ZK-NR: A Layered Cryptographic Architecture for Explainable Non-Repudiation

MINKA MI NGUIDJOI Thierry Emmanuel[a,*], MANI ONANA Flavien Serge[a], DJOTIO NDIÉ Thomas[a]

[a]*Laboratory for Mathematical Engineering and Information Systems (LIMSI)*
National Advanced School of Engineering
University of Yaoundé I, Cameroon

Abstract

This paper introduces ZK-NR, a modular cryptographic protocol designed to ensure privacy-preserving non-repudiation in the co-production of digital public services. By integrating Merkle commitments, zero-knowledge proofs (STARKs), threshold BLS signatures, and post-quantum Dilithium authentication, ZK-NR enables the creation of secure, verifiable, and auditable evidence across decentralized infrastructures. Unlike traditional digital signatures or blockchain-based logs, ZK-NR provides formally verifiable attestations without disclosing sensitive content, making it suitable for public finance, e-government, and regulated digital ecosystems. The protocol is modeled in Tamarin and implemented as a proof-of-concept using open cryptographic tools. This contribution offers a reproducible foundation for future infrastructures requiring long-term trust, data minimization, and legal admissibility, particularly in contexts where citizens and institutions share responsibility for digital evidence. ZK-NR addresses the tension between confidentiality and accountability, providing an interoperable and future-ready layer for trustworthy public service delivery. This preliminary work focuses

---

[*]Corresponding author

Email addresses: minkathierry@gmail.com (MINKA MI NGUIDJOI Thierry Emmanuel), serge.mani@digitworldacademy.com (MANI ONANA Flavien Serge), tdjotio@gmail.com (DJOTIO NDIÉ Thomas)

This manuscript corresponds to the version accepted for presentation at ICTO 2025 but not included in its proceedings. It constitutes the initial implementation-oriented version of the ZK-NR architecture. Formal UC proofs, design rationale, and symbolic security analysis are developed separately in unpublished manuscripts A.0–A.4.

on architectural composition and implementation feasibility. It does not include formal security proofs.

Keywords: Zero-Knowledge, Non-Repudiation, Post-Quantum Crypto, Merkle, STARKs, Signatures, Tamarin

1. Introduction

Distributed infrastructures increasingly require cryptographic mechanisms that ensure verifiability and accountability without compromising confidentiality. In sectors such as audit, compliance, and forensic computing, this balance is critical. Traditional digital signatures offer traceability, but expose data and remain vulnerable to post-quantum threats. Conversely, zero-knowledge proofs protect privacy but often lack composability, interpretability, or long-term resilience.

ZK-NR is a developmental cryptographic protocol designed to address this trade-off. It combines four layers, Merkle-based commitments, STARK proofs, threshold BLS signatures, and post-quantum Dilithium authentication, into a modular and formally verifiable construct. The goal is to support privacy-preserving non-repudiation under adversarial conditions, while enabling symbolic and practical validation.

This paper presents the architecture, formal model, and partial implementation of ZK-NR. It focuses on reproducibility and modular integration rather than completeness. By formalizing its logic, testing implementation feasibility, and identifying open issues, this work aims to contribute a reusable foundation for secure, future-proof non-repudiation infrastructures. It does not contain formal UC security proofs or symbolic verification details.

2. Related Work and Motivation

Numerous systems address integrity, privacy, or auditability using cryptographic primitives. Blockchain platforms like Hyperledger Fabric [1] focus on immutability and consensus but rely on centralized permissioning and lack formal zero-knowledge guarantees. SNARK-based solutions, such as zkLedger [10] and ZKAudit [7], enable privacy-preserving compliance but require trusted setup and offer limited scalability and post-quantum protection.

Lightweight approaches like ZLiTE [8] target IoT environments, prioritizing efficiency but neglecting adversarial modeling. Commercial tools (e.g., Microsoft Event Data Protection [4]) offer verifiable storage within closed, cloud-managed systems, limiting transparency and extensibility.

These systems address isolated challenges but fail to simultaneously deliver:

- Transparent zero-knowledge proofs without trusted setup.
- Long-term security against quantum threats.
- Symbolically modeled non-repudiation.
- Modular integration with audit and compliance workflows.

Newer platforms like Aleo and Aptos embed ZKPs into smart contracts, yet are unsuitable for structured logging, evidential replay, or modular attestation.

ZK-NR addresses these limitations by combining:

- STARK-based zero-knowledge compliance with transparent setup.
- Merkle-root commitments for durable batch integrity.
- Threshold BLS signatures for classical verifiability.
- Post-quantum resilience through Dilithium.

The protocol also raises a critical governance question: *can cryptographic evidence remain institutionally admissible if no human can interpret it?* This challenge, formalized in *Appendix C* as the *Invisible Authenticity Paradox*, motivates the ZK-NR design.

Rather than replacing existing frameworks, ZK-NR proposes a composable, verifiable, and privacy-preserving primitive as a foundation for future systems.

3. Protocol Overview and Roles

ZK-NR operates in distributed environments where multiple parties ensure verifiable, privacy-preserving commitments. The protocol defines four roles:

- *Committer (C)*: Generates a batch of entries (e.g., logs) and computes a Merkle root as a commitment.
- *Prover (P)*: Builds a zero-knowledge proof asserting that a committed entry satisfies a public predicate.

- *Verifier (V)*: Checks the validity of the proof and accompanying signatures.
- *Threshold Signers ($S_1 \ldots S_n$)*: Form a quorum that signs the Merkle root using both BLS and Dilithium.

Each entry e_i is hashed with SHA3-256 and included in a Merkle tree. The root R_M anchors the commitment. The Prover generates a STARK proof π that confirms compliance with predicate ϕ without revealing content. The triplet $(R_M, \pi, \sigma_{BLS}, \sigma_{Dilithium})$ forms the complete attestation.

This design avoids reliance on trusted third parties and operates over unauthenticated networks, tolerating Byzantine faults under the standard $f < \lfloor n/3 \rfloor$ assumption.

The *Figure 1 (Appendix A)* illustrates the full data flow and role interactions. The model assumes asynchronous delivery and authenticated, but not confidential, channels (e.g., TLS).

ZK-NR is composable: roles can be modularized within audit or forensic systems. Key lifecycle operations, including rotation and revocation, are planned (see Section 5).

4. Core Cryptographic Primitives

ZK-NR combines four independent cryptographic layers to produce a verifiable and privacy-preserving attestation. Each layer contributes to a specific property of the protocol:

- Integrity through Merkle commitments.
- Privacy via zero-knowledge proofs (STARK).
- Distributed accountability via threshold BLS signatures.
- Quantum resilience through Dilithium.

4.1. Merkle Commitments

Each data batch is hashed using SHA3-256, producing leaves. These are recursively combined to yield a Merkle root, which acts as a cryptographic anchor for the entire batch [9]. Merkle proofs support inclusion verifiability without exposing data content.

4.2. Zero-Knowledge Proofs via STARKs

The Prover constructs a STARK proof $\pi = \text{Prove}(w, \phi)$ using a witness composed of the entry and its Merkle path. STARKs offer transparency (no trusted setup), scalability, and post-quantum security [2]. The Fiat-Shamir heuristic converts the proof into a non-interactive form [6]. The predicate ϕ is public and encoded in the prover circuit.

4.3. Threshold BLS Signatures

BLS signatures are generated using threshold cryptography over a pairing-friendly curve (BLS12-381) [3]. A quorum of signers jointly produces a signature σ_{BLS} on the Merkle root. The signature is short, publicly verifiable, and efficiently aggregable. However, BLS is not post-quantum secure.

4.4. Post-Quantum Signatures with Dilithium

To ensure long-term non-repudiation, ZK-NR adds a second signature $\sigma_{Dilithium}$ over the same root, generated using CRYSTALS-Dilithium, a lattice-based signature scheme standardized by NIST [5]. This layer guarantees unforgeability under the Module-LWE assumption and complements BLS for hybrid security.

Together, the tuple $(\pi, \sigma_{BLS}, \sigma_{Dilithium})$ forms a multi-layered attestation, ensuring compliance, authenticity, and verifiability across classical and quantum-safe infrastructures.

5. Formalization Strategy and Symbolic Model

The ZK-NR protocol has been formally specified in the Tamarin Prover to verify its critical security properties under adversarial assumptions. The model includes eight symbolic transition rules and six lemmas addressing core guarantees.

5.1. Model Objectives

Each lemma formalizes a specific property:

- *lemma_nonRep_origin*: Every commitment must correspond to a valid proof and signature.
- *lemma_zeroKnowledge*: STARK proofs are simulatable and leak no knowledge.

- *lemma_blsUnforgeability*, *lemma_pqUnforgeability*: Ensure resistance to signature forgery under CDH (BLS) and Module-LWE (Dilithium).
- *lemma_uniqueness*: Only one valid proof per commitment.
- *lemma_binding*: Prevents reuse of signatures across distinct commitments.

5.2. Modeling Status

The symbolic specification is complete, and the Tamarin environment is correctly configured. However, due to resource constraints, the execution of formal proof traces is pending. No lemma has yet been resolved automatically. The full model is available for review and reproducibility. *Appendix B.2* maps lemmas to their corresponding properties, while *Appendix D.1* and *D.2* provide rule summaries and automation scripts.

5.3. Key Lifecycle Management

Key rotation and revocation are not yet implemented in the model but are conceptually supported. Future symbolic rules will handle scheduled key updates and revocation under threshold assumptions, using ephemeral key identifiers and state flags. Related lemmas will ensure revoked credentials cannot be reused to forge attestations.

5.4. Verification Scope

The model assumes perfect cryptography, authenticated but non-confidential channels, adversarial message control, and Byzantine quorum behavior bounded by $f < \lfloor n/3 \rfloor$. It builds upon the security assumptions of Merkle commitments [9], STARKs [2], the Fiat-Shamir heuristic [6], BLS signatures [3], and Dilithium [5]. This foundation supports future symbolic and computational extensions of ZK-NR's verification.

6. Proof-of-Concept Implementation

To evaluate the feasibility of ZK-NR beyond formal definitions, a functional proof-of-concept (PoC) has been developed. The prototype reproduces the core stages of the protocol pipeline, from commitment to verification, using open-source cryptographic libraries.

6.1. Environment and Tools

The implementation runs on Ubuntu 20.04 (WSL2) using: (i) Python 3.10 for Merkle root generation and orchestration, (ii) *py_ecc* for BLS signatures, (iii) Cairo 0.13.5 for dummy STARK execution, (iv) the C reference of CRYSTALS-Dilithium for post-quantum signatures [5], and (v) Bash scripts for automation and logging.

6.2. Components and Execution Flow

To evaluate ZK-NR beyond its formal model, a functional PoC was developed using modular scripts and open-source tools. The pipeline includes: (i) zk_merkle_generate.py to build the Merkle tree, (ii) zk_bls_sign.py for BLS signatures, (iii) zk_dilithium_sign.py for Dilithium signatures, (iv) zk_cairo_run.py to simulate ZK proofs, and (v) poc_run_all.sh to execute the full sequence with logging. Execution takes under 2 seconds on a mid-range laptop, producing the Merkle root, signatures, Cairo trace, and logs. The Dilithium component uses the NIST-aligned reference implementation [5].

6.3. Reproducibility

Each script produces versioned logs, and a README guides full replication. Code, sample data, and execution instructions are provided as supplementary material under the archive file `zk-nr_sup_mat.zip`, included with this article. A screenshot of the execution trace appears in *Figure 2 (Appendix A)*. Current STARK components are placeholders; integration of real predicates is ongoing.

7. Analysis, Open Issues and Design Challenges

While ZK-NR demonstrates architectural consistency and early feasibility, several challenges remain. This section reviews key trade-offs, limitations, and areas for future development.

7.1. Signature Layering

ZK-NR combines BLS (classical) and Dilithium (post-quantum) signatures to ensure both backward compatibility and quantum resilience. This dual scheme increases signature size and verification cost. Future enhancements may support adaptive policies for selecting which signature(s) to verify based on context.

7.2. Predicate Integration

The current prototype uses a placeholder Cairo program for STARK simulation. Real predicates and constraint logic remain to be implemented, limiting the protocol's expressiveness in audit and compliance scenarios.

7.3. Formal Model Scope

The Tamarin model covers symbolic properties but excludes timing, side-channel leakage, and computational soundness. Threshold behavior under partial compromise is also untested. Future work should incorporate hybrid analysis combining symbolic and computational models.

7.4. Interpretability and Institutions

As discussed in the *Invisible Authenticity Paradox* (*Appendix C*), ZK proofs may lack legal or operational interpretability. Possible mitigations include metadata binding, proof replay interfaces, and auxiliary attestations readable by non-experts.

7.5. Deployment and Interoperability

Despite modularity, real-world adoption will require standard APIs, schema compliance, and regulatory alignment. These integration layers, while out of scope here, are critical for practical deployment.

8. Conclusion and Roadmap

ZK-NR introduces a modular cryptographic protocol for non-repudiation in distributed systems where privacy, verifiability, and post-quantum security must coexist. It combines Merkle commitments, STARK proofs, and dual signature layers (BLS and Dilithium) into a reusable and formally specified structure.

This developmental contribution provides a composable, layered architecture, a Tamarin-ready symbolic model, a functional prototype, and a roadmap for proof integration and institutional usability. Pending tasks include full lemma validation, real predicate integration, hybrid signature management, and interfaces for human-auditable compliance.

ZK-NR is not a finalized system but a foundation for future infrastructures requiring durable, privacy-preserving evidence. Its design invites further refinement, evaluation, and integration into regulated environments.

Appendix

Appendix A. Figures and Diagrams

Appendix A.1. workflow

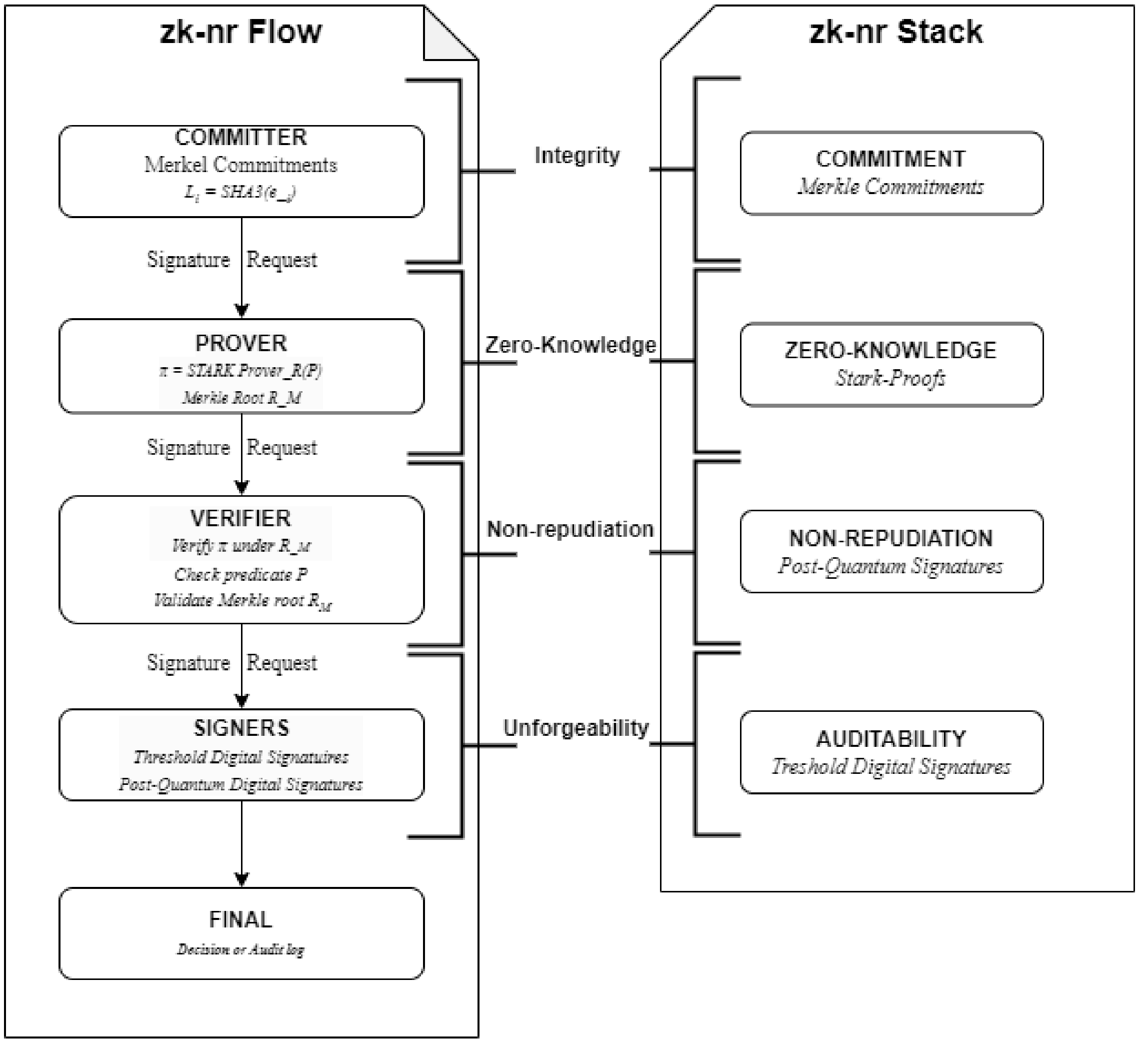
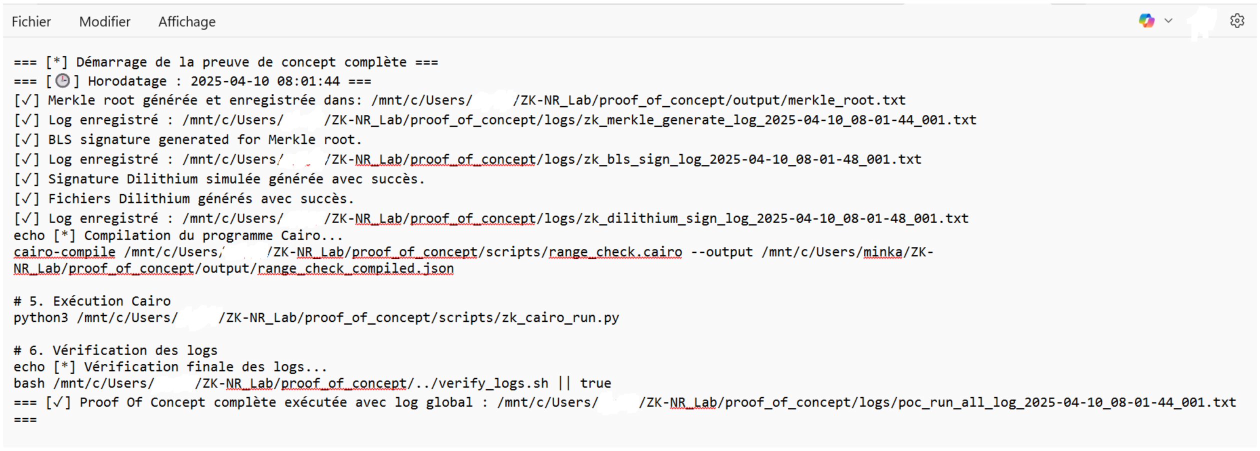


Figure A.1: Protocol workflow diagram – role interactions, commitments, proof generation, and dual signatures. The ZK-NR Flow (left) shows the sequential lifecycle of commitment, proof, verification, and signature generation. The ZK-NR Stack (right) depicts the cryptographic layers that sustain each property of the protocol: integrity, zero-knowledge, non-repudiation, and unforgeability.

Appendix A.2. PoC



```
=== [*] Démarrage de la preuve de concept complète ===
=== [🕒] Horodatage : 2025-04-10 08:01:44 ===
[✓] Merkle root générée et enregistrée dans: /mnt/c/Users/      /ZK-NR_Lab/proof_of_concept/output/merkle_root.txt
[✓] Log enregistré : /mnt/c/Users/      /ZK-NR_Lab/proof_of_concept/logs/zk_merkle_generate_log_2025-04-10_08-01-44_001.txt
[✓] BLS signature generated for Merkle root.
[✓] Log enregistré : /mnt/c/Users/      /ZK-NR_Lab/proof_of_concept/logs/zk_bls_sign_log_2025-04-10_08-01-48_001.txt
[✓] Signature Dilithium simulée générée avec succès.
[✓] Fichiers Dilithium générés avec succès.
[✓] Log enregistré : /mnt/c/Users/      /ZK-NR_Lab/proof_of_concept/logs/zk_dilithium_sign_log_2025-04-10_08-01-48_001.txt
echo [*] Compilation du programme Cairo...
cairo-compile /mnt/c/Users/      /ZK-NR_Lab/proof_of_concept/scripts/range_check.cairo --output /mnt/c/Users/minka/ZK-
NR_Lab/proof_of_concept/output/range_check_compiled.json

# 5. Exécution Cairo
python3 /mnt/c/Users/      /ZK-NR_Lab/proof_of_concept/scripts/zk_cairo_run.py

# 6. Vérification des logs
echo [*] Vérification finale des logs...
bash /mnt/c/Users/      /ZK-NR_Lab/proof_of_concept/../verify_logs.sh || true
=== [✓] Proof Of Concept complète exécutée avec log global : /mnt/c/Users/      /ZK-NR_Lab/proof_of_concept/logs/poc_run_all_log_2025-04-10_08-01-44_001.txt
===
```

Figure A.2: Screenshot of the complete proof-of-concept execution pipeline on a mid-range Linux system. This screenshot shows the successful execution of the ZK-NR prototype across its full lifecycle (poc_run_all.sh): Merkle root generation, BLS and Dilithium signature issuance, Cairo-based zero-knowledge proof compilation, and final audit log verification. All artifacts were generated in an isolated local environment, and the global execution log confirms end-to-end consistency of the ZK-NR workflow. The username in the path have been wipe for anonymization purpose.

Appendix B. Tables and Structured Elements

Appendix B.1. Tamarin Rules (Summary)

Table B.1: Summary of Tamarin transition rules

N°	Rule ID	Description
1	Gen_C	Fresh entry and Merkle commitment
2	Gen_Proof	Proof generation over Merkle path
3	Sign_BLS	Threshold BLS signature on Merkle root
4	Sign_PQ	Dilithium signature on same root
5	Verify_All	Final attestation: verify π, σ_{BLS}, σ_{PQ}
6	Replay_Attack	Adversarial reuse of prior commitments
7	Key_Compromise	Simulates quorum signature degradation
8	Network_Reorder	Models Dolev-Yao channel reordering

Appendix B.2. Lemma Mapping and Security Properties

Table B.2: Lemma mapping to security properties

N°	Lemma Identifier	Security Property	Cryptographic Basis
1	lemma_nonRep_origin	Non-repudiation	Merkle + BLS + Dilithium
2	lemma_zeroKnowledge	Zero-knowledge compliance	STARK (Fiat-Shamir)
3	lemma_blsUnforgeability	BLS signature unforgeability	BLS12-381, CDH
4	lemma_pqUnforgeability	PQ signature unforgeability	Dilithium, Module-LWE
5	lemma_uniqueness	Proof uniqueness	STARK witness commitment
6	lemma_binding	Signature binding to root	Merkle + signature replay

Appendix B.3. Protocol Comparison with Existing Solutions

Table B.3: Comparison with existing systems

N°	System	ZK Proof	PQ	Signature Type	Formal Model	Trusted Setup
1	zkLedger [10]	SNARK	No	None	Partial	Yes
2	ZLiTE [8]	ZK	No	ECC	No	No
3	Fabric [1]	None	No	ECDSA	No	N/A
4	Event Log [4]	None	No	Cloud-controlled	No	N/A
5	ZK-NR	STARK	Yes	BLS + Dilithium	Yes (symbolic)	No

Appendix B.4. Cryptographic Assumptions by Component

Table B.4: Cryptographic assumptions

N°	Component	Assumption Base
1	Merkle Tree	Collision resistance (SHA3-256)
2	ZK Proof	Soundness, Zero-Knowledge (STARK)
3	BLS Signature	Computational Diffie-Hellman (BLS12-381)
4	Dilithium	Module-LWE (Lattice-based PQ)

Appendix B.5. Performance Metrics (PoC)

Table B.5: Proof-of-concept performance metrics

N°	Operation	Avg Time (ms)	Output Size
1	Merkle Root Gen	47	32 bytes
2	BLS Signature	96	96 bytes
3	Dilithium Signature	173	∼2.7 KB
4	STARK Placeholder Trace	220	∼5 KB
5	Total PoC Execution	< 2 sec	∼8 KB

Appendix C. The Invisible Authenticity Paradox

"The more reliable and privacy-preserving a proof becomes, the less visible, explainable, and opposable it is."

This paradox captures a growing epistemic and governance challenge in cryptographic systems: the tension between cryptographic soundness and human interpretability. In traditional digital non-repudiation, evidence is based on visible elements, signatures, timestamps, logs, which are verifiable by machines and understandable by humans. They can be explained, challenged, or admitted in legal or audit contexts. However, Zero-Knowledge Proofs (ZKPs) introduce a new paradigm: they allow one to prove a statement's validity without disclosing its content. While this guarantees data minimization and privacy-preserving compliance, it also creates a situation where the proof is no longer human-readable, nor directly usable in adversarial or institutional settings. The Invisible Authenticity Paradox arises from this shift. As proofs become more secure and non-invasive, they simultaneously lose the characteristics that made them actionable in traditional systems of trust. A ZKP-based proof is:

- Validatable by an algorithm.
- But invisible to external observers.
- And non-explainable without the verification circuit and internal parameters.

This undermines traditional principles of accountability and opposability, especially when disputes arise or legal admissibility is required. The ZK-NR protocol is designed with this paradox in mind. It aims to:

- Preserve privacy through zero-knowledge mechanisms,
- Ensure cryptographic strength via post-quantum signatures,
- But also anchor proofs to auditable references, allowing validation over time without revealing sensitive content.

This hybrid construction recognizes that proof is not only a cryptographic artifact, but also a social, legal, and institutional object. It must be verifiable by systems, yet acceptable to humans. The paradox does not reject zero-knowledge, it frames its limitations and invites the design of systems that balance invisibility with trustworthiness.

Appendix D. Model Structure and Script Automation

Appendix D.1. Script Automation Components

- zk_merkle_generate.py: Merkle root generation and log.
- zk_bls_sign.py: BLS key pair and signature.
- zk_dilithium_sign.py: Post-quantum signature (binary call).
- zk_cairo_run.py: Dummy STARK trace placeholder.
- poc_run_all.sh: Workflow orchestration and full log export.

Scripts are parameterized for reproducibility and generate outputs in *logs*, *keys*, and *proofs* folders. Execution traces match the proof structure specified in the Tamarin model.

Appendix D.2. The ZK-NR Protocol theory (Tamarin version)

```
theory zk-nr
begin
builtins: hashing, signature
functions:
root/1, zproof/2, sign_bls/2, verify_bls/3,
sign_pq/2, verify_pq/3, commit/1, pk_bls/1, pk_pq/1,
checkproof/2

// === PROTOCOL RULES ===
// Commit batch of entries
rule CommitBatch:
[ Fr(x) ]
--[ Commit(batch_data) ]->
[ !Committed(batch_data, root(batch_data)),
Out(commit(root(batch_data))) ]

// Generate ZK proof (STARK placeholder)
rule GenerateZKProof:
[ !Committed(b, r), Fr(p) ]
--[ ZKProof(r) ]->
[ !Proof(r, zproof(p, r)),
Out(zproof(p, r)) ]

// Verify ZK proof
rule VerifyZKProof:
[ !Proof(r, zproof(p, r)) ]
--[ ZKProofVerified(r) ]->
[ Out(checkproof(zproof(p, r), r)) ]

// Sign with BLS
rule SignWithBLS:
[ Fr(sk), !Committed(_, r) ]
--[ BLS_Sign(r) ]->
[ Out(sign_bls(sk, r)) ]

rule VerifyBLS:
[ !Committed(_, r) ]
--[ BLS_Verified(r) ]->
[ Out(verify_bls(pk_bls(sk), r, sign_bls(sk, r))) ]

// Sign with Dilithium (PQ)
rule SignWithPQ:
[ Fr(sk2), !Committed(_, r) ]
--[ PQ_Sign(r) ]->
[ Out(sign_pq(sk2, r)) ]

rule VerifyPQ:
```

```
[ !Committed(_, r) ]
--[ PQ_Verified(r) ]->
[ Out(verify_pq(pk_pq(sk2), r, sign_pq(sk2, r))) ]

// Final verification of all steps
rule CombinedVerification:
[ !Proof(r, zproof(p, r)), !Committed(_, r) ]
--[ AllVerified(r) ]->
[ Out("ZK-NRVerified") ]

// === ADVERSARY RULES (Dolev-Yao) ===
// Leak BLS secret key
rule KeyCompromise_BLS:
[ Fr(sk) ]
--[ KeyCompromised(sk) ]->
[ Out(sk) ]

// Leak PQ secret key
rule KeyCompromise_PQ:
[ Fr(sk2) ]
--[ KeyCompromised(sk2) ]->
[ Out(sk2) ]

// Replay existing ZK proof
rule Replay_ZKProof:
[ !Proof(r, zproof(p, r)) ]
--[ ReplayAttempt(r) ]->
[ Out(zproof(p, r)) ]

// Replay of a signature
rule Replay_Signature:
[ Out(sig) ]
--[ SignatureReplay(sig) ]->
[ Out(sig) ]

// === ADVERSARY RULE (Byzantine behavior) ===
// Forge a fake signature on a bogus root without any prior commitment
rule ByzantineSignerBLS:
```

```
[ Fr(sk) ]
--[ ForgedBLS ]->
[ Out(sign_bls(sk, bogus_root)) ]

rule ByzantineSignerPQ:
[ Fr(sk2) ]
--[ ForgedPQ ]->
[ Out(sign_pq(sk2, bogus_root)) ]

// === LEMMAS (Symbolic Security Properties) ===
// Assumes CDH hardness for BLS and Module-LWE for PQ
lemma lemma_nonRep_origin:
"All r #i. Commit(r) @i ==> (Ex #j. ZKProofVerified(r) @j & (Ex #k.
BLS_Verified(r) @k | Ex #k. PQ_Verified(r) @k))"

lemma lemma_zeroKnowledge:
"not (Ex p1 p2 r1 r2 #i #j. zproof(p1, r1) != zproof(p2, r2) &
Out(zproof(p1, r1)) @i & Out(zproof(p2, r2)) @j)"

lemma lemma_blsUnforgeability:
"All r sig #i. Out(sig) @i & sig = sign_bls(sk, r) ==> (Ex #j. Fr(sk)
@j)"  // CDH assumption for BLS

lemma lemma_pqUnforgeability:
"All r sig #i. Out(sig) @i & sig = sign_pq(sk2, r) ==> (Ex #j.
Fr(sk2) @j)"  // Module-LWE assumption for Dilithium

lemma lemma_uniqueness:
"All r p1 p2 #i #j. Proof(r, zproof(p1, r)) @i & Proof(r, zproof(p2,
r)) @j ==> zproof(p1, r) = zproof(p2, r)"

lemma lemma_binding:
"All r1 r2 sig #i. Out(sig) @i & r1 != r2 & sig = sign_pq(sk2, r1)
==> not (verify_pq(pk_pq(sk2), r2, sig))"
end
```

References

[1] Androulaki, E., et al. (2018). *Hyperledger Fabric: A distributed operating system for permissioned blockchains*. Proceedings of the Thirteenth EuroSys Conference 2018. `https://doi.org/10.1145/3190508.3190538`

[2] Ben-Sasson, E., Bentov, I., Horesh, Y., & Riabzev, M. (2018). *Scalable, transparent, and post-quantum secure computational integrity*. IACR ePrint Archive. `https://eprint.iacr.org/2018/046`

[3] Boneh, D., Lynn, B., & Shacham, H. (2004). *Short signatures from the Weil pairing*. Journal of Cryptology, 17(4), 297–319. `https://doi.org/10.1007/s00145-004-0314-9`

[4] Desrosiers, A., Sharma, A., & Freeman, J. (2021). *Event Data Protection in Azure: Tamper-evident logging*. Microsoft Whitepaper. `https://learn.microsoft.com/en-us/security/azure-event-data-protection`

[5] Ducas, L., et al. (2018). *CRYSTALS-Dilithium: Digital signatures from module lattices*. EuroS&P 2018. `https://doi.org/10.1109/EuroSP.2018.00032`

[6] Fiat, A., & Shamir, A. (1987). *How to prove yourself: Practical solutions to identification and signature problems*. In CRYPTO'86. `https://doi.org/10.1007/3-540-47721-7_12`

[7] Kiayias, A., Miller, A., & Zindros, D. (2020). *Non-interactive proofs of proof-of-work*. Financial Cryptography and Data Security. `https://doi.org/10.1007/978-3-030-51280-4_27`

[8] Kottmann, M., Unterluggauer, T., Krenn, S., & Mangard, S. (2023). *ZLiTE: Lightweight Zero-Knowledge Logging for IoT*. CCS 2023. `https://doi.org/10.1145/3576915.3623169`

[9] Merkle, R. C. (1989). *A certified digital signature*. In CRYPTO'89. `https://doi.org/10.1007/0-387-34805-0_21`

[10] Narula, N., Vasquez, W., & Virza, M. (2018). *zkLedger: Privacy-preserving auditing for distributed ledgers*. USENIX NSDI 2018. `https://www.usenix.org/conference/nsdi18/presentation/narula`