Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

PACE: Policy-Bound Multi-Party Authorization with State-Consistent Execution for Smart Contracts

Yaowei Sun
Tianjin University of Science and Technology
Cong Wang
wangcong.jcdd@tust.edu.cn
Tianjin University of Science and Technology
Maode Ma
Shenzhen University of Advanced Technology
Zhilu Zhang
Tianjin University
Yanan Zhang
Tianjin University of Science and Technology
Guangquan Xu
Tianjin University

Research Article

Keywords: Smart contracts, Multi-party authorization, Lattice-based cryptography, Policy-bound authorization, State-consistent execution

Posted Date: October 12th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11162863/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

PACE: **P**olicy-Bound Multi-Party **A**uthorization with State-**C**onsistent **E**xecution for Smart Contracts

Abstract

Multi-party authorization for smart contracts requires not only evidence of collective approval, but also consistency between the authorization decision, the active policy, the intended contract call, and the authorization state at execution time. Existing mechanisms often address policy updates, request binding, and authorization lifecycle management separately. To address this gap, we propose **PACE**, a policy-bound multi-party authorization framework with state-consistent execution for smart contracts. **PACE** associates the designated authority set with an active policy instance and binds the execution domain, target contract, operation, call data, session identifier, nonce, and deadline within a canonical execution context. A two-round lattice-based multi-signature instantiation generates joint evidence over this context. At execution time, the gateway revalidates the active policy, matches the submitted request against the approved context, checks the authorization state, and atomically couples authorization consumption with the protected operation. We formalize security through policy–decision consistency, decision–request binding, and authorization–execution consistency under the stated cryptographic and execution assumptions. A simulated prototype further evaluates the consistency checks, semantic ablations, and resource costs. The results show that the specified checks behave as intended in the tested scenarios, while collaborative signing and aggregation dominate the computational cost and context processing and contract-side verification remain comparatively stable within the evaluated settings.

Keywords: Smart contracts, Multi-party authorization, Lattice-based cryptography, Policy-bound authorization, State-consistent execution

1 Introduction

Smart contracts encode rules governing sensitive operations such as asset transfers, permission changes, and access to shared resources. Before a protected operation executes, the contract must determine whether the submitted request has received the required approval and whether that approval remains applicable. When several organizations or stakeholders jointly control a resource, authorization can be distributed across multiple authorities or authorization committees rather than relying on a single

decision point (Xing et al. 2025). In the setting considered here, multiple authorities jointly approve a specific execution request off chain, while a requester submits the resulting evidence for contract-side enforcement.

Consider authorities that approve a particular contract call under a given policy version. By the time the requester submits it, the policy may have changed, the authorization may have expired, or a prior transaction may already have consumed it. The signature can remain valid even though the approval no longer permits execution. The requester may also submit call parameters that differ from those reviewed by the authorities. Even when the request matches the approved call, separate handling of authorization consumption and the protected operation can leave a successful execution without a consumption record, or consume approval for an operation that fails. These cases require a consistent relationship between the governing policy, joint approval, the submitted request, and the authorization-state transition. We refer to this requirement as end-to-end authorization consistency. Achieving this consistency requires coordinating policy applicability, request binding, authorization validity, and execution-state transitions across the approval and execution stages.

1.1 Research Challenges

The separation between off-chain approval and contract-side execution raises four research questions (RQs). RQ1 and RQ2 correspond to policy–decision consistency and decision–request binding, respectively, while RQ3 and RQ4 jointly motivate authorization–execution consistency.

- **RQ1: Policy–decision consistency.** How can accepted joint approval be tied to the active policy and the authority configuration required at execution time? A configuration update may change who must participate, so approval from a previously authorized group need not remain applicable.
- **RQ2: Decision–request binding.** How can an authorization decision be restricted to the intended execution domain, target contract, operation, and call parameters? Because the requester controls the final submission, agreement among the authorities does not establish that the submitted call is the one they approved.
- **RQ3: Authorization lifecycle management.** How can a mechanism determine whether an authorization remains eligible for use and reject expired or previously consumed approval? Validity and prior use must be evaluated at execution time rather than inferred from the existence of cryptographic evidence.
- **RQ4: Authorization–execution consistency.** How can a protected operation be executed consistently with the verified request and the policy and authorization state applicable at execution time? The committed operation and the corresponding authorization-state transition must represent the same use of approval.

1.2 Research Motivation

Existing mechanisms provide several of the controls needed for this workflow. Smart-contract access control supports auditable policy evaluation (Di Francesco Maesa et al. 2019). SMACS combines off-chain decisions with scoped, expiring, one-time authorization tokens checked at protected calls (Liu et al. 2020). More recent decentralized

access-control systems distribute authorization-related decisions across multiple committees and use blockchain consensus to maintain consistent access-control state (Xing et al. 2025). Research on transaction execution consistency (Ivanov et al. 2023), misuse of existing approvals (Ruaro et al. 2025), and signature replay (Wang et al. 2026) further examines failures during authorization use and execution.

Joint-signature protocols authenticate a common message, including an execution context defined by the application (Nick et al. 2021; Chen 2023). This cryptographic interface does not prescribe which policy must be active when the request is executed, how the submitted call is compared with the signed context, or how authorization consumption is coordinated with the protected operation. These conditions must be defined by the application and enforced consistently with the approved message.

Our focus is the consistency of this composition. The active authority configuration, the jointly approved call, and the authorization-state transition at execution must refer to the same authorization instance, despite policy updates and an untrusted requester. This requires the signed context, contract-side acceptance conditions, and protected invocation to be specified together. We analyze when this composition prevents unauthorized execution under the stated assumptions. Table 1 compares representative mechanisms against RQ1–RQ4 within their stated scopes.

1.3 Main Contributions

To address the fragmented handling of policy updates, request binding, and authorization lifecycle management in existing mechanisms, we propose **PACE**, a policy-bound multi-party authorization framework with state-consistent execution for smart contracts. The joint-evidence layer uses the lattice-based DualMS construction for generation and verification (Chen 2023). It requires contributions from all authorities designated by the active policy under an n-out-of-n authorization mode; each honest authority approves the context before participating. The two-round interaction refers to one cryptographic signing attempt, excluding configuration establishment, transaction submission, and retries. Our contributions are as follows.

1. We formalize policy–decision consistency, decision–request binding, and authorization–execution consistency for stateful multi-party authorization. We analyze these requirements in the target-public-key security experiment and relate unauthorized successful execution to joint-signature forgery or a context-hash collision under the stated cryptographic, registration, and execution assumptions. The analysis remains in the classical random-oracle model.
2. We construct **PACE** around an epoch-specific authority configuration, a canonical execution context, and state-consistent execution. The configuration binds the active policy to the registered verification keys, while the context binds the execution domain, requested call, and authorization session. At execution, the gateway validates the evidence against the current policy, request, and authorization state, and atomically couples authorization consumption with the protected operation.
3. We implement a Python/SageMath prototype in a simulated execution environment. We examine selected consistency cases and semantic ablations, measure

the incremental overhead over the DualMS core, and evaluate computation, communication, configuration storage, and retry behavior. The experiments vary the authority count, call data length, and attempt budget to characterize costs within the tested settings; they do not measure deployment-specific blockchain latency or gas cost.

The remainder of the paper is organized as follows. Section 2 reviews related work. Sections 3 and 4 present the system and threat models and the security definitions, respectively. Section 5 describes **PACE**, and Section 6 analyzes its security. Section 7 reports the prototype evaluation, and Section 8 concludes the paper.

2 Related Work

We review related research on smart contract access control and multi-party approval, authorization use and execution security, and post-quantum joint signatures. These three areas provide the basis for positioning **PACE** within existing research.

2.1 Smart Contract Access Control and Multi-Party Authorization

Di Francesco Maesa et al. encode attribute-based access policies and attribute management as smart contracts, making access decisions auditable (Di Francesco Maesa et al. 2019). Policies are evaluated against the requesting subject, resource, action, and relevant attributes. Their smart-resource scenario also places enforcement on chain, linking access decisions to protected contract operations. These mechanisms are relevant to context-dependent authorization and enforcement, but do not define the lifecycle of a separately generated multi-party approval. In particular, contextual policy evaluation should be distinguished from binding joint evidence to the complete request and session context considered in RQ2.

SMACS combines off-chain authorization decisions with token verification in protected contracts (Liu et al. 2020). Argument tokens bind the caller, target contract, method, and arguments, directly addressing invocation binding within their scope (RQ2). Expiration checks and on-chain tracking of one-time tokens constrain authorization use and counter replay (RQ3). Contract-side verification also connects token validity to execution of the protected call (RQ4). These contributions provide a baseline for **PACE**; they should not be treated as missing simply because SMACS does not require multi-party approval. **PACE** additionally links designated authorities and their joint evidence to an active policy instance, addressing RQ1 within the same workflow.

Xing et al. propose X-LOCK, a decentralized access-control framework based on multi-committee attribute-based encryption and a sharding blockchain (Xing et al. 2025). Multiple nodes within each attribute committee collectively participate in attribute-key issuance and revocation through consensus, improving the decentralization, consistency, and fault tolerance of access-control decisions. X-LOCK is relevant to **PACE** because both distribute authorization-related responsibilities across multiple entities. However, X-LOCK focuses on decentralized attribute-based access control

and revocation, whereas **PACE** binds joint approval to an epoch-specific authority configuration and a concrete smart-contract invocation. It therefore does not provide the exact-call or single-use execution bindings considered in **PACE**.

2.2 Context Binding, Authorization Lifecycle, and Execution Security

Existing engineering specifications provide an important basis for comparison. EIP-712 defines structured data hashing and domain separation but explicitly provides no replay protection by itself (Bloemen et al. 2017). The ERC-2612 `permit` mechanism includes `owner`, `spender`, `value`, `nonce`, and `deadline` in the signed message and updates the nonce upon acceptance (Lundfall 2020). One-time use of a permit signature does not imply that the resulting allowance can be used only once. Nor does the deadline automatically become an expiry time for that allowance. Comparisons must therefore identify whether lifecycle management applies to signed requests, authorization credentials, or subsequent access rights. A general distinction based only on support for one-time authorization is insufficient.

TxT, proposed by Ivanov et al., uses transaction encapsulation to study the correspondence between testing and execution in Ethereum smart contracts (Ivanov et al. 2023). It addresses transaction execution semantics at a different level from the policy and authorization state checks in **PACE**. **PACE** determines whether current authorization conditions permit a specified request to execute. It does not guarantee that every business-logic state remains identical to its value when authorization was generated.

Recent studies further expose risks during the use of authorization. At USENIX Security 2025, Ruaro et al. studied controllable `transferFrom` calls in approved applications and introduced Osprey to detect and validate the resulting vulnerabilities (Ruaro et al. 2025). At ASE 2025, Liu et al. constructed a benchmark from real-world access control vulnerabilities to examine the coverage and blind spots of existing detection tools (Liu et al. 2025). LASiR, presented by Wang et al. at ICSE 2026, targets signature replay. It combines semantically assisted taint analysis with symbolic execution to examine signature reuse paths (Wang et al. 2026). These studies support including call conditions, signature usage state, and execution paths in authorization security analysis. Their primary goal, however, is to identify incorrect or dangerous behavior. They should therefore be distinguished from runtime multi-party authorization protocols in comparative analysis.

2.3 Multi-Signatures and Post-Quantum Joint Authorization

Multi-signatures provide cryptographic evidence that multiple independent signers approve a common message. MuSig2 implements two-round Schnorr multi-signatures (Nick et al. 2021). In lattice-based cryptography, Damgård et al. study two-round n-out-of-n signatures, multi-signatures, and their commitment constructions (Damgård et al. 2022). MuSig-L reduces the online signing phase to a single round (Boschini et al. 2022). DualMS further develops two-round lattice-based multi-signatures with trapdoor-free simulation (Chen 2023). These studies establish

lattice-based joint signing constructions with different round complexities and simulation strategies. They also show why comparisons must distinguish online signing rounds from total interaction rounds.

Post-quantum threshold signatures advance practical deployment under different participation policies. Threshold Raccoon studies practical threshold signatures based on standard lattice assumptions (del Pino et al. 2024). Ringtail, presented at IEEE S&P 2025, provides two-round threshold signatures based on Learning with Errors (LWE) (Boschini et al. 2025). At EUROCRYPT 2026, TalonG addresses communication efficiency in two-round signing (Gao et al. 2026), whereas Tweed develops two-round lattice-based threshold signatures that tolerate adaptive corruption (Jiang et al. 2026). These studies differ in their participation policies, security models, and resource costs. Efficiency comparisons must therefore consider threshold conditions, corruption models, preprocessing requirements, and the conventions used to measure communication.

More directly related to our n-out-of-n setting, Lai et al. construct two-round n-out-of-n signatures and multi-signatures in the quantum random oracle model (QROM) at PKC 2026 (Lai et al. 2026). Their result highlights the need to characterize security under quantum queries through the random oracle model and the relevant reduction conditions. Quorus, presented at USENIX Security 2026, uses multi-party computation to generate threshold signatures compatible with standard ML-DSA verification (Bienstock et al. 2026). Compatibility with standard verification and the authorization lifecycle consistency addressed by **PACE** are distinct dimensions of design and evaluation.

These protocols primarily specify how signers generate joint evidence and which cryptographic security properties that evidence satisfies under the relevant model. A multi-signature primitive can authenticate an application-defined message, including a message that contains execution-context fields. The primitive itself does not specify how that context is constructed, how the active policy is checked against it, or how authorization consumption is coordinated with execution. **PACE** focuses on this system-level composition of the cryptographic interface with smart contract authorization semantics.

2.4 Comparison and Research Positioning

Table 1 compares representative system mechanisms against the four questions introduced in Section 1.1. The comparison concerns the stated authorization requirements, rather than overall security or performance. Signature constructions and vulnerability detectors are discussed separately because they provide cryptographic components or diagnostic evidence, rather than complete runtime authorization workflows.

The distinctions in the table are substantive. Maesa et al. evaluate access context and enforce decisions for on-chain resources, but do not manage a separate single-use approval object. X-LOCK distributes access-control decisions across multi-node committees and uses blockchain consensus to improve consistency, fault tolerance, and revocation handling. Its authorization object, however, is an attribute-based access credential rather than a jointly approved smart-contract invocation. It therefore does

Table 1 Comparison of representative mechanisms against RQ1–RQ4.

Scheme	Multi-party authorization	RQ1	RQ2	RQ3	RQ4
Maesa et al. (2019)	No	No	Partial	No	Partial
X-LOCK (2025)	Partial	Partial	No	Partial	No
SMACS (2020)	No	No	Yes	Yes	Yes
TxT (2023)	No	No	No	No	Yes*
PACE (ours)	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**

Note: Yes indicates direct coverage; Partial indicates limited coverage; No indicates no explicit coverage. All assessments are within each scheme's stated scope. * TxT concerns tested-versus-actual execution, not authorization-state consistency.

not provide the policy-epoch, exact-call, and single-use execution bindings considered in **PACE**. SMACS enforces scoped, expiring, one-time tokens at protected calls, addressing RQ2–RQ4 within its token model. TxT addresses execution-state consistency under its transaction-replicability conditions, which supports the broader concern in RQ4. Its guarantee concerns tested transaction behavior, whereas **PACE** checks the policy and authorization state at execution. A positive entry therefore denotes coverage within the stated scope, not identical guarantees across schemes.

Among these selected mechanisms, none combines joint approval of a specific smart-contract invocation with all four requirements in one authorization lifecycle. This comparison motivates the integration studied in **PACE**; it does not imply that context binding or one-time enforcement is new in isolation. The framework links joint approval to the active policy and approved request, then checks authorization state and couples consumption with protected execution.

3 System and Threat Models

We consider smart contract operations whose execution requires approval from multiple designated authorities. This section defines the participating entities, their high-level interactions, and the adversarial and trust assumptions used in the subsequent security definitions. The concrete construction of **PACE** is deferred to Section 5.

3.1 System Model

Figure 1 illustrates the system model and the interaction between off-chain authorization and contract-side enforcement.

3.1.1 System Entities

Policy owner (PO). The policy owner defines a policy $P = (op, \mathcal{A}, \Gamma)$ for a protected operation. Here, $\mathcal{A}$ denotes the designated authority set and Γ specifies the authorization rules. These rules may include criteria evaluated locally by the authorities as well

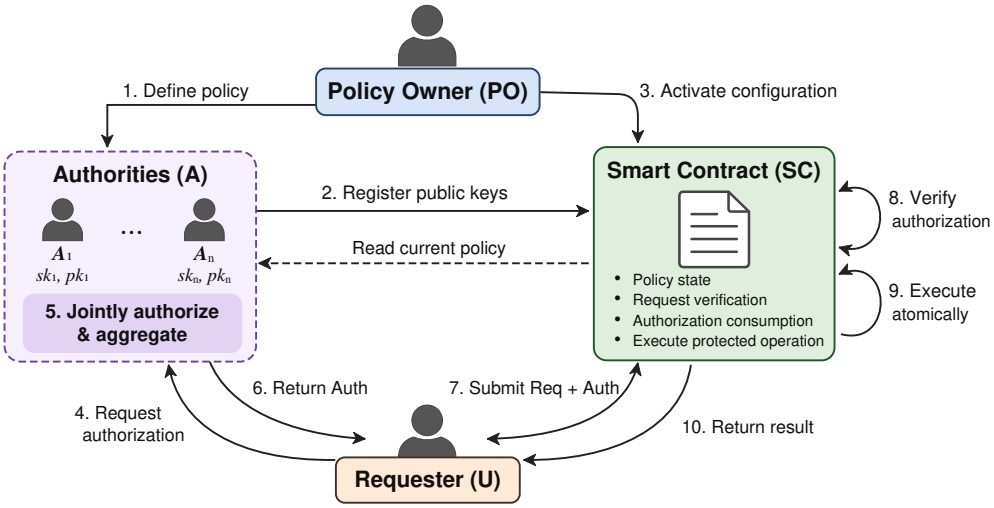


Fig. 1 System model and high-level workflow for smart contract multi-party authorization.

as deterministic conditions that can be enforced by the smart contract. Policy management also maintains the currently active policy version and its public verification configuration.

Authorization authorities ($\mathcal{A}$). The designated authorities are written compactly as $\mathcal{A} = \{A_i\}_{i=1}^{n}$. Each authority retains its private key locally and registers a public key under its identity. Authorities review requests according to the active policy and make local approval decisions. We denote these decisions by $D_{\mathcal{A}} = ((A_i, d_i))_{i=1}^{n}$. The current setting uses n-out-of-n authorization, so all authorities specified by the active policy are required to approve and participate. For an honest authority, signing participation follows its local approval; the resulting cryptographic evidence does not reveal or certify the internal decision process of a corrupted authority.

Requester (U). The requester prepares an execution request $Req = (op, target, callData)$ and obtains authorization from the designated authorities. It then submits the request together with the resulting authorization evidence to the on-chain execution component. The requester therefore also acts as the transaction submitter and is not assumed to be trusted.

Smart contract execution component (SC). The execution component maintains the active policy, registered public verification information, and authorization-related state. It serves as the contract-side authorization and execution-enforcement gateway. After a submission is accepted, SC invokes the protected contract identified by $target$. The distinction between SC and $target$ allows authorization enforcement to be separated from the business logic of the protected contract.

3.1.2 Authorization Workflow

At a high level, policy management and authority registration establish the public configuration used by the authorization process. For a protected operation, the requester

sends Req to the designated authorities for off-chain review. The authorities produce collective authorization evidence, which is returned to the requester as an authorization object $Auth$. The requester then submits $(Req, Auth)$ to SC, where contract-side validation is performed before the protected operation is invoked. The execution result is finally returned to the requester.

The system-level authorization instance is represented as

$$\mathcal{I}_{\text{auth}} = (P, Req, D_{\mathcal{A}}, SC). \tag{1}$$

This abstraction links the governing policy, the submitted request, the local approval decisions, and the execution component. These elements do not necessarily remain unchanged over time: the active policy or authorization state may evolve between approval and execution. Consequently, the existence of earlier approval alone is insufficient to determine whether a later execution is still authorized. Section 5 specifies how **PACE** instantiates and enforces these relationships.

3.2 Threat Model

3.2.1 Adversarial Capabilities and Corruption Model

We consider external adversaries, a potentially malicious requester, and colluding corrupted authorities. Public blockchain information and public verification data are observable. An adversary may also capture authorization messages and may submit modified, delayed, or repeated requests. In particular, the requester controls the final transaction submitted to SC; therefore, the request presented for execution cannot be assumed to be identical to the request previously reviewed by the authorities.

The adversary may corrupt a subset of the designated authorities and coordinate their behavior. A corrupted authority may deviate arbitrarily from the prescribed authorization procedure, choose its own registered public key, and send arbitrary authorization contributions. The current construction uses an n-out-of-n policy. For security analysis, we consider an honest authority in the target configuration and denote it by A_1. This analytical indexing does not change the protocol's canonical ordering or designate a permanently trusted identity. Following the DualMS security model (Chen 2023), each experiment uses a target public key pk_1 associated with A_1, which remains uncorrupted throughout that experiment. This experiment-level convention imposes no system-wide requirement that an authority retain the same key across policy epochs.

At configuration establishment, each authority identity is associated with one registered public key. Corrupted authorities may select the keys registered for their own identities, but they cannot replace the registered key of an uncorrupted authority or modify an active configuration $\text{Config}[p, e]$. Registered keys must be valid encodings in the public-key domain. As in the DualMS public-key-list setting, the model does not introduce an additional proof-of-possession requirement. These conditions separate adversarial control of corrupted signers from unauthorized changes to the registered identity–key association of an honest authority.

3.2.2 Trust Assumptions and Scope

The policy owner and the authorized configuration-management process are trusted. The blockchain state and deployed contract logic are assumed to remain intact, and SC follows the specified atomic and non-reentrant execution semantics. Accordingly, unauthorized modification of the policy-management mechanism, blockchain-consensus compromise, and contract code tampering are outside the threat model. Legitimate policy updates, changes of the active authority configuration through the authorized management process, and adversarially chosen execution requests remain within the model.

The security objective concerns unauthorized execution rather than availability. Because authorization is n-out-of-n, any required authority can prevent completion by withholding its contribution. Such refusal may cause denial of service, but it does not by itself constitute a violation of the authorization properties studied here. The threat model therefore focuses on whether an adversary can cause an execution to be accepted without the policy, approval, request, and authorization-state relationships required by the system.

3.2.3 Threats

The separation between policy management, off-chain approval, requester-side submission, and contract execution creates three classes of inconsistency.

Policy–decision inconsistency. Authorization evidence may remain available after the governing policy or designated authority configuration has changed. An adversary may therefore attempt to reuse approval produced under an obsolete policy version or a superseded authority set. It may also seek acceptance without approval from the honest authority required by the active configuration. The central risk is that cryptographic validity of previously generated evidence does not, by itself, establish that the evidence corresponds to the policy and authority configuration currently governing execution.

Decision–request misbinding. A malicious requester may obtain valid approval for one request and later associate that approval with a different operation, target contract, or $callData$. This threat is relevant because the requester controls the final submission, while the authorities make their decisions before that submission reaches the contract. Therefore, even correctly generated authorization evidence may be misapplied if the request accepted for execution is not the request that the authorities actually reviewed.

Authorization–execution inconsistency. Authorization eligibility may change after approval has been produced. An adversary may delay submission until an authorization is no longer valid, replay an authorization that has already been consumed, or submit competing uses associated with the same authorization session. A further inconsistency can arise when authorization consumption and the protected operation are committed separately, because the resulting state may no longer represent one coherent authorization outcome. The state considered here is limited to the policy and authorization state enforced by SC; the model does not require arbitrary business-state variables to remain unchanged between approval and execution.

Section 4 formalizes the corresponding consistency requirements and the stateful authorization-unforgeability experiment. These properties are analyzed in Section 6 under the adversarial capabilities and trust assumptions defined above.

4 Security Definitions

Under the system and adversarial assumptions in Section 3.2, secure execution requires approval from the active authority configuration, binding to the submitted request, and consistent authorization consumption at execution. These goals address the three threats above. We define their failure events and the authorization-unforgeability experiment below; they provide the security criteria for Section 6.

We define end-to-end authorization security through the consistency of the active policy, collective approval, submitted request, and authorization-state transition. Let $\mathcal{D}$ be a probabilistic polynomial-time adversary with the capabilities specified in the threat model; $\mathcal{A}$ denotes the authority set.

An authorization instance is $Auth = (P_e, \mu_{SC}, \tilde{\sigma})$, where $P_e = (op, \mathcal{A}, \Gamma, e)$, $Req = (op, target, callData)$, and $\mu_{SC} = (domain, P_e, target, op, callData, sid, nonce, deadline)$, where $domain = (\mathsf{chainID}, \mathrm{address}(SC), p)$ and $sid = (p, e, nonce)$. The signed message is $m = H(\mathrm{Encode}(\mu_{SC}))$. For execution, the contract reads $e_{\mathrm{cur}} = \mathrm{currentEpoch}[p]$ and $(P_{\mathrm{cur}}, L, \widetilde{pk}) = \mathrm{Config}[p, e_{\mathrm{cur}}]$.

Write $\mathrm{Execute}(p, Auth, Req, state_{SC}) = 1$ when Algorithm 3 successfully commits the protected operation. Here, $state_{SC}$ denotes the state immediately before the transaction. In each security experiment, A_1 denotes the honest target authority under the analytical indexing convention above. It remains uncorrupted throughout that experiment, while the adversary $\mathcal{D}$ may control the other authorities. The experiment uses a single target public key pk_1, and the target configurations considered in it contain this key. Approval and signing participation by A_1 are recorded when it approves the context and initiates the first signing round. Subsequent rejection or retry does not erase this record.

Definition 1 (End-to-End Authorization Security) An adversary $\mathcal{D}$ breaks end-to-end authorization security if it causes a successful execution for which at least one of F_{PD}, F_{DR}, or F_{AE} occurs. These events are defined below and each includes successful execution as a necessary condition.

The adversary's advantage is

$$\mathrm{Adv}_{\mathcal{D}}^{\text{E2E-Auth}}(\lambda) = \Pr[F_{PD} \vee F_{DR} \vee F_{AE}], \tag{2}$$

where the probability is over the randomness of the system and the adversary throughout the experiment. The mechanism is end-to-end secure if this advantage is negligible for every probabilistic polynomial-time adversary.

Definition 2 (Policy–Decision Consistency) The event F_{PD} occurs if $\mathrm{Execute}(p, Auth, Req, state_{SC}) = 1$, but the policy supplied in $Auth$ or embedded in μ_{SC} differs from P_{cur}, the request violates the contract-enforceable conditions of the active policy, or A_1 has not approved and initiated signing for the accepted context under the registered authority configuration.

Definition 3 (Decision-Request Binding) The event F_{DR} occurs if $\text{Execute}(p, Auth, Req, state_{SC}) = 1$, but the domain does not match the current execution environment, or Req differs from the operation, target, or call data encoded in μ_{SC}, or evidence from an approved context is accepted for a different context for which A_1 has not approved and initiated signing under the accepted configuration.

Contexts are compared through their canonical encodings, including domain, policy, and freshness fields. A different context separately approved and submitted to signing by A_1 is not an unauthorized substitution.

Definition 4 (Authorization-Execution Consistency) The event F_{AE} occurs if $\text{Execute}(p, Auth, Req, state_{SC}) = 1$, but the authorization violates any of $e = e_{\text{cur}}, sid = (p, e, nonce), sid \notin Used, \text{CurrentTime} \leq deadline$, evaluated against the pre-execution authorization state, or the committed operation differs from the verified request. The event also includes successful execution without committing the corresponding authorization consumption.

These conditions are evaluated under the assumed atomic, non-reentrant execution semantics: checks, consumption, and the protected operation form one transaction, and execution failure rolls back its effects. The definition concerns the policy and authorization state checked by the framework, rather than arbitrary business-state conditions.

Definition 5 (Stateful Authorization Unforgeability) The framework satisfies stateful authorization unforgeability if no probabilistic polynomial-time adversary $\mathcal{D}$ can cause successful execution of a context for which the experiment's honest target authority A_1 has not approved and initiated signing under the configuration accepted at execution.

The experiment between a challenger $\mathcal{C}$ and $\mathcal{D}$ is defined as follows.

Initialization. The challenger initializes the public parameters, registered authority configurations, and contract state. The challenger designates an honest target authority, denoted by A_1, and associates it with the target public key pk_1. This authority remains uncorrupted throughout the experiment, whose target configurations contain pk_1. The adversary receives the public information and the secret keys of the authorities it controls, subject to the threat model.

Authorization Query Phase. The adversary adaptively requests authorization interactions and execution attempts. For A_1, the challenger applies the approval procedure in Algorithm 2. When A_1 approves a context and initiates the first signing round, the challenger records its canonical context and configuration, together with (L, m), where $m = H(\text{Encode}(\mu_{SC}))$. The record persists even if the signing attempt subsequently aborts. Rejection and retries follow the prescribed algorithm.

The challenger processes execution requests using Algorithm 3 and maintains Config, currentEpoch, and $Used$. Configuration changes follow the authorized management process. Within this experiment, the target configurations remain restricted to those containing the target public key pk_1.

Forgery Phase. The adversary outputs $(p^*, Req^*, Auth^*)$, where $Auth^* = (P_e^*, \mu_{SC}^*, \tilde{\sigma}^*)$. It wins if

$$\text{Execute}(p^*, Auth^*, Req^*, state_{SC}) = 1 \tag{3}$$

and no record exists showing that A_1 approved μ^*_{SC} and initiated signing under the configuration accepted at execution.

Freshness is determined by the configuration and canonical context, not by the signature bytes. A new signature for an already approved context does not constitute a fresh authorization forgery, and failed signing attempts do not remove recorded approval and participation.

The adversary's advantage is

$$\mathrm{Adv}^{\mathrm{Auth}}_{\mathcal{D}}(\lambda) = \Pr[\mathsf{Game}^{\mathcal{D}}_{\mathrm{Auth}}(\lambda) = 1]. \tag{4}$$

The framework is statefully unforgeable if, for every probabilistic polynomial-time adversary $\mathcal{D}$,

$$\mathrm{Adv}^{\mathrm{Auth}}_{\mathcal{D}}(\lambda) \leq \mathrm{negl}(\lambda). \tag{5}$$

This framework-level property is analyzed through the underlying DualMS UF-CMA security and context-hash collision resistance in Section 6.2.

5 PACE: Design and Construction

Figure 2 shows the three components of **PACE**: policy-bound configuration establishment, context-bound joint authorization, and state-consistent execution through stateful verification and atomic consume-and-execute enforcement. The configuration-management process associates each policy epoch with its registered authorities and verification information. The authorities review a common execution context and generate joint evidence off chain. The requester submits the evidence to the execution gateway, which checks it against the request and the current policy and authorization state before invoking the protected operation.

The system-level authorization instance introduced in Section 3 is represented by the protocol-level object

$$Auth = \Phi(\mathcal{I}_{auth}) = (P_e, \mu_{SC}, \tilde{\sigma}),$$

where P_e is the epoch-specific policy, μ_{SC} is the canonical execution context, and $\tilde{\sigma}$ is the joint signature on the context-derived message. DualMS supplies joint-signature generation and verification; **PACE** specifies the policy and execution conditions governing their use. The n-out-of-n policy requires contributions from all designated authorities, and each honest authority approves the context before participating. The two rounds refer to one cryptographic signing attempt, excluding configuration establishment, transaction submission, and retries. Sections 5.1– 5.3 describe the three components.

5.1 Policy-Bound Configuration Establishment

Algorithm 1 implements configuration establishment, corresponding to steps 1–3 in Figure 2. Given a policy identifier p and a policy $P = (op, \mathcal{A}, \Gamma)$, the authorized

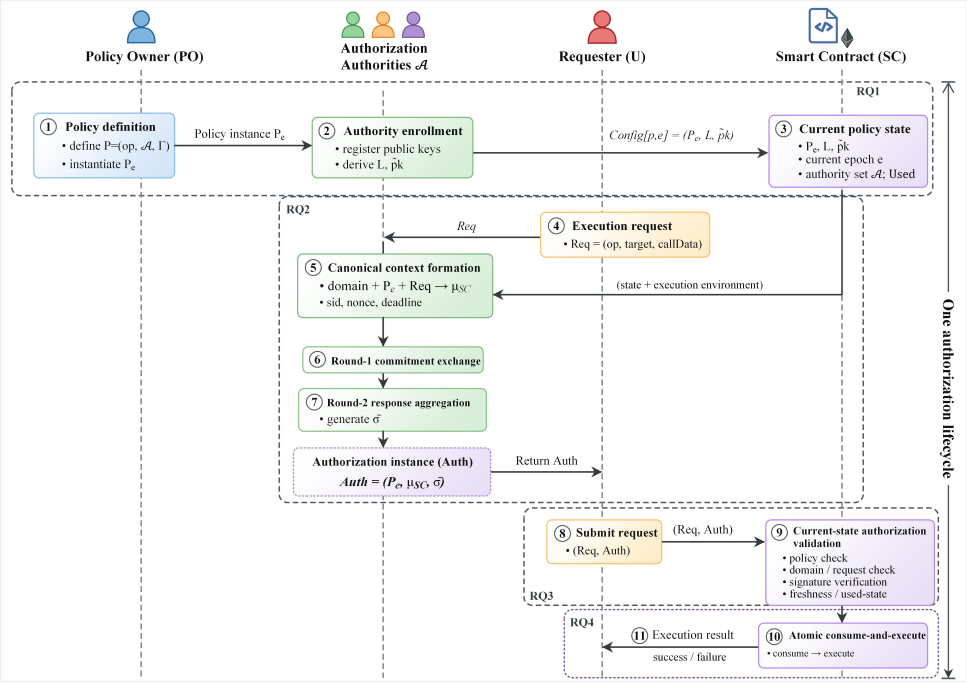


Fig. 2 Protocol workflow of **PACE**.

configuration-management process establishes

$$\mathrm{Config}[p, e] = (P_e, L, \widetilde{pk}),$$

where $P_e = (op, \mathcal{A}, \Gamma, e)$, L is the canonical list of registered public keys, and $\widetilde{pk}$ is the aggregate verification key. The record associates the policy with the keys used to authorize requests under epoch e.

The public matrix A and its extension $\bar{A} = [A \mid I_k]$ are established at system initialization. Each authority generates its key pair locally and registers its public key through the authorized registration process. This process links the authority identity to its verification key without exposing the secret key. Key aggregation follows DualMS: the coefficient $a_i = H_{\mathrm{agg}}(L, pk_i)$ depends on the registered key list, and the aggregate key combines the weighted public keys. The registration assumptions are those in Section 3.2.

Activation stores P_e and its verification information and advances currentEpoch$[p]$ within one atomic update. A new epoch distinguishes a configuration change even when some policy fields or keys are reused. Atomic activation prevents the current-epoch reference from pointing to a partially installed configuration. Joint authorization reads the active configuration, while execution retrieves it again from contract state;

Algorithm 1 Policy-Bound Configuration Establishment

Require: Policy identifier p and policy $P = (op, \mathcal{A}, \Gamma)$.
Ensure: Active configuration Config$[p, e]$.

At system initialization, sample $A \leftarrow_{\$} R_q^{k \times \ell}$ and set $\bar{A} = [A \mid I_k]$.
for $A_i \in \mathcal{A}$ **do**
    $sk_i \leftarrow_{\$} S_\eta^{\ell+k}$; $pk_i \leftarrow \bar{A} sk_i$.
    Retain sk_i locally; register pk_i under A_i.
end for
$L \leftarrow \text{CanonicalList}(pk_1, \ldots, pk_n)$.
$a_i \leftarrow H_{\text{agg}}(L, pk_i)$ for each $i \in [n]$.
$\widetilde{pk} \leftarrow \sum_{i=1}^{n} a_i pk_i$.
Atomically:
    $e \leftarrow \text{currentEpoch}[p] + 1$.
    $P_e \leftarrow (op, \mathcal{A}, \Gamma, e)$.
    Config$[p, e] \leftarrow (P_e, L, \widetilde{pk})$.
    currentEpoch$[p] \leftarrow e$.
return Config$[p, e]$.

a previously generated signature is not, by itself, sufficient to establish that its policy remains active.

5.2 Context-Bound Joint Authorization

Algorithm 2 covers request processing and joint signing, corresponding to steps 4–7 in Fig. 2. Its inputs are the policy identifier p, the request $Req = (op, target, callData)$, locally held authority keys, *nonce*, *deadline*, auxiliary information $\mathcal{B}$, and an attempt budget R. It retrieves the active configuration and constructs

$$\begin{aligned} domain &= (\mathsf{chainID}, \text{address}(SC), p), \\ sid &= (p, e, nonce), \\ \mu_{SC} &= (domain, P_e, target, op, callData, sid, nonce, deadline). \end{aligned}$$

The domain identifies the chain, execution gateway, and policy. The request fields identify the protected invocation, while *sid* and *deadline* identify the authorization session and its validity limit. The gateway SC verifies the authorization; $target$ identifies the protected contract that it subsequently invokes.

Each required authority reviews the same context under the active policy before participating in joint signing. An honest authority applies its approval criteria in Γ and may consult auxiliary information $\mathcal{B}$. This local review is not repeated by the contract. The evidence authenticates signing participation, not the internal decision of a corrupted authority. If a required approval is absent, generation terminates. Auxiliary information can inform local review but is neither signed nor enforced by the contract; information that changes the approved operation must be included in the signed context.

Algorithm 2 Context-Bound Joint Authorization

Require: p, Req, local keys $\{sk_i\}_{i=1}^{n}$, $nonce$, $deadline$, $\mathcal{B}$, and attempt budget R.
Ensure: $Auth = (P_e, \mu_{SC}, \tilde{\sigma})$ or $\perp$.

1: $e \leftarrow \text{currentEpoch}[p]$; retrieve $(P_e, L, \widetilde{pk})$ from $\text{Config}[p, e]$.
2: $domain \leftarrow (\mathsf{chainID}, \text{address}(SC), p)$, $sid \leftarrow (p, e, nonce)$.
3: Construct $\mu_{SC} = (domain, P_e, target, op, callData, sid, nonce, deadline)$.
4: Distribute $(\mu_{SC}, \mathcal{B})$ for policy and request approval; return $\perp$ if any required approval is absent.
5: $m \leftarrow H(\text{Encode}(\mu_{SC}))$, $B \leftarrow H_{\text{com}}(\widetilde{pk}, m)$; $\bar{B} \leftarrow [B \mid I_k]$.
6: $a_i \leftarrow H_{\text{agg}}(L, pk_i)$ for each $A_i \in \mathcal{A}$.
7: **for** at most R attempts **do**
8: For each $A_i \in \mathcal{A}$, sample fresh, isolated $y_i \leftarrow D_{\xi}^{\ell+k}$ and $r_i \leftarrow D_{\xi'}^{\ell'+k}$.
9: Each A_i computes $w_i = \bar{A}y_i + \bar{B}r_i$ and broadcasts $pm_1^i = w_i$.
10: Collect $M_1 = (pm_1^1, \ldots, pm_1^n)$; retry if incomplete.
11: Each A_i computes $c_i = H_{\text{sig}}\big(\widetilde{pk}, m, \sum_j w_j\big)$, $z_i = y_i + a_i c_i sk_i$.
12: Each A_i releases $pm_2^i = (c_i, z_i, r_i)$ with probability $\min\left\{1, \dfrac{D_{\xi}^{\ell+k}(z_i)}{M D_{a_i c_i sk_i, \xi}^{\ell+k}(z_i)}\right\}$; otherwise it releases $\perp$.
13: Collect $M_2 = (pm_2^1, \ldots, pm_2^n)$; retry if incomplete, rejected, or challenges differ.
14: Set $c \leftarrow c_1$; for each i, split $z_i = (z_i', z_i'')$ and $r_i = (r_i', r_i'')$.
15: Compute $\tilde{z} = \sum_i z_i'$, $\tilde{r} = \sum_i r_i'$, $\tilde{e} = \sum_i (z_i'' + r_i'')$.
16: Retry if $\|\tilde{z}\|_2 > \mathbb{B}_n$, $\|\tilde{r}\|_2 > \mathbb{B}_n'$, or $\|\tilde{e}\|_2 > \mathbb{B}_n''$.
17: $\tilde{\sigma} \leftarrow (c, \tilde{z}, \tilde{r}, \tilde{e})$; $Auth \leftarrow (P_e, \mu_{SC}, \tilde{\sigma})$.
18: **return** $Auth$.
19: **end for**
20: **return** $\perp$.

The authorities use canonical encoding to derive the common message

$$m = H(\text{Encode}(\mu_{SC})).$$

During the first round, each authority samples fresh randomness and broadcasts a commitment. During the second round, it derives the challenge from the aggregate key, the message, and the collected commitments, and computes its response. A response is released only if the prescribed rejection-sampling test accepts. Both rounds use the same key list and context-derived message.

Complete responses with a common challenge are aggregated into $\tilde{\sigma} = (c, \tilde{z}, \tilde{r}, \tilde{e})$, subject to the three aggregate norm bounds in Algorithm 2. An incomplete message set, a rejected response, inconsistent challenges, or an exceeded norm bound triggers a retry. Every retry restarts at Step 8 of Algorithm 2 with fresh randomness and isolated attempt state, while preserving μ_{SC} and sid. A retry therefore changes the signing attempt, not the authorization being requested. Successful generation returns $Auth$; exhaustion of the budget returns $\perp$.

5.3 State-Consistent Execution

Algorithm 3 processes the request and authorization submitted to the gateway, corresponding to steps 8–11 in Fig. 2. It checks whether the evidence is valid and whether the authorization remains applicable under the policy and authorization state at execution time. All checks and the protected call are performed within the atomic, non-reentrant execution model in Section 3.2.

Algorithm 3 Stateful Verification and Atomic Execution

Require: p, $Auth$, Req, and $state_{SC}$.
Ensure: Success or transaction revert.
Execution: One atomic, non-reentrant transaction; any failed check or target call reverts all effects.

1: Parse the submitted request $Req = (op_{\text{req}}, target_{\text{req}}, callData_{\text{req}})$.
2: Parse $Auth = (P_e, \mu_{SC}, \tilde{\sigma})$ and $P_e = (op_P, \mathcal{A}, \Gamma, e)$.
3: Parse $\mu_{SC} = (domain, P_\mu, target, op, callData, sid, nonce, deadline)$.
4: Read $\mathsf{chainID}$ and address(SC) from the current execution environment.
5: Read $e_{\text{cur}} = \text{currentEpoch}[p]$, retrieve $(P_{\text{cur}}, L, \widetilde{pk})$ from $\text{Config}[p, e_{\text{cur}}]$, and read $Used$ from $state_{SC}$.
6: Check $domain = (\mathsf{chainID}, \text{address}(SC), p)$.
7: Check $P_e = P_{\text{cur}}$ and $P_\mu = P_e$.
8: Check $(op, target, callData) = (op_{\text{req}}, target_{\text{req}}, callData_{\text{req}})$ and check that Req satisfies the contract-enforceable conditions in Γ.
9: Parse $\tilde{\sigma} = (c, \tilde{z}, \tilde{r}, \tilde{e})$; set $n \leftarrow |L|$.
10: Check canonical encodings, dimensions, and $c \in \mathcal{C}$.
11: Check $\|\tilde{z}\|_2 \le \mathbb{B}_n$, $\|\tilde{r}\|_2 \le \mathbb{B}'_n$, and $\|\tilde{e}\|_2 \le \mathbb{B}''_n$.
12: $m \leftarrow H(\text{Encode}(\mu_{SC}))$; $B \leftarrow H_{\text{com}}(\widetilde{pk}, m)$.
13: $\widehat{w} \leftarrow A\tilde{z} + B\tilde{r} + \tilde{e} - c\widetilde{pk}$.
14: Check $c = H_{\text{sig}}(\widetilde{pk}, m, \widehat{w})$.
15: Check $sid = (p, e, nonce)$ and $sid \notin Used$.
16: Check CurrentTime $\le deadline$.
17: Revert if any check fails.
18: $Used \leftarrow Used \cup \{sid\}$.
19: Invoke Call$(target_{\text{req}}, op_{\text{req}}, callData_{\text{req}})$ using the submitted request fields checked above; revert the entire transaction on failure.
20: **return** 1.

The gateway obtains the current epoch and verification configuration from contract state, rather than accepting a verification key selected by the submitter. Both the policy supplied in $Auth$ and the policy embedded in μ_{SC} must equal P_{cur}. The embedded domain must match the current chain, gateway address, and policy identifier. The context μ_{SC} specifies the authorized operation, target, and call data, whereas the submitted request $Req = (op_{\text{req}}, target_{\text{req}}, callData_{\text{req}})$ supplies the actual invocation arguments. The gateway checks that these request fields equal the corresponding

fields in μ_{SC} and satisfy the contract-enforceable conditions in Γ. After all checks pass, the protected call uses the checked fields from Req.

Signature verification checks canonical encodings, dimensions, challenge membership, and the three aggregate norm bounds. The gateway then derives m and B as in generation and reconstructs

$$\widehat{w} = A\tilde{z} + B\tilde{r} + \tilde{e} - c\widetilde{pk}.$$

It accepts the signature only if $c = H_{\mathrm{sig}}(\widetilde{pk}, m, \widehat{w})$. Signature validity and authorization applicability are separate conditions: the former is evaluated under the registered verification key, while the policy, domain, request, and freshness checks determine whether the evidence may be used for the submitted operation.

Freshness requires a session identifier consistent with the verified policy epoch, an unused authorization, and an unexpired deadline:

$$sid = (p, e, nonce), sid \notin Used, \mathrm{CurrentTime} \leq deadline.$$

The consumption record is indexed by sid, independently of the signature bytes. Another signature for the same authorization therefore does not provide another execution right. The deadline limits the period in which an unused authorization remains eligible for execution.

After all checks pass, the gateway adds sid to $Used$ before invoking the protected operation. A successful transaction commits both consumption and the target effects; a failed check or target call reverts the transaction. This permits at most one successful execution per authorization while rolling back consumption when the target call fails. The enforced consistency concerns the active policy, the approved request, and the authorization lifecycle; it does not preserve arbitrary business-state values between approval and execution.

6 Security Analysis

This section analyzes the consistency requirements defined in Section 4 and the cryptographic security of the authorization layer. The analysis follows the target-authority convention in Section 4. Within each experiment, A_1 denotes the uncorrupted target authority, and the target configuration contains its target public key pk_1. The adversary $\mathcal{D}$ may control the other authorities, subject to the stated threat model.

Authority A_1 participates only after approving the context under the relevant configuration. Its approval and signing participation are recorded when it initiates the first signing round, regardless of subsequent rejection or retry. Thus, the analysis distinguishes an unapproved context from an approved signing attempt that does not complete.

6.1 End-to-End Authorization Consistency Analysis

We analyze F_{PD}, F_{DR}, and F_{AE} using $Auth = (P_e, \mu_{SC}, \tilde{\sigma})$ and $\text{Execute}(p, Auth, Req, state_{SC}) = 1$ for successfully committed execution. The analysis assumes authorized configuration management, canonical encoding, and faithful execution of the atomic, non-reentrant procedure specified in Algorithm 3.

Let $\mathsf{Bad}_{\text{sig}}$ denote accepted signature evidence for a pair (L, m) on which A_1 has not initiated signing. Let Bad_H denote a collision between distinct canonically encoded authorization contexts. These events capture the cryptographic failures relevant to the consistency analysis. Their relation to the underlying security assumptions is established in Section 6.2.

6.1.1 Policy-Decision Consistency

Algorithm 1 binds the policy, registered public-key list, and aggregate verification key through $\text{Config}[p, e] = (P_e, L, \widetilde{pk})$. Atomic activation associates this configuration with $\text{currentEpoch}[p]$, providing a common reference for authorization generation and verification.

Algorithm 3 retrieves $(P_{\text{cur}}, L, \widetilde{pk})$ using the current epoch, rather than a version selected by the submitter. It checks that both the supplied policy and the policy in μ_{SC} equal P_{cur}, and that the request satisfies the active policy. An outdated policy, mismatched policy field, or request outside the active policy is therefore rejected before execution.

Consider accepted evidence whose context lacks approval from A_1 under the accepted configuration. The signature is verified on $m = H(\text{Encode}(\mu_{SC}))$. If A_1 has not initiated signing for (L, m), acceptance constitutes $\mathsf{Bad}_{\text{sig}}$. Otherwise, its signing interaction followed approval of some context μ'_{SC} under that configuration. Equality $\mu'_{SC} = \mu_{SC}$ contradicts the absence of approval. If the contexts differ, their equal message hashes establish Bad_H.

Consequently,

$$\Pr[F_{PD}] \leq \Pr[\mathsf{Bad}_{\text{sig}}] + \Pr[\mathsf{Bad}_H]. \tag{6}$$

Policy-bound registration identifies the required signing configuration, while the current-policy check determines whether its approval remains applicable at execution.

6.1.2 Decision-Request Binding

The honest authority A_1 approves μ_{SC} before initiating signing on $m = H(\text{Encode}(\mu_{SC}))$ under the registered configuration. Its signing participation therefore binds approval to the request and freshness fields in that context.

Algorithm 3 checks that the domain in μ_{SC} matches the current chain, gateway address, and policy identifier. It then checks that the operation, target, and call data in the submitted request Req equal the corresponding fields in μ_{SC}. A mismatch is rejected independently of signature validity. After all checks pass, the protected call uses the checked fields from Req, ensuring that the actual invocation matches the authenticated context.

To execute an unapproved request while passing the matching check, the adversary must submit a context containing the altered request. Let L be the public-key

list obtained from the accepted configuration and let $m = H(\text{Encode}(\mu_{SC}))$. If A_1 has not initiated signing for (L, m), acceptance of the evidence constitutes $\mathsf{Bad}_{\text{sig}}$. Otherwise, A_1 initiated signing after approving a context μ'_{SC} under that configuration. If $\mu'_{SC} \neq \mu_{SC}$, their equal message hashes establish Bad_H. If the contexts coincide, the submitted request was already approved and does not constitute an unauthorized substitution.

Consequently,

$$\Pr[F_{DR}] \leq \Pr[\mathsf{Bad}_{\text{sig}}] + \Pr[\mathsf{Bad}_H]. \tag{7}$$

The same argument covers changes to the policy, *sid*, *nonce*, and *deadline*, because these fields are included in the signed context. Auxiliary information $\mathcal{B}$ lies outside this binding guarantee and does not determine execution semantics.

6.1.3 Authorization-Execution Consistency

Authorization-execution consistency concerns the applicability and consumption of approval under the state immediately before execution. Algorithm 3 enforces $e = e_{\text{cur}}, sid = (p, e, nonce), sid \notin Used, \text{CurrentTime} \leq deadline$. The epoch equality follows from matching the supplied policy with the active policy. These checks reject outdated, expired, inconsistently identified, or consumed authorizations.

Consumption is indexed by *sid* rather than signature bytes. Consequently, obtaining another signature for the same authorization does not restore its execution eligibility. Signing retries preserve this identifier and do not alter the consumption condition.

After successful verification, the contract updates $Used \leftarrow Used \cup \{sid\}$ before invoking the protected operation. The non-reentrant entry point prevents nested execution through that entry. Concurrent submissions are evaluated in transaction order, so a successful use causes later submissions with the same *sid* to fail.

Consumption and the protected operation belong to one atomic transaction. The operation uses the submitted request fields that have been checked against μ_{SC}. Success commits both consumption and the target effects; a failed check or target call rolls back the transaction. The committed invocation therefore matches the approved request and cannot succeed without its corresponding consumption. Under these execution and state-integrity assumptions, $\Pr[F_{AE}] = 0$. This conclusion concerns the policy and authorization state enforced by the specified algorithm. Unauthorized changes to signed fields are covered by the preceding binding analysis. This property provides the formal basis for the state-consistent execution claimed by **PACE**.

6.2 Cryptographic Security Analysis

Let Σ denote the DualMS construction used in Algorithms 1–3. The authorization layer supplies the context-derived message and current-state enforcement; the underlying construction supplies joint-signature unforgeability. The following reduction uses the analysis-designated honest authority A_1 and the signing-participation convention stated above.

Security Claim. Assume that Σ is UF-CMA secure in the DualMS public-key-list model, in which the adversary may choose the public keys of corrupted signers

while the target public-key list contains the registered key of an uncorrupted authority. Assume further that H is collision resistant and that the execution assumptions in Section 3.2 hold. For any probabilistic polynomial-time adversary $\mathcal{D}$ against the target-public-key experiment defined in Definition 5, there exist algorithms $\mathcal{R}_{\text{sig}}$ and $\mathcal{R}_H$ such that

$$\text{Adv}_{\mathcal{D}}^{\text{Auth}}(\lambda) \leq \text{Adv}_{\Sigma}^{\text{UF-CMA}}(\mathcal{R}_{\text{sig}}; \lambda) + \text{Adv}_{H}^{\text{CR}}(\mathcal{R}_H; \lambda). \tag{8}$$

Here, unauthorized execution means successful execution of a context for which A_1 has not approved and initiated signing under the accepted configuration.

Proof. The signature reduction registers its challenge public key as pk_1 and forwards the signing interactions of A_1 to the underlying signing oracles. It preserves the registered public-key lists, messages, and protocol transcripts. Other-authority behavior follows the adversary's permitted capabilities. For corrupted authorities, the reduction uses the public keys selected by the adversary in their registered positions of L. It embeds the registered key pk_1 of the analysis-designated honest authority, which the adversary cannot replace in an accepted target configuration. The simulated target configurations contain pk_1, as required by the experiment, including after authorized configuration updates. The reduction uses the honest target already designated by the experiment, so it incurs no additional authority-selection guessing factor.

For each approved context, the reduction computes $m = H(\text{Encode}(\mu_{SC}))$ and records the context, configuration, and first-round signing query. It maintains Config, currentEpoch, and $Used$, and processes execution requests using Algorithm 3. These records persist when a signing attempt aborts.

Consider a successful unauthorized submission $(p^*, Req^*, Auth^*)$, where $Auth^* = (P_e^*, \mu_{SC}^*, \tilde{\sigma}^*)$. Let L^* be the public-key list read from the accepted current configuration and $m^* = H(\text{Encode}(\mu_{SC}^*))$. Successful verification supplies valid evidence under the aggregate key of L^*, which includes pk_1.

If A_1 has never initiated signing for (L^*, m^*), the reduction outputs $(L^*, m^*, \tilde{\sigma}^*)$ as a DualMS forgery. Otherwise, a recorded interaction exists for an approved context μ_{SC} with the same list and message. If $\mu_{SC} \neq \mu_{SC}^*$, then

$$H(\text{Encode}(\mu_{SC})) = H(\text{Encode}(\mu_{SC}^*))$$

gives a collision between distinct encodings. If the contexts coincide, A_1 already approved and initiated signing for the accepted context. The policy and identifier contained in that context identify the same authorization configuration, contradicting the winning condition.

The reduction enforces current-policy, request, and freshness checks directly through Algorithm 3; it does not assume framework unforgeability to simulate them. Unauthorized execution therefore yields one of the two cryptographic events, giving Eq. (8). **Rejection sampling and retries.** The reduction forwards the prescribed rejection outcomes and does not replace failed attempts with successful responses. A first-round query remains recorded even when the response is rejected, matching the underlying message-freshness condition. Retries retain the approved context and use fresh ephemeral randomness within budget R.

Table 2 P32 parameter profile used in the **PACE** prototype.

Parameter	P32 value or definition	Use in **PACE**
$n_{\max}$	32	maximum evaluated authority count
N	256	degree of $f(X) = X^N + 1$
q	37-bit prime, $q \equiv 5 \pmod 8$	modulus of R_q
R, R_q	$\mathbb{Z}[X]/(X^N+1)$, $\mathbb{Z}_q[X]/(X^N+1)$	computation domains
k, ℓ, ℓ'	$6, 7, 8$	matrix dimensions, with $\ell' = \ell + 1$
η, η'	$1, 1$	secret-key coefficient bounds
κ	60	challenge sparsity parameter
$\mathcal{C}$	$\{c \in R : \|c\|_\infty = 1, \|c\|_1 = \kappa\}$	challenge domain
t, γ	$13.5, 1.1$	rejection and tail-bound parameters
α	$8.5n$	rejection-sampling parameter
ξ, ξ'	$\xi = s,\ \xi' = s'$	Gaussian widths for y_i, r_i
M	$\exp(t/\alpha + 1/(2\alpha))$	single-party restart parameter
M_n	M evaluated at $\alpha = 8.5n$	authority-dependent single-party parameter
$\mathbb{B}_n, \mathbb{B}'_n, \mathbb{B}''_n$	derived from n, ξ, ξ', γ	aggregate verification bounds

The rejection parameter M and Gaussian parameters must satisfy the conditions of the adopted DualMS construction. Statistical simulation errors belong to the underlying unforgeability bound. The operational retry probabilities evaluated in Section 7.5 are not additional forgery-probability terms.

Security scope. The claim is conditional on the stated target-public-key experiment and the registration, parameter, and execution conditions. The formal argument remains in the classical random-oracle model. The lattice-based construction supports the post-quantum design objective without extending this proof to a quantum random-oracle model.

6.3 Security Summary

The consistency analysis in Section 6.1 and the reduction in Section 6.2 apply to the same authorization context and current configuration. Algorithm 3 integrates these guarantees by obtaining the registered verification key from current state, authenticating the context, matching the request, and checking authorization eligibility.

Let $m = H(\mathrm{Encode}(\mu_{SC}))$ and let $\widetilde{pk}$ be read from the current configuration. The verification-stage acceptance condition is

$$\begin{aligned} \mathrm{Accept} = [P_e = P_{\mathrm{cur}}] \wedge \mathrm{Vf}(\widetilde{pk}, m, \tilde{\sigma}) \\ \wedge \mathrm{Match}(\mu_{SC}, P_e, Req) \wedge \mathrm{Fresh}(\mu_{SC}, state_{SC}). \end{aligned} \tag{9}$$

Here, Vf abbreviates the encoding, dimension, norm, and challenge checks explicitly given in Algorithm 3. Match checks the embedded domain and policy, request fields, and compliance with the active policy, with the domain comparison evaluated against the current execution environment. Fresh checks the epoch, $sid = (p, e, nonce)$, deadline, and absence of sid from $Used$.

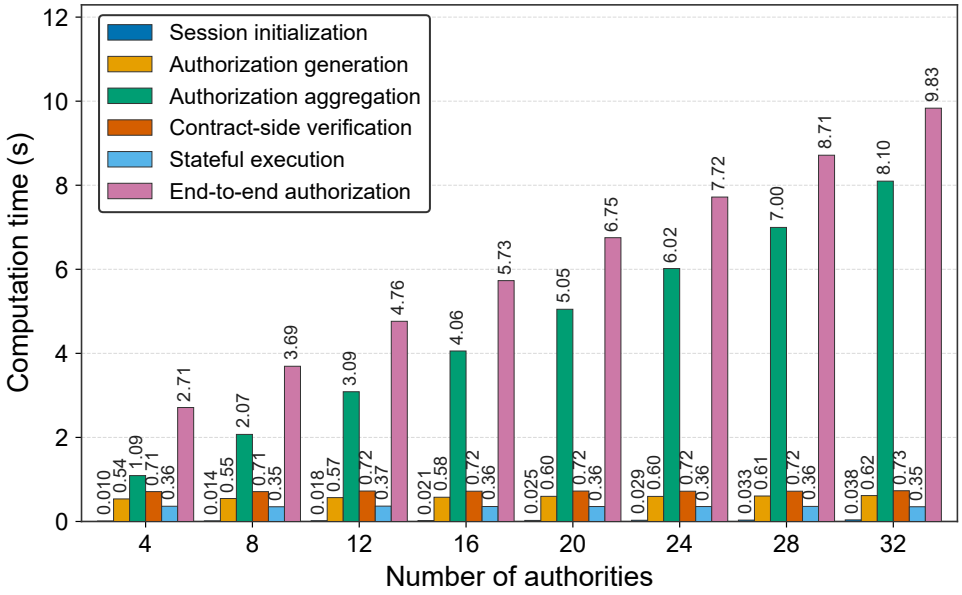


Fig. 3 Computational overhead under different numbers of authorities

Acceptance permits an execution attempt; successful execution additionally requires the target call to complete. Within the atomic, non-reentrant transaction, *sid* is consumed before the protected call uses the submitted request fields checked against μ_{SC}. Consumption and the target effects are committed on success and rolled back on failure.

Outside $\mathsf{Bad}_{\mathrm{sig}}$ and Bad_H, the policy and request checks exclude F_{PD} and F_{DR}, while the execution semantics exclude F_{AE}. The union of the three failure events is therefore contained in the same two cryptographic events:

$$\Pr[F_{PD} \vee F_{DR} \vee F_{AE}] \leq \Pr[\mathsf{Bad}_{\mathrm{sig}}] + \Pr[\mathsf{Bad}_H]. \tag{10}$$

Under the conditions of Section 6.2, this probability is negligible. The resulting guarantee connects approval by the experiment's honest target authority A_1 to the active policy, the intended request, and a valid authorization-state transition.

7 Prototype Implementation and Evaluation

We evaluated the Python 3.10/SageMath prototype on Ubuntu 24.04.3 LTS, using an Intel(R) Core(TM) i7-14600KF @ 3.50 GHz processor and 32.0 GB RAM.

Unless otherwise stated, experiments used the P32 DualMS profile from (Chen 2023), supporting up to 32 authorities. P32 fixes the base-ring and signature dimensions, while M_n and the aggregate response bounds are derived for each tested

$n = 4$–32. Table 2 summarizes the parameters; the definitions below specify the sampling dimensions and verification bounds.

For each authority, the secret key is sampled from $S_\eta^{\ell+k}$, and the public key is $pk_i = \bar{A}sk_i$, where $\bar{A} = [A \mid I_k]$ and $A \in R_q^{k\times\ell}$. The commitment matrix is $B = H_{\mathrm{com}}(\widetilde{pk}, m) \in R_q^{k\times\ell'}$, with $\bar{B} = [B \mid I_k]$. Each signing attempt samples $y_i \leftarrow D_\xi^{\ell+k}$ and $r_i \leftarrow D_{\xi'}^{\ell'+k}$.

The response bounds are

$$\begin{aligned} \mathbb{B}_n &= \sqrt{n}\mathbb{B}, \qquad \mathbb{B}'_n = \sqrt{n}\mathbb{B}', \qquad \mathbb{B}''_n = \sqrt{n}\mathbb{B}'', \\ \mathbb{B} &= \gamma\sqrt{\frac{\xi^2}{2\pi}N\ell}, \qquad \mathbb{B}' = \gamma\sqrt{\frac{\xi'^2}{2\pi}N\ell'}, \\ \mathbb{B}'' &= \gamma\sqrt{\frac{\xi^2+\xi'^2}{2\pi}Nk}. \end{aligned} \tag{11}$$

The functions H_{agg}, H_{com}, and H_{sig} are modeled as independent random oracles with outputs in $\mathcal{C}$, $R_q^{k\times\ell'}$, and $\mathcal{C}$, respectively. For a fixed authority configuration, $M = M_n$.

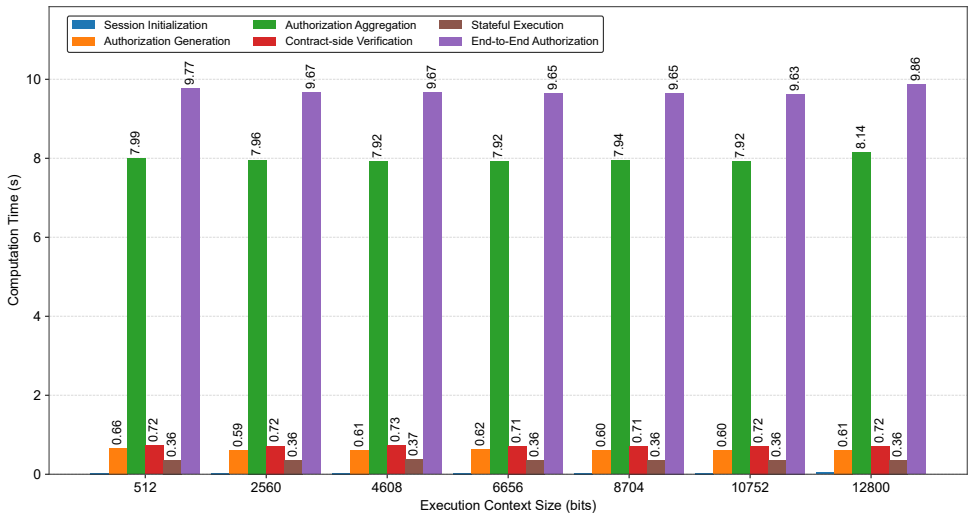


Fig. 4 Computational overhead under different execution context sizes

7.1 Prototype Implementation and Measurement Scope

The prototype simulated multiple authorities within a single process using a thread pool. Each authority maintained its own secret key and local signing state, and registered public keys were aggregated during configuration establishment. The authorities'

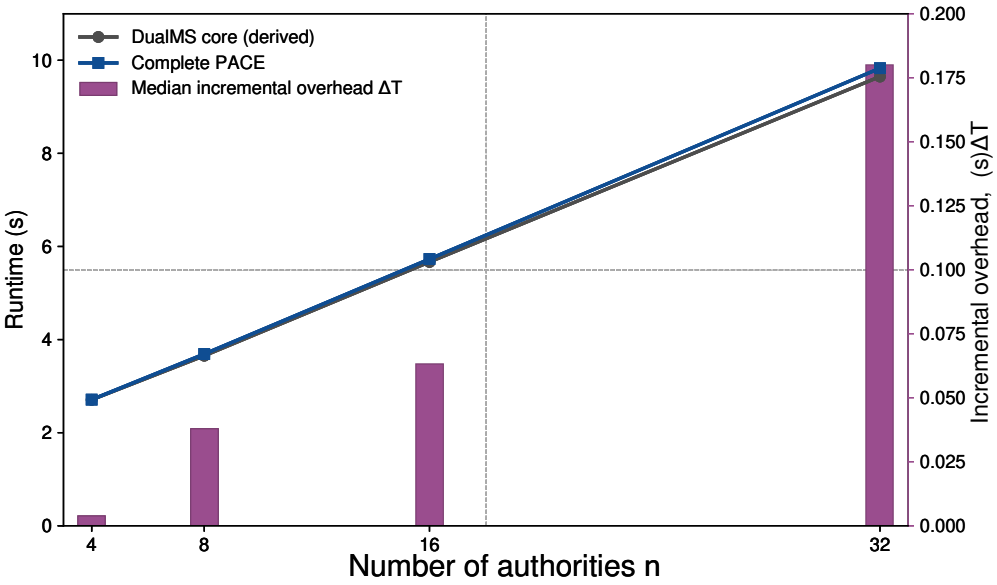


Fig. 5 System-level overhead of the complete PACE workflow over the DualMS cryptographic core. The left axis shows median runtimes for the DualMS core and the complete PACE workflow. The right axis shows the median of paired runtime differences, $\Delta T = T_{\text{Complete PACE}} - T_{\text{DualMS core}}$. All measurements use the P32 profile with $|callData| = 12800$ bits

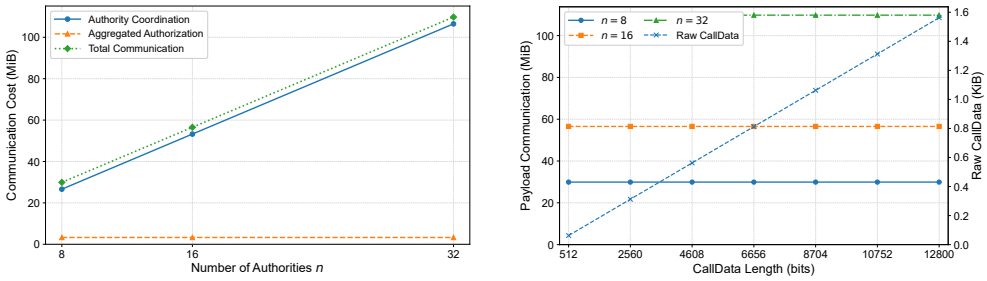


Fig. 6 Logical communication payload of PACE. **a** Communication overhead under different numbers of authorities. **b** Logical protocol payload under different call-data lengths

first-round contributions were collected into M_1 before second-round responses were computed. Responses accepted by rejection sampling formed M_2 and were aggregated into joint authorization evidence. Failed signing attempts were retried with fresh randomness and temporary state, while preserving the authorization context and session identifier.

Contract-side verification and execution were simulated in memory, without deployment to a live blockchain. The active policy record contained the policy epoch,

registered public-key list, and aggregate verification information, providing the active-configuration functions of Config and currentEpoch. The set $Used$ stored authorization consumption records. After verifying the evidence against the active configuration, the gateway invoked a separate local target callback that recorded SHA-256($callData$) and incremented a call counter. Before invocation, the gateway created a target-state snapshot and inserted the corresponding consumption record into $Used$. Callback failure restored the snapshot and removed the new record; success retained both updates. Execution requests were processed sequentially, and the test callback made no recursive calls to the gateway. No explicit non-reentrancy guard was implemented.

Communication was calculated from fixed-width representations of all contributions in M_1 and M_2, together with one aggregate signature per signing attempt. Each ring polynomial occupied

$$N \left\lceil \frac{\log_2 q}{8} \right\rceil$$

bytes. Experiments varying the call-data length additionally included the raw $callData$ payload. Transport protocol overhead, repeated broadcast delivery, and failed retries were excluded. The reported sizes therefore represent logical protocol payloads rather than measured network traffic.

For key generation and signing, thread CPU time was measured separately for each authority, and the maximum was reported for each stage. Setup, key aggregation, session initialization, evidence aggregation, contract-side verification, and stateful execution were measured as elapsed time. Contract-side verification covered signature verification and the policy, request, and freshness checks required before execution. Stateful execution was timed only after these checks had passed. It covered the target-state snapshot, authorization consumption, and the target callback, including rollback handling when the callback failed. Verification and stateful execution therefore used non-overlapping timing intervals. Runtime measurements were based on 100 independent executions, with medians calculated over successfully completed runs. Retry experiments report completion probabilities and the conditional mean number of attempts among successful instances, as specified in Section 7.5.

7.2 Consistency Invariants and Semantic Ablation

Table 3 links the execution tests to RQ1–RQ4 under the P32 profile. Each test records the authorization decision, execution count, change in `Used`, and committed target effects. Stale-policy and altered-request submissions were rejected, testing policy binding (RQ1) and request matching (RQ2). Expired authorizations were rejected, and replay produced no second execution (RQ3). When a target call failed, both authorization consumption and target effects were rolled back, checking the rollback invariant associated with RQ4. This test supports the atomic execution semantics assumed in Definition 4; it does not directly test F_{AE}, which requires successful execution. The valid case completed once and recorded the session as consumed.

The test-only ablations show how these checks affect execution. Removing the current-policy check admitted obsolete authorization; removing request matching allowed altered call data to be dispatched; and removing the single-use check permitted a second execution. Without atomic rollback, a failed call left a partial state update,

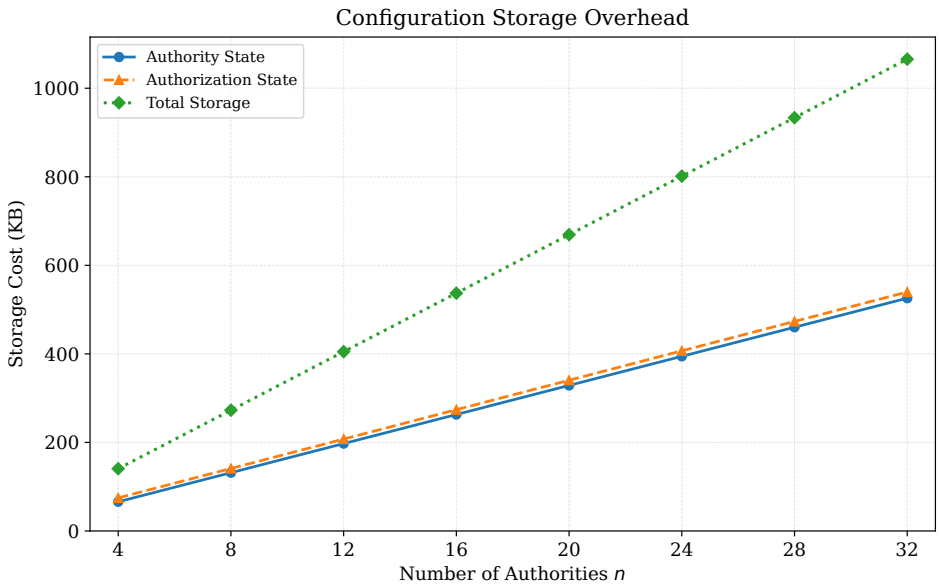


Fig. 7 Configuration storage overhead under different numbers of authorities.

Table 3 Functional evaluation and semantic ablations.

Scenario	PACE	Ablation
RQ1: Obsolete-policy authorization	Reject; no execution	Accepted without policy check
RQ2: Modified request	Reject; no execution	Accepted without request matching
RQ3: Expired authorization	Reject; unchanged state	–
RQ3: Reused `sid`	At most one execution	Second execution accepted without single-use check
RQ4: Target-call failure	Full rollback	Partial consumption without atomic rollback
RQ3–RQ4: Valid authorization	One execution; `sid` $\in$ `Used`	–

violating the rollback invariant. These contrasting outcomes support the tested authorization checks and rollback behavior, complementing the formal analysis without establishing the full security definitions.

7.3 Computational Evaluation

We measured session initialization, two-round authorization generation (Sign1 and Sign2), aggregation (SAgg), contract-side verification, and stateful execution.

Contract-side verification included Vf and the execution-time authorization checks. Stateful execution measured only the operations performed after these checks had passed. We examined how these costs varied with the authority count and call-data length. Session initialization reads the already activated configuration for each request. Increasing n enlarges this configuration and adds contributions to both M_1 and M_2. For a request $Req = (op, target, callData)$, the execution context is

$$\mu_{SC} = (domain, P_e, target, op, callData, sid, nonce, deadline),$$
$$domain = (\mathsf{chainID}, \mathrm{address}(SC), p).$$

The authorization message is $m = H(\mathrm{Encode}(\mu_{SC}))$. In the call-data-length experiment, we fixed the authority count at $n = 32$ and varied $|callData|$ from 512 to 12800 bits, while holding all other execution-context fields fixed. Increasing $|callData|$ enlarges the input to context encoding, hashing, and request matching, while leaving the cryptographic dimensions unchanged.

With the execution context fixed, aggregation time increased from approximately 1.09 s at $n = 4$ to 8.10 s at $n = 32$ (Fig. 3). As the authority count grows, more commitments and responses must be collected and processed into $\tilde{\sigma}$. This accounts for most of the increase in computation. Session initialization time also increased as the authority count grew. Contract-side verification remained near 0.71–0.73 s across this experiment: the contract checks one aggregated evidence object together with the policy, context, and freshness conditions, rather than verifying every authority's messages separately. Collaborative generation and aggregation therefore accounted for most of the observed scaling cost.

At $n = 32$, increasing $|callData|$ from 512 to 12800 bits produced no monotonic increase in the measured computation times (Fig. 4). Aggregation remained within 7.92–8.14 s. Contract-side verification remained near 0.72 s (0.71–0.73 s), while stateful execution was approximately 0.36 s in this experiment. Authorization-generation times varied without growing proportionally to the payload. These results are consistent with hashing the variable-length context into a fixed-length message before joint signing. Contract-side request and freshness checks add context processing, but do not enlarge the underlying lattice objects. Within the tested payload range, context binding therefore had a limited effect on the dominant computation.

The incremental experiment measured the cost of the authorization layer around the same P32 DualMS core. The core comprises Sign1+Sign2+SAgg+Vf; the full workflow also performs context construction, policy and state processing, and protected execution, with verification incorporated into contract-side enforcement. DualMS is the cryptographic reference for this cost decomposition. For each paired measurement, the increment is

$$\Delta T = T_{\text{Complete } \mathbf{PACE}} - T_{\text{DualMS core}}.$$

Both paths used $|callData| = 12800$ bits. The runtime curves report the median of each path, whereas the bars report the median of the paired differences ΔT. The latter is not the difference between the two runtime medians.

Figure 5 shows similar core and full-workflow runtime curves. The reported median increments were 0.0038 s, 0.0376 s, 0.0632 s, and 0.1800 s for $n = 4$, 8, 16, and

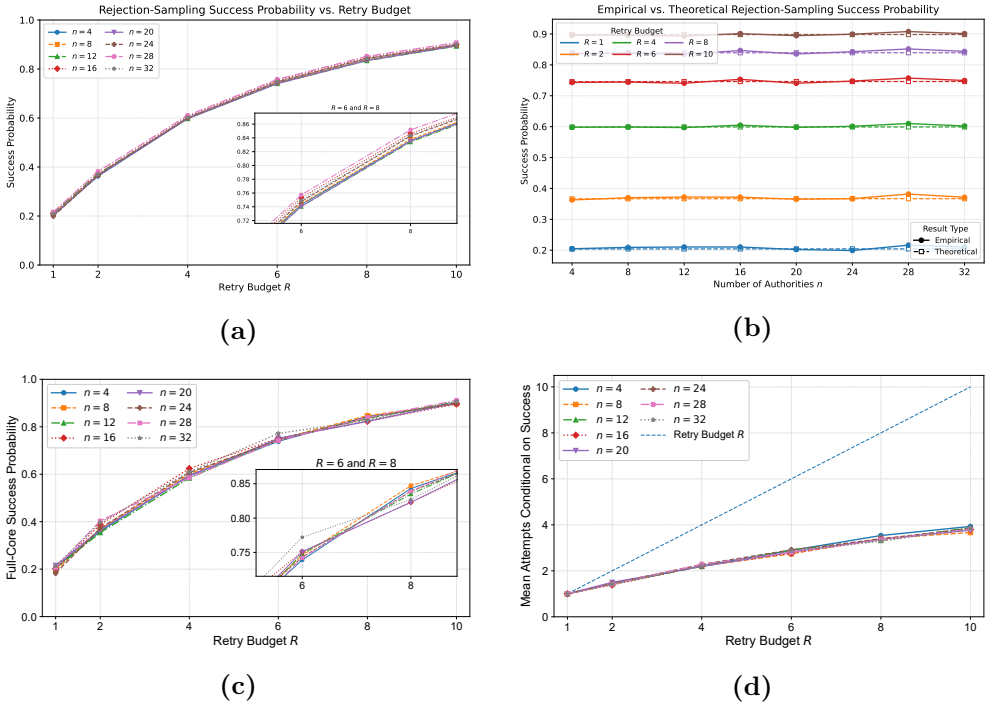


Fig. 8 Retry and completion evaluation of the proposed multi-party authorization mechanism. **a** Rejection-sampling success probability under different retry budgets. **b** Empirical and theoretical rejection-sampling success probabilities. **c** Full cryptographic-core success probability under different retry budgets. **d** Conditional mean number of attempts for successful authorization

32, respectively. Although the increment grew with the authority count, it remained small relative to the total runtime in these configurations. These results and the stage measurements indicate that collaborative signing and aggregation account for most of the runtime, with a smaller measured increment for context binding and stateful execution.

7.4 Communication and Storage Overhead

Communication consists of authority coordination and submission of the aggregated evidence. Each authority contributes one message to each round, giving

$$C_{\text{coord}} = C_{M_1} + C_{M_2}, \tag{12}$$

and

$$C_{\text{crypto}} = C_{\text{coord}} + C_{\tilde{\sigma}}. \tag{13}$$

Coordination grows with n, whereas aggregation produces a single evidence object with fixed dimensions under the adopted parameter set. The context-size experiment also counts the transmitted *callData* payload. Its length affects request transmission but does not change the dimensions of M_1, M_2, or $\tilde{\sigma}$.

Coordination traffic increased approximately linearly from 26 MiB at $n = 8$ to 107 MiB at $n = 32$, while aggregated evidence remained near 3 MiB (Fig. 6(a)). Total communication consequently grew from approximately 30 to 110 MiB. Adding authorities expands the two message sets and hence the cost of collecting contributions, while the evidence submitted to the contract retains its dimensions.

For $|callData| = 512$–12800 bits, the total logical protocol payload remained near 30, 56, and 110 MiB for $n = 8$, 16, and 32, respectively (Fig. 6(b)). The payload increased from only 0.0625 to 1.5625 KiB, so its contribution was too small to produce visible growth against the MiB-scale coordination traffic. The fixed-length context hash also prevents larger payloads from expanding the cryptographic messages. Authority count was therefore the dominant communication factor in these experiments.

The storage experiment measures Authority State, which includes public authority information and its registration record, and Authorization State, which contains the active policy record. Both retain information that grows with the authority configuration. As n increased from 4 to 32, Authority State grew from approximately 66 to 526 KB and Authorization State from 75 to 540 KB (Fig. 7). Their combined size increased from 141 KB to 1.07 MB. The approximately linear trends follow from registering additional authority information and associating it with the active policy, while retaining the same basic record structure. The measurements cover configuration storage as the authority set expands.

7.5 Retry, Completion, and Applicability

Rejection sampling during response generation can abort a signing attempt. Because the authorization policy requires all n authorities to participate, rejection of any response prevents completion of M_2 and requires a fresh attempt. This mechanism limits secret-key leakage through the response distribution (Chen, 2023; Boschini et al., 2022); its intentional aborts differ from the statistical simulation error in the security analysis.

We varied the authority count n and retry budget R. Under the implemented parameters, each authority accepts with probability approximately $1/M_n$, where M_n denotes the single-authority rejection parameter at authority count n:

$$M_n = \exp\left(\frac{t}{\alpha} + \frac{1}{2\alpha}\right), \qquad \alpha = 8.5n.$$

The algorithms write this parameter as M for a fixed configuration (Table 2). The leading joint acceptance probability is

$$p_{\mathrm{RS}}(n) = M_n^{-n}. \tag{14}$$

With at most R fresh attempts, the reference completion probability is

$$p_{\mathrm{RS}}^{(R)}(n) = 1 - \left(1 - M_n^{-n}\right)^R. \tag{15}$$

We evaluated rejection sampling in isolation against this reference, then measured the complete cryptographic core, including both signing rounds, aggregation, and verification.

For isolated rejection sampling, success probability rose from about 0.20 at $R = 1$ to 0.60 at $R = 4$, 0.75 at $R = 6$, and nearly 0.90 at $R = 10$ (Fig. 8(a)). The curves for $n = 4$–32 remained close because the authority-dependent values of M_n compensate for the greater number of required responses. Agreement between empirical results and Eq. (15) further supports this explanation (Fig. 8(b)).

The complete core followed a similar trend: success probability was approximately 0.20 at $R = 1$, 0.60 at $R = 4$, above 0.80 at $R = 8$, and close to 0.90 at $R = 10$ (Fig. 8(c)). Once acceptable responses were available, aggregation and verification caused little additional loss of completion probability. Rejection sampling thus accounted for most honest retries under the tested parameters.

Among successful instances, the conditional mean number of attempts increased with R but remained below 4 even at $R = 10$ (Fig. 8(d)). A larger budget admits instances that require more retries, while successful instances usually finish before exhausting that budget. This conditional statistic describes completed instances; the completion-probability curves also account for failures. These measurements show how the retry budget controls cryptographic completion while retaining the n-out-of-n participation requirement. The observed stability across authority counts depends on the adopted M_n configuration.

8 Conclusion

This paper presented **PACE**, a policy-bound multi-party authorization framework with state-consistent execution for protected smart contract operations. **PACE** links an epoch-specific authority configuration, a canonical execution context, joint authorization evidence, and execution-time authorization state within a unified authorization lifecycle. We formalized the security objectives through policy–decision consistency, decision–request binding, and authorization–execution consistency, and analyzed them under the stated cryptographic, registration, and execution assumptions. A Python/SageMath prototype was further used to examine consistency checks, semantic ablations, and resource costs. The results show that the specified authorization checks behave as intended in the tested scenarios, while collaborative signing and evidence aggregation dominate the computational cost and communication and configuration storage grow with the number of authorities.

The current framework uses an n-out-of-n authorization model and assumes trusted configuration management and atomic execution semantics. Its security analysis follows a target-public-key experiment with an uncorrupted authority, denoted by A_1 for analysis. The prototype was evaluated in a simulated environment, and the formal analysis remains in the classical random-oracle model. Future work will investigate threshold authorization and extend the security analysis to configurations without a common honest target key, including changes in authority membership or keys. Further work will also examine stronger post-quantum security models and deployment-level

costs, including blockchain latency, gas consumption, long-term authorization-state growth and management.

References

Bienstock A, de Castro L, Escudero D, et al (2026) Quorus: Efficient, Scalable Threshold ML-DSA Signatures from MPC. In: 35th USENIX Security Symposium (USENIX Security 26). USENIX Association, URL https://www.usenix.org/conference/usenixsecurity26/presentation/bienstock

Bloemen R, Logvinov L, Evans J (2017) EIP-712: Typed Structured Data Hashing and Signing. Ethereum Improvement Proposals, URL https://eips.ethereum.org/EIPS/eip-712, standards Track specification; accessed 7 September 2026

Boschini C, Takahashi A, Tibouchi M (2022) MuSig-L: Lattice-Based Multi-signature with Single-Round Online Phase. In: Advances in Cryptology – CRYPTO 2022. Springer Nature Switzerland, pp 276–305, https://doi.org/10.1007/978-3-031-15979-4_10, URL https://doi.org/10.1007/978-3-031-15979-4_10

Boschini C, Kaviani D, Lai RWF, et al (2025) Ringtail: Practical Two-Round Threshold Signatures from Learning with Errors. In: 2025 IEEE Symposium on Security and Privacy (SP). IEEE, pp 149–164, https://doi.org/10.1109/sp61157.2025.00070, URL https://doi.org/10.1109/sp61157.2025.00070

Chen Y (2023) DualMS: Efficient Lattice-Based Two-Round Multi-signature with Trapdoor-Free Simulation. In: Advances in Cryptology – CRYPTO 2023. Springer Nature Switzerland, pp 716–747, https://doi.org/10.1007/978-3-031-38554-4_23, URL https://doi.org/10.1007/978-3-031-38554-4_23

Damgård I, Orlandi C, Takahashi A, et al (2022) Two-Round n-out-of-n and Multi-Signatures and Trapdoor Commitment from Lattices. Journal of Cryptology 35(2). https://doi.org/10.1007/s00145-022-09425-3, URL https://doi.org/10.1007/s00145-022-09425-3, article 14

Di Francesco Maesa D, Mori P, Ricci L (2019) A blockchain based approach for the definition of auditable access control systems. Computers & Security 84:93–119. https://doi.org/10.1016/j.cose.2019.03.016

Gao L, Tang G, Jia D, et al (2026) TalonG: Bandwidth-Efficient Two-Round Threshold Signatures from Lattices. In: Advances in Cryptology – EUROCRYPT 2026. Springer Nature Switzerland, pp 94–123, https://doi.org/10.1007/978-3-032-25291-3_4, URL https://doi.org/10.1007/978-3-032-25291-3_4

Ivanov N, Yan Q, Kompalli A (2023) Txt: Real-time transaction encapsulation for ethereum smart contracts. IEEE Transactions on Dependable and Secure Computing https://doi.org/10.1109/TDSC.2023.3238312

Jiang K, Tessaro S, Wee H, et al (2026) Tweed: Adaptively Secure Lattice-Based Two-Round Threshold Signatures. In: Advances in Cryptology – EUROCRYPT 2026. Springer Nature Switzerland, pp 124–153, https://doi.org/10.1007/978-3-032-25291-3_5, URL https://doi.org/10.1007/978-3-032-25291-3_5

Lai Q, Liu FH, Lu Y, et al (2026) Scalable Two-Round n-Out-of-n and Multi-signatures from Lattices in the Quantum Random Oracle Model. In: Public-Key Cryptography – PKC 2026. Springer Nature Switzerland, pp 235–268, https://doi.org/10.1007/978-3-032-26731-3_8, URL https://doi.org/10.1007/978-3-032-26731-3_8

Liu B, Sun S, Szalachowski P (2020) Smacs: Smart contract access control service. IEEE Transactions on Dependable and Secure Computing https://doi.org/10.1109/TDSC.2020.2994065, early Access

Liu H, Wu D, Sun Y, et al (2025) Have We Solved Access Control Vulnerability Detection in Smart Contracts? A Benchmark Study. In: 2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, pp 1995–2007, https://doi.org/10.1109/ase63991.2025.00166, URL https://doi.org/10.1109/ase63991.2025.00166

Lundfall M (2020) ERC-2612: Permit Extension for EIP-20 Signed Approvals. Ethereum Improvement Proposals, URL https://eips.ethereum.org/EIPS/eip-2612, standards Track specification; accessed 7 September 2026

Nick J, Ruffing T, Seurin Y (2021) MuSig2: Simple Two-Round Schnorr Multi-signatures. In: Advances in Cryptology – CRYPTO 2021. Springer International Publishing, pp 189–221, https://doi.org/10.1007/978-3-030-84242-0_8, URL https://doi.org/10.1007/978-3-030-84242-0_8

del Pino R, Katsumata S, Maller M, et al (2024) Threshold Raccoon: Practical Threshold Signatures from Standard Lattice Assumptions. In: Advances in Cryptology – EUROCRYPT 2024. Springer Nature Switzerland, pp 219–248, https://doi.org/10.1007/978-3-031-58723-8_8, URL https://doi.org/10.1007/978-3-031-58723-8_8

Ruaro N, Gritti F, Meng D, et al (2025) Approve Once, Regret Forever: On the Exploitation of Ethereum's Approve-TransferFrom Ecosystem. In: 34th USENIX Security Symposium (USENIX Security 25). USENIX Association, pp 1281–1298, URL https://www.usenix.org/conference/usenixsecurity25/presentation/ruaro

Wang Z, Chen J, Lin Z, et al (2026) One Signature, Multiple Payments: Demystifying and Detecting Signature Replay Vulnerabilities in Smart Contracts. In: 2026 IEEE/ACM International Conference on Software Engineering (ICSE), URL https://conf.researchr.org/details/icse-2026/icse-2026-research-track/156/One-Signature-Multiple-Payments-Demystifying-and-Detecting-Signature-Replay-Vulnera

Xing X, Liu Y, Wu Q, et al (2025) Multi-committee ABE based decentralized access control with sharding blockchain for Web 3.0. IEEE Transactions on Dependable and

Secure Computing 22(3):2533–2549. https://doi.org/10.1109/TDSC.2024.3520121