It is a feature that is valuable in some contexts: a ZK proof of knowledge of a secret (for example, knowledge of a hash preimage) should be presentable multiple times to the same or different verifiers. The proof is a credential asserting that its generator knew the preimage at the time of generation. Presenting the credential multiple times does not increase the number of valid preimages.

In access control contexts, however, replay is a vulnerability. An age verification credential that can be presented multiple times to access an age-restricted service could be shared across users, allowing multiple users to gain access on the basis of a single verification event. The ZK property that makes the proof valuable in a privacy context (it reveals nothing about the verifier) also makes it easy to share: sharing a credential that reveals nothing about you carries no privacy risk for the original verifier.

SECTION 3

Attack Scenarios

Three distinct attack scenarios motivate the double-spend prevention mechanisms.

Credential sharing. User A completes age verification and receives a ZK eligibility proof. User A shares the proof bytes with User B. User B presents the proof to the service and gains access without completing age verification. This is the digital equivalent of lending an ID card. The cryptographic validity of the proof is unaffected by who presents it.

Session replay. User A completes age verification for Session 1. User A saves the proof bytes. Session 1 expires. User A starts Session 2 and presents the saved proof from Session 1, bypassing the age verification requirement for Session 2. If the service does not check whether a proof was generated for the current session, this replay succeeds.

Multi-account use. User A completes age verification under Account 1. User A creates Account 2 and presents the proof from Account 1 to satisfy the age verification requirement for Account 2. If the service does not require proof generation to be associated with the specific account requesting access, the same proof works for multiple accounts.

Session-nonce binding addresses session replay and partially addresses multi-account use. The proof consumption ledger addresses all three scenarios by ensuring any proof can be accepted at most once.

SECTION 4

Session-Nonce Binding

Session-nonce binding embeds a service-generated, single-use random value (the nonce) in the proof generation request. The nonce is included as a witness to the circuit or as an additional input to the proof digest computation, binding the resulting proof to the specific session that generated the nonce.

The binding is implemented at the proof digest level rather than the circuit level, to avoid increasing the circuit's constraint count. The proof digest is computed as `a composite digest binding circuit identity, outcome, proof material, and session context`, where a session-bound nonce is mixed into the digest generated by the service at session initialisation. When the service later verifies a presented proof digest, it checks that the nonce embedded in the digest matches the nonce it generated for that session. A proof digest with a different or missing nonce is rejected.

Digest binding implementation omitted from public documentation.

The session nonce is generated by the service and sent to the proof generation service as part of the proof request. The proof generation service includes the nonce in the digest computation. The nonce is stored in the session context on the service side. When the proof digest is presented by the client, the service retrieves the session nonce and verifies that the digest was computed with that nonce. A proof generated with a different nonce produces a different digest and fails the check.

> **Nonce generation:** Session nonces are generated using a cryptographically secure random number generator. Each nonce is used for exactly one proof generation request and discarded after the proof digest is verified. Nonces are not stored after session expiry.

SECTION 5

The Proof Consumption Ledger

Session-nonce binding prevents replay within the same service (a nonce can only be used once per session) but does not prevent credential sharing if the same proof is never presented to a second session. The proof consumption ledger provides the second layer of defence: it records every proof digest that has been accepted, and rejects any proof digest that has been accepted before.

The registry is a write-optimised key-value store. The key is the proof digest (32 bytes). The value is the timestamp of first acceptance. When a proof digest is presented, the registry is checked for the digest. If the digest is present, the proof is rejected as already spent. If the digest is absent, the proof is accepted and the digest is immediately written to the registry.

The write operation must be atomic with the acceptance decision: a race condition in which two concurrent presentations of the same proof both check the registry before either has written to it would allow double-spend. AffixIO uses a conditional-write operation (write-if-absent) in the registry implementation to provide atomicity without explicit locking.

```
# Atomic write-if-absent using Redis NX (Not eXists) flag
redis.set(proof_digest, timestamp_iso, nx=True)
# Returns True if key was written (first use)
# Returns None if key already existed (replay detected)
```

The registry is append-only in the logical sense: digests are written once and never deleted. This allows the registry to grow continuously with the volume of governed events. For a system processing 1,000 governed events per day, the registry grows by approximately 32 KB per day (32 bytes per digest at 1,000 digests per day). At this rate, a year's registry is approximately 11.5 MB, easily manageable.

SECTION 6

Combined Defence

Session-nonce binding and the proof consumption ledger address different attack vectors. Session-nonce binding prevents replay within the service context (the nonce ensures the proof was generated for this service and this session). The proof consumption ledger prevents replay regardless of context (the digest ensures no proof has been accepted before anywhere in the system).

The combined defence handles all three attack scenarios described in Section 3. For credential sharing: User B receives a proof from User A, but the proof digest includes User A's session nonce, which does not match User B's session nonce, and additionally the digest is already in the spent registry from User A's session. Either check independently rejects the replay. For session replay: the spent registry check rejects the digest from a previous session. For multi-account use: the nonce check rejects a proof generated for a different account's session (different nonce), and the registry check additionally rejects any digest that has been used before.

ATTACK	NONCE BINDING PREVENTS?	REGISTRY PREVENTS?	COMBINED
Credential sharing	Yes (different session context)	Yes (digest already spent)	Prevented by both
Session replay	Partial (same nonce may be reused if session persists)	Yes (digest already spent)	Prevented by registry
Multi-account use	Yes (different session nonce per account)	Yes (digest already spent)	Prevented by both

SECTION 7

Performance Implications

The proof consumption ledger lookup adds approximately 1-2 ms to the proof acceptance latency. The write operation on acceptance adds approximately 1-2 ms. For systems with a Redis instance in the same network region as the governance service, the round-trip latency is under 1 ms per operation. The total overhead from both the lookup and the write is approximately 2-4 ms, negligible within the overall governance pipeline latency budget.

Registry throughput is bounded by the Redis instance's write throughput, which is typically hundreds of thousands of operations per second. For most governance deployments, registry throughput is not the binding constraint. At very high throughput (millions of governed events per day), a Redis cluster with horizontal sharding provides adequate throughput.

The session-nonce binding check is a local operation (comparing the nonce in the digest to the nonce in the session context) with essentially zero latency. It adds no network round-trips to the acceptance process.

SECTION 8

Privacy Properties

The double-spend prevention mechanisms do not compromise the data minimisation properties of the broader governance architecture. The proof consumption ledger stores only 32-byte SHA-256 hashes. It does not store any information about the individual whose credentials were proved, the content of the verified attributes, or the circuit witnesses. The hash of a proof is not reversible to the proof inputs; knowing the proof digest reveals nothing about the individual.

The session nonce is a random value with no connection to the individual's identity. The nonce generated for User A's session is computationally indistinguishable from the nonce generated for User B's session. Knowing that a given nonce was used in a given proof digest does not identify the session owner.

The registry's append-only growth means it retains all proof consumption digests indefinitely. This is required for correctness: a spent proof could theoretically be replayed years after its initial use if the registry deleted old entries. The indefinite retention of proof digests (32-byte hashes with no personal data content) does not create GDPR retention obligations, because the retained data is not personal data as defined in GDPR Article 4.

SECTION 9

Failure Modes

The double-spend prevention system has two significant failure modes.

Registry unavailability. If the proof consumption ledger is unavailable when a proof is presented, the service must choose between rejecting all proof presentations (high availability impact) and accepting presentations without registry check (security impact). AffixIO's default policy is to reject all presentations when the registry is unavailable: the security impact of accepting a replayed proof outweighs the availability impact of rejecting legitimate presentations. Services with strict availability requirements may configure a fallback to accept presentations with a warning flag, subject to later replay detection when the registry becomes available again.

Registry corruption or loss. If the registry data is corrupted or lost, previously-proof consumption digests are no longer recorded, enabling replay of all proofs generated before the corruption event. The registry must be backed up with at least daily frequency. After a registry data loss event, all proof digests generated before the loss event should be treated as potentially replayable and, if possible, the affected proofs should be invalidated and new proofs requested.

SECTION 10

Application to Age Verification

In the privacy-preserving age verification context (WP-009), the session-nonce binding prevents a verified user from sharing their age verification proof with unverified users. The online service generates a unique nonce at the start of the age verification flow for each session. The nonce is sent to the AffixIO ZK service as part of the proof generation request. The resulting proof digest includes the session nonce. When the user presents the proof digest to complete the age verification flow, the service verifies the nonce binding and the registry check. A proof generated for a different session (User A's nonce in User B's session context) fails the nonce check. A proof already presented in a previous session fails the registry check.

The combination effectively prevents all three attack scenarios for age verification. A minor attempting to gain access by borrowing a verified user's proof is blocked by both mechanisms. A user attempting to reuse their verification across multiple accounts is blocked by the registry check (same

proof, different account context). The Online Safety Act's effectiveness requirement for age assurance is better satisfied by a system that prevents proof reuse than by one that merely prevents credential falsification.

SECTION 11

Application to AI Governance

In the AI governance context, the double-spend problem takes a different form. AI governance proofs are generated for specific AI responses and are not intended to be transferable between responses. The governance record for Response A should not be presentable as a governance record for Response B. Session-nonce binding addresses this: each AI response generates a unique proof digest that includes the session nonce for that response. Attempting to present Response A's proof as governance evidence for Response B produces a proof digest that does not correspond to any response in the Merkle tree for the current session.

The more important application in AI governance is completeness assurance: every AI response must have exactly one governance proof, and that proof must not be shared across responses or sessions. The Merkle tree's ordered structure provides completeness assurance at the audit trail level (the tree contains exactly one leaf per governed response). The proof consumption ledger provides session-level assurance that no proof has been accepted more than once. Together they provide a governance infrastructure where every response is governed exactly once.

SECTION 12

Conclusion

Double-spend prevention for ZK proofs is a solved problem once its existence is recognised. The proof consumption ledger and session-nonce binding together provide comprehensive replay prevention for ZK eligibility proofs in

access control and governance contexts, without storing any personal data and with negligible performance overhead.

The mechanisms are general: they apply to any ZK proof-based access control or eligibility system, not only AffixIO's specific circuits. Any system that generates ZK proofs as access credentials and needs to prevent those credentials from being shared or replayed can adopt the proof consumption ledger pattern with minimal implementation effort. The registry is a standard key-value store with a write-if-absent operation; no specialist cryptographic infrastructure is required beyond what is already needed for the ZK proof generation and verification.

The combination of session-nonce binding and the proof consumption ledger closes the double-spend vulnerability in ZK-based access control, enabling ZK proofs to serve as reliable, non-transferable, single-use credentials in regulated access control and eligibility contexts. This is a necessary property for ZK-based age verification, KYC, and AI governance to function correctly in production.

Related reading

- WP-006: PII-Free KYC by Design with Zero-Knowledge Identity Circuits
- WP-017: ZK Selective Disclosure for eIDAS 2.0 and the EUDI Wallet
- WP-009: Privacy-Preserving Age Verification with Zero-Knowledge Proofs

Frequently asked questions

What is a ZK double-spend?

Reusing or sharing a valid proof to claim the same entitlement multiple times or across different users.

Does the spent registry store PII?

No. Only cryptographic digests of consumed proofs are retained.

How does session-nonce binding work?

Each verification session receives a fresh nonce embedded in the proof witness so proofs from old sessions fail validation.
