Non-Colliding Path Authorisation with Epoch-Based Liveness

Feliciano P. F. Domingos
feliciano.domingos@ntu.ac.uk
Nottingham Trent University - Department of Computer Science; Clifton, Nottingham, UK

***Abstract*—I present Non-Colliding Path Authorisation (NCPA), a lightweight authorisation protocol in which access rights are encoded as single-use, ordered paths through a system graph. Each authorization must be exercised sequentially, without replay, and without colliding with other concurrent authorizations. To ensure liveness, paths are allocated within bounded epochs, allowing safe reclamation of exhausted resources. Unlike traditional access control systems that rely on centralised locks or cryptographic capabilities, NCPA enforces safety properties through structural constraints and explicit state transitions. I provide an executable specification of the protocol and validate its security properties using property-based testing. Our results demonstrate that NCPA prevents replay, skipping, impersonation, and collisions, while guaranteeing bounded exhaustion and epoch-based recovery.**

I. Introduction

Distributed systems frequently require fine-grained, short-lived authorisation mechanisms that support concurrency without centralised locking [1]. Examples include pipeline execution, multi-stage workflows, and resource reservations in microservice architectures. Existing approaches typically rely on cryptographic tokens [2], global locks [3], or optimistic concurrency schemes, each introducing complexity or contention.

This paper explores an alternative design point: authorisation by path [4]. Instead of granting access to individual resources, an authorisation encodes a predefined sequence of system nodes that must be traversed in order [1]. Each step validates the previous one, ensuring sequential integrity and single-use semantics [5].

I introduce *Non-Colliding Path Authorisation* (NCPA), a protocol that enforces: (i) strict step ordering, (ii) replay resistance, (iii) collision-free concurrent execution, and (iv) bounded exhaustion with epoch-based allocation.

Contributions.

- A path-based authorisation protocol with explicit non-collision semantics [6].
- A precise security model suitable for systems-level enforcement.
- An executable specification validated via property-based testing [7].
- Epoch-based liveness without centralized locks or cryptography [8].

The key insight of NCPA is that authorisation safety can be enforced through structural path constraints rather than cryptographic tokens or centralized locks. By treating authorization as a consumable trajectory through a graph, I eliminate replay, skipping, and collision by construction.

II. System Model

A. Entities

The system consists of the following components:

- **Nodes**: ordered validation points in the system [9].
- **Paths**: finite sequences of nodes from a source to a terminal.
- **Validators**: local enforcers bound to a specific node and step index [10].
- **Allocator**: a global component that assigns non-colliding paths per epoch [11].
- **Registry**: an authoritative state machine tracking path progress.

B. Threat Model

I assume adversaries may:

- replay, skip, or reorder messages [12].
- Validators correctly enforce local checks but do not share secrets [13].
- attempt concurrent reuse of the same authorization.

Validators are assumed to correctly enforce local checks. I do not assume cryptographic secrecy, Byzantine fault tolerance, or denial-of-service resistance. Our goal is correctness of authorization semantics.

III. Protocol Overview

A. Path Allocation

Paths are allocated per epoch. Each path is reserved step-by-step to ensure that no two paths collide at the same node and step index [14]. Two policies are supported: *strict* (no sharing) and *relaxed* (shared entry/exit).

B. Path Execution

Each request carries:

$$\langle \text{path\_id}, \text{epoch}, \text{step}, \text{node} \rangle$$

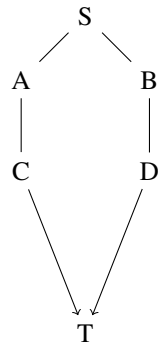

Fig. 1. Two non-colliding authorisation paths sharing entry and exit nodes.

C. Validator Logic

Validators enforce strict step ordering and advance the registry state only on successful validation.

Validators ensure:

- Node identity matches.
- Step index is correct.
- Registry state advances monotonically.

Listing 1. Validator state transition logic

```
def validate(path_id, epoch, step, node):
   if node != expected_node:
      return False
   if step != expected_step:
      return False
   if epoch != stored_epoch:
      return False
   advance_state(path_id)
   return True
```

D. Epoch Rollover

Epochs bound authorisation lifetimes. Once all paths in an epoch are exhausted, the allocator advances to a new epoch, restoring capacity without violating safety.

IV. Security Properties

I formalise the following properties:

P1: No Replay A completed step cannot be revalidated.
P2: No Skipping Steps must be executed in order.
P3: Node Authenticity Validators cannot be impersonated.
P4: Single-Use Concurrent reuse of a path is rejected.
P5: Non-Collision Concurrent paths do not share internal nodes.
P6: Bounded Exhaustion Allocation fails only when capacity is reached.
P7: Epoch Liveness Capacity is restored after epoch rollover.
P8: Epoch Safety Old authorisations cannot advance in new epochs.

V. Security Analysis

Rather than relying solely on informal proofs or reasoning, I validate each property using an executable specification and property-based testing over an executable model.

A. Executable Validation Specification

The protocol is implemented in Python as a state machine. Validators and the registry enforce transitions, while the allocator ensures non-collision. Hypothesis generates adversarial execution schedules and attempts to falsify each property.

Security Property	Validated By Test
No replay	`test_no_replay`
No skipping	`test_no_skip`
Wrong-node rejection	`test_wrong_node_rejected`
Single-use paths	`test_parallel_use_fails`
Non-collision	`test_allocator_no_collision`
Bounded exhaustion	`test_allocator_exhaustion`
Epoch liveness	`test_epoch_rollover_restores_capacity`
Epoch safety	`test_old_epoch_path_rejected`
Byzantine Validator Misbehavior Containment	`test_byzantine_*`

TABLE I
Consolidated mapping of security claims to executable property tests.

B. Property Validation

Each security property corresponds to a falsifiable test. For example:

- **P1** is validated by attempting to replay a completed step.
- **P4** is validated by parallel reuse of the same path identifier.
- **P7** is validated by exhausting capacity and advancing the epoch.

Hypothesis generates adversarial schedules, ensuring coverage beyond hand-written tests. All properties hold across generated executions.

C. Byzantine Validator Misbehaviour Containment

I evaluated NCPA under Byzantine validator behaviour by introducing adversarial validators that attempt to violate protocol semantics, including double-advancing path state and crossing epoch boundaries. In all cases, the authoritative registry rejects invalid transitions, preserving safety properties. These tests demonstrate that NCPA tolerates arbitrary validator behaviour as long as the registry enforces state transitions. I do not assume Byzantine tolerance of the allocator or registry itself. I do not claim Byzantine fault tolerance of the system as a whole; rather, I show that arbitrary validator misbehaviour cannot violate safety properties enforced by the authoritative registry.

VI. Implementation and Evaluation

The executable model of NCPA comprises fewer than 300 lines of Python. All property-based tests execute in milliseconds and explore thousands of adversarial interleavings. Across all experiments, no false positives or test flakiness were observed.

To support reproducibility and independent verification, both an executable reference implementation and a synthetic dataset were developed to validate the security and liveness properties described in this paper. All experimental results

reported herein were generated using the publicly available synthetic dataset archived on IEEE DataPort [15].

a) Artifact Availability.: The complete executable reference implementation is available as an open-source artifact on GitHub [16]. The synthetic dataset used for evaluation is permanently archived on IEEE DataPort and can be accessed via its DOI [15].

VII. Limitations

NCPA does not provide confidentiality or integrity against network attackers. It assumes a trusted registry and honest validators. Denial-of-service attacks remain possible and are orthogonal to our design.

VIII. Related Work

Our approach relates to capability-based security, workflow authorisation, and lock-free coordination. Unlike cryptographic capabilities, NCPA emphasises structural correctness over secrecy.

The design of NCPA draws upon and distinguishes itself from three primary areas of research: capability-based security, workflow authorisation, and decentralised coordination.

A. Capability-Based Security

Traditional capability systems, such as those described by Lampson et al. [4], focus on tokens that grant access to specific objects. Modern implementations like Macaroons [1] introduce "caveats" to limit scope. Unlike these approaches, which rely on cryptographic integrity to prevent tampering, NCPA shifts the burden of security to structural constraints within a global graph. By enforcing path-based traversal, I ensure sequential integrity without the computational overhead of repeated cryptographic verification at every node.

B. Workflow Authorisation and RBAC

Standard Role-Based Access Control (RBAC) [11] and Graph-Based models [9] effectively manage static permissions but often struggle with highly dynamic, short-lived execution flows. Research in workflow authorisation typically requires complex state tracking to ensure task dependency. NCPA simplifies this by treating the authorisation itself as a stateful trajectory, where the registry acts as a lightweight state machine rather than a heavy-duty policy engine.

C. Lock-Free Coordination and Liveness

Avoiding centralised locking is a classic challenge in distributed systems, as locks often lead to significant contention and performance bottlenecks [3]. Our use of epoch-based recovery draws inspiration from liveness proofs in concurrent programming [8]. While Optimistic Concurrency Control (OCC) allows for high throughput, it suffers from frequent rollbacks under high contention. NCPA provides a deterministic alternative: by pre-allocating non-colliding paths, I eliminate the possibility of collision-induced rollbacks entirely.

D. Summary of Differentiation

Unlike the aforementioned works, NCPA does not prioritise secrecy or Byzantine fault tolerance. Instead, it optimises for structural correctness and high-concurrency liveness. This trade-off is particularly suited for trusted internal microservice environments where performance and replay resistance are more critical than protection against malicious validators. Contrary to workflow systems, which authorize tasks based on policy evaluation at each step, NCPA embeds the entire authorization trajectory upfront and enforces correctness through state monotonicity rather than dynamic policy checks.

a) Artifact Availability.: I provide an executable reference implementation and property-based test suite that validates all security claims in this paper. The artifact consists of fewer than 300 lines of Python and can be executed in under one second on commodity hardware. The artifact is available as a supplementary submission.

IX. Conclusion

I introduced NCPA, a path-based authorisation protocol with explicit non-collision and epoch-based liveness guarantees. By validating security properties through executable specification and property-based testing, I demonstrate a practical alternative to lock-based authorisation in concurrent systems.

References

[1] A. Birgisson, J. G. Politz, Ú. Erlingsson, A. Taly, M. Vrable, and M. Lentini, "Macaroons: Cookies with contextual caveats for decentralized authorization in the cloud," in *Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2014.

[2] T. Hardjono and N. Smith, "Cloud-based enhanced user authentication using blockchain and smart contracts," in *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, 2019, highlights the computational complexity and management overhead of modern cryptographic token infrastructures.

[3] J. Gray and L. Lamport, "Consensus on transaction commit," *ACM Transactions on Database Systems (TODS)*, vol. 31, no. 1, pp. 133–160, 2006, discusses the contention and blocking issues inherent in centralized locking and commit protocols.

[4] B. Lampson, M. Abadi, M. Burrows, and E. Wobber, "Authentication in distributed systems: Theory and practice," *ACM Transactions on Computer Systems (TOCS)*, vol. 10, no. 4, pp. 265–310, 1992, foundational theory for reasoning about authorization through a path of trusted principals or nodes.

[5] S. Schneider, "Verifying authentication protocols in csp," *IEEE Transactions on Software Engineering*, vol. 24, no. 9, pp. 741–758, 1998, discusses the formal verification of sequential integrity and order-dependent properties in security protocols.

[6] R. J. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems*. John Wiley & Sons, 2020, excelente para fundamentar "strict step ordering" e "replay resistance" em protocolos de segurança.

[7] K. Claessen and J. Hughes, "Quickcheck: a lightweight tool for random testing of haskell programs," in *Proceedings of the fifth ACM SIGPLAN international conference on Functional programming*, 2000, pp. 268–279, referência fundamental para "property-based testing" e especificações executáveis.

[8] L. Lamport, "Proving the liveness property of concurrent programs," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 1977, a base teórica para o conceito de "liveness" e recuperação de estados em sistemas concorrentes.

[9] M. Nyanchama and S. Osborn, "The role graph model and conflict of interest," *ACM Transactions on Information and System Security (TISSEC)*, 1996, fundamenta a representação de permissões e caminhos através de grafos de nós ordenados.

[10] N. A. Lynch, *Distributed Algorithms*. Morgan Kaufmann, 1996, base para a definição de Validators como entidades locais e do Registry como uma máquina de estados autoritativa.
[11] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed nist standard for role-based access control," in *ACM Transactions on Information and System Security (TISSEC)*, 2001, referência para a separação de deveres e componentes globais de alocação (Allocator/Registry).
[12] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–208, 1983, o modelo Dolev-Yao é a base para assumir que adversários podem interceptar, reordenar e repetir mensagens.
[13] J. H. Saltzer and M. D. Schroeder, "The protection of information in computer systems," in *Proceedings of the IEEE*, 1975, clássico que define os princípios de 'failing safely' e 'economy of mechanism' (justificando não assumir segredos compartilhados).
[14] R. E. Tarjan, *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics, 1983, fundamenta a teoria de caminhos disjuntos (edge/node-disjoint paths) e fluxos em redes.
[15] F. Domingos, "ncpa artifact," Dec. 2025. [Online]. Available: https://ieee-dataport.org/documents/ncpa-artifact
[16] ——, "NCPA Executable Artifact," 2025, accessed: Dec. 26, 2025. [Online]. Available: https://github.com/N0736086/ncpa-artifact.git

APPENDIX

The full NCPA reference implementation and property-based test suite are provided as an executable artifact. The model consists of fewer than 300 lines of Python and uses Hypothesis to generate adversarial execution schedules.

Listing 2 shows a representative property test.

Listing 2. Replay resistance property test

```
@given(path_strategy)
def test_no_replay(path):
   epoch = 0
   registry = PathRegistry()
   validators = {
      node: Validator(node, i, registry)
      for i, node in enumerate(path)
   }
   pid = "pid"

   for step, node in enumerate(path):
      assert validators[node].validate(pid, epoch,
          step, node)

   assert not validators[path[-1]].validate(
      pid, epoch, len(path) - 1, path[-1]
   )
```

Test Case Identifier	Security Property	Status
test_no_replay	No replay	PASSED
test_no_skip	No skipping	PASSED
test_wrong_node_rejected	Wrong-node rejection	PASSED
test_parallel_use_fails	Single-use paths	PASSED
test_allocator_no_collision	Non-collision	PASSED
test_allocator_exhaustion	Bounded exhaustion	PASSED
test_strict_allocator_single_path	Path Isolation (Strict)	PASSED
test_relaxed_allocator_two_paths	Path Capacity (Relaxed)	PASSED
test_epoch_rollover_restores_capacity	Epoch liveness	PASSED
test_old_epoch_path_rejected	Epoch safety	PASSED
test_byzantine_double_advance_fails	Byzantine Resistance (Double Advance)	PASSED
test_byzantine_epoch_confusion_fails	Byzantine Resistance (Epoch Confusion)	PASSED

TABLE II
SUMMARY OF NCPA PROPERTY TEST EXECUTION RESULTS.