



Deposited via The University of Sheffield.

White Rose Research Online URL for this paper:

<https://eprints.whiterose.ac.uk/id/eprint/247159/>

Version: Accepted Version

Proceedings Paper:

Khanji, S., Abdolmaleki, B., Clark, J. et al. (2026) Zero-knowledge proof of progress: secure multi-phase capture-the-flag competitions. In: 2026 IEEE 39th Computer Security Foundations Symposium (CSF). 2026 IEEE 39th Computer Security Foundations Symposium (CSF), 26-29 Jul 2026, Lisboa, Portugal. Institute of Electrical and Electronics Engineers (IEEE), pp. 253-268. ISBN: 9798319518071. ISSN: 1940-1434. EISSN: 2374-8303.

<https://doi.org/10.1109/csf68417.2026.00017>

© 2026 The Authors. Except as otherwise noted, this author-accepted version of a conference paper published in 2026 IEEE 39th Computer Security Foundations Symposium (CSF) is made available via the University of Sheffield Research Publications and Copyright Policy under the terms of the Creative Commons Attribution 4.0 International License (CC-BY 4.0), which permits unrestricted use, distribution and reproduction in any medium, provided the original work is properly cited. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>

Reuse

This article is distributed under the terms of the Creative Commons Attribution (CC BY) licence. This licence allows you to distribute, remix, tweak, and build upon the work, even commercially, as long as you credit the authors for the original work. More information and the full terms of the licence here: <https://creativecommons.org/licenses/>

Takedown

If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

Zero-Knowledge Proof of Progress: Secure Multi-Phase Capture-the-Flag Competitions

Salam Khanji, Behzad Abdolmaleki, John Clark, Aryan Pasikhani
University of Sheffield, UK

{sirkhanj1, behzad.abdolmaleki, john.clark, aryan.pasikhani}@sheffield.ac.uk

Abstract—Existing Capture-the-Flag (CTF) platforms trust a single organizer, offer limited auditability, and are vulnerable to infrastructure-level manipulation. We propose zk-MPSFV, a zk-SNARK-based, multi-phase sub-flag verification scheme that replaces centralized scoring with an on-chain, zero-knowledge, publicly verifiable scoreboard. Challenges are decomposed into sub-challenges arranged as a directed acyclic graph (DAG): a team unlocks the next step only after proving completion of all parent nodes. Sub-flags and decryption keys are jointly generated by n organizers and released via an off-chain (t, n) Shamir-BLS threshold signature produced through multi-party computation (MPC), preventing any single organizer from leaking or altering keys. Teams submit zk-PLONK proofs that the contract verifies, timestamps, and records immutably. Under standard assumptions (collision-resistant hashing, SNARK soundness/zero-knowledge, IND-CCA2 ECIES, and at least t honest organizers), we prove that zk-MPSFV achieves the stated security goals, including DAG-gated progress, anti-replay, and threshold-robust organizer security, while out-of-band flag sharing remains out of scope. On a three-organizer testbed with 30 simulated teams, setup costs 0.45 ms per sub-flag, proof generation averages 5.34 s on an 8-core system, and on-chain verification costs $\approx 170k$ L2 gas on zkSync Era with a median fee of 1.33×10^{-6} ETH (about \$0.0046 at \$3,435/ETH). Stress replays sustain ≈ 7 proof transactions/s up to 5000 proofs; extrapolating to 50,000 proofs (1000 teams $\times$ 50 submissions) yields ≈ 0.0665 ETH (about \$200–\$228) and ≈ 2 hours of settlement time. Overall, zk-MPSFV is practical for small- to mid-scale, audit-ready progression CTFs.

Index Terms—zero-knowledge, capture-the-flag, blockchain, security

I. INTRODUCTION

A global cybersecurity skills gap, estimated at 4.8 million professionals, has increased demand for effective training and assessment [1]. Capture-the-Flag (CTF) competitions address this need by simulating real-world security tasks [2], but their integrity depends on authentic flag capture, since competitors may cheat through flag sharing or covert collaboration [3]. CTF infrastructure is also vulnerable to denial-of-service and exploitation attempts [4], motivating stronger guarantees for flag confidentiality and authenticity [5]. Zero-knowledge proofs (ZKPs), which enable verifiable computation without revealing sensitive information, have been applied in authentication [20], blockchain verification [38], and secure machine learning [21, 22]. While NIZKCTF [23] introduces zero-knowledge flag verification, it does not enforce multi-phase progression, fine-grained sub-flag workflows, or on-chain public verifiability, limiting suitability for modern multi-stage CTFs. More broadly, improving CTF credibility benefits from reproducibility-oriented evidence and transparent

audit trails that let observers and organizers assess claimed progress [6]. We propose zk-MPSFV, a multi-phase sub-flag verification scheme that models each master challenge S as sub-challenges $(S_1, \dots, S_n)$ with corresponding sub-flags $(F_1, \dots, F_n)$ arranged as a DAG. This supports stepwise verification of intermediate milestones (e.g., reconnaissance and exploitation) via commitments and zk-SNARK “micro proofs,” keeping flags secret while making partial progress auditable. zk-MPSFV also supports multi-organizer events by distributing responsibilities and reducing reliance on any single trusted entity. Although most CTFs are run honestly, community reports raise concerns about perceived unfair advantages, alleged flag leakage, and opaque scoring or challenge handling [7, 8, 9, 10]. Our goal is not to claim organizer collusion is widespread, but to remove the assumption of a fully trusted organizer by using cryptographic mechanisms, including multi-party computation (MPC), that better match the operational realities of modern CTFs and enable secure, scalable coordination among organizers [11].

A. Related Work

Most CTF platforms still validate plaintext flags through centralized portals and rely on operational anti-cheating measures such as per-team flag randomization, obfuscation, and timed release [2, 3, 5]. These mechanisms offer limited auditability and remain vulnerable to replay, tampering, and collusion [4, 6], so fairness largely depends on trust in the platform and organizers. Existing platforms also use ad-hoc integrity and scoring schemes: FBCTF checks a team-specific file via a scoring agent [12], RootTheBox rewards sustained control through bots [13, 14], FBCTF uses decrementing bonus points [13], RootTheBox uses periodic bot rewards [14], FBCTF and CTFd support export/import backups [12, 15], and Mellivora provides controls such as attempt limits, hint penalties, and per-challenge timeframes [16]. To avoid plaintext flags, NIZKCTF [23] replaces submissions with non-interactive zero-knowledge proofs of flag knowledge and enables public verifiability through Git-based repositories. However, it does not support on-chain verification, dynamic scoring, fine-grained multi-step workflows, native replay protection, or DAG-gated sequential solving, limiting its suitability for modern multi-phase CTFs [23]. In NIZKCTF, teams prove simultaneous knowledge of a team secret key sk_t for identity pk_t and a flag-derived secret sk_c for pk_c ; although this Ed25519-based design binds proofs to identities,

it provides neither organizer thresholding nor distributed trust, leaves plaintext flags accessible to a central maintainer, and relies on Git logs rather than decentralized cryptographic audit trails [23]. The authors report thousands of commits during Pwn2Win 2017, but provide no systematic evidence of CI scalability and note undocumented GitHub merge API issues under load [23]. Large CTFs also increasingly depend on multiple organizers, which improves robustness but complicates trust and result aggregation. DEFCON CTF is co-organized by expert teams such as PPP and Dragon Sector [17], Google CTF combines internal teams and external contributors [18], and ECSC coordinates national agencies across Europe [19], highlighting practical challenges in trust, secure aggregation, and fair scoring across independently managed components. In light of these issues, we ask: (1) *Can a logically sound and practical system model enable stepwise verification of DAG-based challenge-solving progress while preserving flag confidentiality and trust across multiple organizers?* (2) *Can a cryptographic construction in this model provide partial stepwise reproducibility, immutable auditability, and verifiability?*

B. Our Contributions

We introduce zk-MPSFV, a multi-phase zk-SNARK-based verification scheme for Jeopardy-style CTFs that keeps sub-flags confidential, enforces DAG-ordered progress, and records verifiable partial solutions on-chain without relying on a single trusted organizer. Our main contributions are: (1) *DAG-gated, multi-phase sub-flag verification*. A master challenge S is decomposed into sub-challenges $(S_1, \dots, S_n)$ with sub-flags F_i and symmetric keys K_{S_i} ; dependencies form a DAG (Figure 1) so a node unlocks only after its prerequisites are proven, while accepted proofs are logged on-chain without revealing F_i . (2) *Team-bound proofs with circuit-enforced gating*. Unlike NIZKCTF [23], zk-MPSFV embeds dependency checks in the proof circuit and binds each proof to a team identity via per-team commitments hashing the hidden sub-flag with the team public key, preventing cross-team replay. (3) *Threshold-MPC organizer trust distribution*. Sub-flags are secret-shared and reconstructed only when eligible using Shamir (t, n) sharing [24] and MPC, reducing leakage or tampering risk from any single organizer. (4) *On-chain verification, scoring, and audit trail*. A smart contract verifies zk-PLONK submissions, timestamps accepted sub-flags (enabling first-blood tie-breaks that rank teams with equal points by awarding the higher rank to the team whose valid submission for a given sub-challenge was accepted earliest) and maintains an immutable public audit trail without trusted backends.

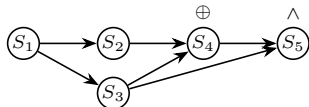


Fig. 1: Example DAG with five sub-challenges; edges denote prerequisites. $\oplus$ indicates an OR-gated unlock (for S_4), and $\wedge$ indicates an AND-gated unlock (for S_5).

C. zk-MPSFV: Technical Overview

zk-MPSFV targets three goals: (i) each intermediate step is verifiable in zero knowledge, (ii) teams advance only along an organizer-defined dependency graph, and (iii) no single organizer can leak, alter, or withhold secret material. Although easy to state, this is *not* a naive composition: the protocol must simultaneously bind proofs to team identities (to prevent cross-team replay), enforce DAG prerequisites without a trusted backend, and release per-team decryption keys without a single trusted key-holder. Most cryptography runs off-chain; the blockchain records only succinct proofs, timestamps, and threshold signatures. Setup begins with an MPC ceremony in which n organizers sample fresh secrets for each sub-flag (and the final master flag) and encode sub-challenges as a DAG. Secrets are split using Shamir (t, n) sharing so reconstruction requires at least t honest organizers [24]. For each team, organizers publish on-chain *team-bound commitments* that hash each sub-flag with the team public key, defining the public statements proved later. We instantiate proofs with PLONK [25]; keys are embedded in the contract and verification is efficient on EVM chains [38]. Challenge binaries are symmetrically encrypted and stored off-chain (e.g., IPFS), and unlock keys are released only upon authorization (since a ZK proof alone does not prevent reuse once a flag is shared, and encryption alone reintroduces a trusted key-holder unless release is threshold-authorized). When a team learns a sub-flag, it submits a PLONK proof of knowledge of the committed preimage (without revealing the flag) together with an EIP-712 typed-data signature to bind the submission to the team and prevent replay. On roll-ups such as zkSync Era [45], these submissions are inexpensive. The contract verifies the signature and proof, checks DAG prerequisites, and timestamps acceptance (for first-blood tie-breaks). It then emits a key-eligibility event; off-chain, at least t organizers aggregate BLS partial signatures into a threshold signature [27], which the contract verifies via a single pairing. If valid, the contract publishes the corresponding unlock key as an ECIES ciphertext under the solver's public key [34], preserving key privacy and preventing unilateral leakage by any organizer. After all DAG nodes are proven, the team proves completion of the master challenge; all proofs, timestamps, and organizer signatures remain on-chain, enabling public replay and scoreboard reconstruction without trusting a backend. This overcomes limitations of traditional CTFd [42] and Git-based NIZKCTF deployments [23]. We formalize assumptions and notation in section II, and detail protocol phases in section IV.

D. State-of-the-art Comparison

NIZKCTF [23] already achieves its main privacy goal, where no one can extract a flag, because the EUF-CMA (Existential Unforgeability under Chosen Message Attack) signature proof provides key-retrieval resistance. The gap we address is not stronger secrecy of the flag itself, but everything that surrounds a submission: enforcing arbitrary solve dependencies, ensuring that only the solver receives the next-stage

TABLE I: Comparison of zk-MPSFV, NIZKCTF, and Legacy CTF Platforms

Feature	zk-MPSFV	NIZKCTF	Legacy (CTFd)
Proof mechanism	zk-SNARK (PLONK)	Sig. PoK	Plain flag
Dependency enforcement	✓ DAG (on-chain)	✗	✗
Team-bound commitment	✓ $H(F_i pk_T)$	✗	✗
Replay protection	✓ nonce+ledger	Limited	Limited
Flag confidentiality	✓	✓	✗
Constraint expressiveness	✓	✗	✗
Key privacy (K_i)	✓ enc. to team	✗	n/a
Multi-organizer trust	✓ (t, n) MPC	✗	✗
Submission method	Smart contract	Git+CI	HTTPS POST
Auditability	On-chain	Git history	Central DB
Scoring logic	On-chain	CI script	Server logic

key, and removing single-organizer trust. zk-MPSFV therefore adds three capabilities that signatures plus Git cannot express in a publicly-verifiable way: (i) on-chain checking of a DAG of prerequisites, (ii) per-team, hash-bound commitments that block cross-team replay, and (iii) a (t, n) multi-organizer setup that prevents any single organizer from leaking or altering sub-flags. Table I compares against (i) NIZKCTF, the only prior cryptographic flag-verification platform, and (ii) CTFd [42], the most widely deployed traditional framework. This highlights the gap between current practice and cryptographically verifiable solutions.

II. BACKGROUND AND CRYPTOGRAPHIC PRIMITIVES

In this section, we provide a brief introduction to the background related to zk-MPSFV and the cryptographic primitives used in our scheme. Symbols and notations are summarized in Table II.

Non-Interactive Zero-Knowledge (NIZK) Proofs: Let $\mathcal{R}$ be a polynomial-time decidable relation over statement-witness pairs (x, w) and $\mathcal{L}_{\mathcal{R}} = \{x \mid \exists w : (x, w) \in \mathcal{R}\}$. A CRS-based NIZK proof system Ψ consists of PPT algorithms: (i) $K_{\text{crs}}(\lambda) \rightarrow (\text{crs}, \text{td})$ (setup), (ii) $P(\text{crs}, x, w) \rightarrow \pi$ (prove), (iii) $V(\text{crs}, x, \pi) \rightarrow \{0, 1\}$ (verify), and (iv) $\text{Sim}(\text{crs}, \text{td}, x) \rightarrow \pi$ (simulate). It satisfies: *completeness* (if $(x, w) \in \mathcal{R}$ then V accepts P w.h.p.), *zero-knowledge* (real proofs are indistinguishable from Sim outputs for true statements), *soundness* (no PPT adversary can produce an accepting proof for $x \notin \mathcal{L}_{\mathcal{R}}$ except with negligible probability), and *public verifiability* (any party given crs can run $V(\text{crs}, x, \pi)$, enabling on-chain verification).

Digital Signature: Digital signatures provide authenticity and non-repudiation in our scheme: only the holder of the private key sk_{team} can produce a valid signature σ on a submission message m , preventing impersonation. A signature scheme $\Sigma = (\text{Gen}, \text{Sign}, \text{Verify})$ generates keys $(pk_{\text{team}}, sk_{\text{team}}) \leftarrow \text{Gen}(\lambda)$, signs $\sigma \leftarrow \text{Sign}(sk_{\text{team}}, m)$, and verifies $\text{Verify}(pk_{\text{team}}, m, \sigma) \in \{\text{true}, \text{false}\}$. We assume existential unforgeability under chosen-message attacks (EUF-CMA) [35]. To prevent replay, each signed submission includes a fresh, strictly increasing nonce (a “number used once”), so previously accepted transactions cannot be reused.

Commitment Schemes: A commitment scheme $\Pi = (\text{KGen}, \text{Com}, \text{Decom})$ classically provides two core proper-

TABLE II: Symbols and Notations

Symbol / Notation	Description
λ	Security parameter
$\mathcal{G} = (V, E)$	DAG of the challenge structure with E set of edges between V set of vertices
$P(j)$	Parent set of node j in $\mathcal{G}$
F_j	Sub-flag at node j
$E(S_j) = E_{K_j}(S_j)$	Ciphertext of sub-challenge S_j under key K_j
K_j	Symmetric key for decrypting S_j
F_M	Final master flag
PK, VK	SNARK proving and verification keys
$\pi_j = \text{Prove}(\text{PK}, F_j, C_j)$	SNARK proof for sub-flag F_j
$\text{Verify}(\text{VK}, \pi_j, C_j)$	On-chain verification of π_j
n, t	n organizers and t threshold for reconstruction
$\text{share}_{j,n}$	Shamir share of F_j held by organizer n
$\text{SigAgg}_t(\cdot)$	BLS threshold signature from t organizers
teamID	Unique team identifier
$C_{j\text{teamID}} = \text{Hash}(F_j pk_{\text{teamID}})$	team-bound commitment of F_j
sk_{teamID}	Private secret key of the team
$m_j = (j, \pi_j, \text{nonce}, \text{teamID})$	EIP-712 message structure
$\sigma = \text{Sign}(sk_{\text{teamID}}, m_j)$	Team’s signature on submission
nonce	Monotonic counter for replay protection
$\text{teamProgress}[\text{teamID}]$	Set of solved node indices stored on-chain
$\text{teamPoints}[\text{teamID}]$	Set of team score stored on-chain
$\text{solveTime}[\text{teamID}][j]$	On-chain time-stamp for solving sub-flag j
$\text{basePointsPerFlag} = X$	Score awarded per sub-flag stored on-chain
$\text{finalBonus} = Y$	Final bonus for full challenge solve stored on-chain
$\text{submitSubFlagProof}$	Submission function in the Scoreboard contract
releaseKey	Threshold signature key-release function in the Scoreboard contract
$\text{KeyEligibilityRequest}$	Scoreboard contract event to provide the decryption key for the next sub-challenge by t -threshold organizers
ReleasedKey	Scoreboard contract event to release the cipher of the decryption key for the next sub-challenge for a team
$\text{MasterFlagRevealed}$	Final Scoreboard contract event marking the completion of a challenge for a team

ties: *binding* and *hiding*. In its simplest form, the scheme operates as follows: **Key Generation:** $\text{KGen}(\lambda) \rightarrow ck$: Given a security parameter λ , this generates a public parameter ck (sometimes called a “commitment key”). **Commitment:** $(c, \delta) \leftarrow \text{Com}_{ck}(m, r)$: Given ck , a message m from some message space M , and randomness r , produce a commitment c together with opening information δ . **Decommitment:** $m' \leftarrow \text{Decom}(ck, c, \delta)$: Given ck , a commitment c , and an opening δ , recover the original message m' . If c does not correctly open, return $\perp$. Consequently, we say Π is *binding* if an adversary cannot produce two distinct openings δ, δ' that cause the same commitment c to open to two different messages $m \neq m'$. It is *hiding* if observing c alone does not reveal anything about m .

Typed-Data Signature (EIP-712): EIP-712 standardizes *Typed Structured Data Hashing and Signing* for Ethereum accounts.¹ Instead of signing an opaque 32-byte hash (as in `eth_sign`), the signer deterministically encodes a human-readable typed message, hashes it together with a *domain separator*, and signs the resulting digest using the account’s

¹<https://eips.ethereum.org/EIPS/eip-712>

standard ECDSA key over secp256k1 [46]. The domain separator binds signatures to a specific contract address, chain ID, and version (optionally also an application name), reducing cross-contract/network replay and signature-phishing. Replay protection is strengthened by including call-specific fields (e.g., a monotonically increasing nonce or timestamp) in the typed data. On-chain, the contract reconstructs the domain separator and type-hash, recomputes the EIP-712 digest, and recovers the signer via `ecrecover`; any mismatch causes verification to fail. EIP-712 is widely supported in Ethereum tooling (e.g., *ethers.js* and OpenZeppelin’s *EIP712* helper) and is commonly used for secure off-chain→on-chain authentication.

PLONK zk-SNARK: We instantiate Ψ with PLONK [25], a zk-SNARK with a universal setup, i.e., a single (pk, vk) supports circuits up to a fixed size bound. We use PLONK to prove knowledge of committed sub-flags without revealing them.

Threshold secret sharing & signatures: Each sub-flag F_j is split via Shamir (t, n) secret sharing into shares $\{s_{j,1}, \dots, s_{j,n}\} \leftarrow \text{Share}(F_j, t, n)$, so reconstruction requires at least t organizers. Organizers authorize key release via a threshold BLS signature scheme [26], where t partial signatures are aggregated into $\sigma = \text{AggSign}(\{\sigma_k\}_{k \in Q})$ and verified on-chain before releasing ciphertexts. **Key-privacy encryption:** We use an IND-CCA2 secure ECIES scheme over secp256k1 [34]; the contract publishes $C_{k,T} := \text{Enc}_{pk_T}(K_k)$ so only the intended team holding sk_T can recover K_k .

Blockchain and Smart Contracts Assumption: We assume on-chain verification and scoring execute as deployed on Ethereum, leveraging decentralized consensus and an immutable ledger [36]. Ethereum blocks confirm on the order of 12–15s on mainnet [37], and deployed contracts are immutable absent explicit upgrade mechanisms, yielding tamper-resistant verification and scoring. A zk-SNARK verifier contract (e.g., `Verifier.sol` generated by *circom/snarkJS*) checks proofs on-chain, typically costing hundreds of thousands of gas units (the unit of computational work in Ethereum for executing opcodes/operations). per proof [38]. Submissions can be authenticated using EIP-712 typed-data signatures [39], with the contract recovering the signer via `ecrecover` and enforcing nonce-based replay protection.

III. THREAT MODEL AND SECURITY GOALS

A. Threat Model

Our setting is a Jeopardy-style CTF managed by n co-organizers who run a Shamir-MPC ceremony [24] off-chain and a PLONK-verifying *Scoreboard* contract on Ethereum. All teams interact with the contract through EIP-712 messages; every key-release event and zero-knowledge proof is permanently logged on chain. We identify three adversary types, each strictly stronger than the previous one:

- $[\mathcal{A}_1]$ The Type 1 (Katz–Lindell stand-alone) [35] network adversary that controls the peer-to-peer network and mem-pool (the pool of pending, not-yet-mined transactions maintained

by blockchain nodes). It can front-run, delay, drop, re-order or duplicate any transaction before it becomes final. As soon as a Layer-1 block is confirmed k times (such as $k=64$ on Ethereum PoS, ≈ 12 min [37]) the adversary can no longer revert it. Likewise, Arbitrum-style roll-up provides cheaper L2 interactions with guaranteed forced inclusion on L1 [43] where a roll-up batch is final once posted to L1, even if the sequencer is hostile. $\mathcal{A}_1$ is PPT and therefore respects all cryptographic assumptions as in the collision resistance of Hash, IND-CCA2 security of ECIES [34], EUF-CMA security of threshold BLS [27], and the zero-knowledge/knowledge-soundness of PLONK.

- $[\mathcal{A}_2]$ The Type 2 (Colluding-Team/Privacy) adversary that embeds $\mathcal{A}_1$ and additionally corrupts an arbitrary subset of teams, learning their secret keys sk_T and internal state. Its objective is to violate key privacy, link ciphertexts or proofs across honest teams, or to replay an honest proof under a different identity. $\mathcal{A}_2$ cannot compromise organizers, forge BLS aggregate signatures without t partials, or decrypt a ciphertext encrypted under a public key it does not control.

- $[\mathcal{A}_3]$ Finally, we consider a Type 3 (Organizer-Corruption) adversary that controls $\mathcal{A}_1$ and $\mathcal{A}_2$ and may adaptively corrupt up to $n-t$ organizers, obtaining their Shamir shares and BLS partial keys. It can refuse to co-sign releases, attempt to learn flags prematurely, or publish conflicting key material. If at least t organizers remain honest and online, Shamir hiding plus BLS unforgeability prevent $\mathcal{A}_3$ from reconstructing any secret or forging a threshold signature.

Across these types, the adversary may front-run or re-order any pending transaction before L1 finality, and submit unlimited proof or key-release transactions of its own. The adversary may also corrupt teams or up to $n-t$ organizers at any time, and delay partial organizer signatures—yet it cannot break the assumed cryptographic primitives or revert a block that has achieved k L1 confirmations. The scheme guarantees liveness provided that (i) at least one Ethereum full node remains reachable, (ii) the forced-inclusion window of the roll-up is honoured, and (iii) at least t organizers respond to key-release requests. Sustained denial-of-service on IPFS gateways or RPC endpoints, and subsecond (“real-time”) leaderboard display, are outside our scope. zk-MPSFV binds each accepted sub-flag proof to a team-specific commitment $C_{j,teamID} = \text{Hash}(F_j \parallel pk_{teamID})$ and requires both an EIP-712 signature under pk_{teamID} and a one-time-use flag `proofHashUsed[Cj,teamID]`. Thus, a proof that was accepted for team A cannot be replayed on-chain as a valid submission for team B . Competitor collusion poses a significant integrity threat in online qualifiers and prize events, as it can impact rankings, qualification cutoffs, and prize distribution. Competitors may collude for reasons such as prize incentives [30, 5], team size limits [5, 33, 32] or cross-team alliances [29, 5], and prior measurements and community reports document informal flag sharing, solution trading and “multi-team” coaching behaviour in CTF [5, 29, 30, 31, 32]. However, the scheme does *not* try to prevent out-of-band collusion in which one team discloses the underlying witness

F_j (or a derived exploit or write-up) to another team: once F_j is known, any party can generate a valid proof for its own commitment. This limitation is inherent to proof-of-knowledge-based scoring: the proof certifies *knowledge* of F_j but not the *provenance* of that knowledge [28]. Practical mitigations such as per-team challenge instances or automatic variant generation [5] can reduce direct flag copying, but they are orthogonal to our scheme design, and we treat them as out of scope. Under Types $\mathcal{A}_1$ – $\mathcal{A}_3$, zk-MPSFV achieves the ten security goals detailed in subsection III-B and formalized in subsection IV-A: sub-flag zero-knowledge, soundness, completeness, public verifiability, DAG-gated progress, chain-of-proof consistency, proof-of-completion, key-privacy, anti-replay, and threshold-robust organizer security. Theorem 1 and its proof in subsection A demonstrate that zk-MPSFV attains all ten goals except with negligible probability $\text{negl}(\lambda)$ under these conditions.

B. Security Goals

zk-MPSFV provides: (i) *sub-flag zero-knowledge* (proofs reveal only knowledge of F_j), (ii) *soundness* and (iii) *completeness* for sub-flag proofs, (iv) *public verifiability* via an on-chain log of proofs, timestamps, and threshold signatures, (v) *DAG-gated progress* (a key K_j is released only after all parents verify and a threshold signature SigAgg_t validates), (vi) *chain-of-proof consistency* through EIP-712 authenticated, team-bound commitments $C_{j,\text{teamID}} = H(F_j \parallel pk_{\text{teamID}})$, (vii) *proof-of-completion* (final bonus iff the same team proves all DAG sub-flags and the master flag F_M), (viii) *key privacy* (keys are released only as ECIES ciphertexts for the intended team), (ix) *anti-replay* (accepted $C_{j,\text{teamID}}$ are one-time spend), and (x) *threshold-robust organizer security* under (t, n) sharing assuming at least t organizers remain honest and online.

IV. ZK-MPSFV DESIGN

Consider a Jeopardy-CTF where organizers prepare a master challenge M as a DAG whose vertices $S_1, \dots, S_N$ are sub-challenges. For each vertex j , the n organizers jointly generate a secret sub-flag F_j via an (t, n) MPC ceremony, where t is the threshold of organizers required to reconstruct a secret or produce an aggregated threshold signature SigAgg_t . Figure 2 illustrates zk-MPSFV, which proceeds in four phases described below:

Phase 1: System Setup: This phase is demonstrated in algorithm 1 where it declares the global parameters required to ensure that the blockchain and the organizers possess all the data required to begin the competition. As depicted in lines (1–5), the MPC parameters (n, t) are set to map the total number of organizers (n), and at least t honest organizers (threshold) are required to complete the MPC procedure. Line 4 defines the DAG vertices of the sub-flag dependencies for each master challenge, whereas line 5 handles the registry of the participating teams by storing each team’s public key pk_{teamID} . The MPC generation of sub-flags and shares is illustrated in lines (6–10), where

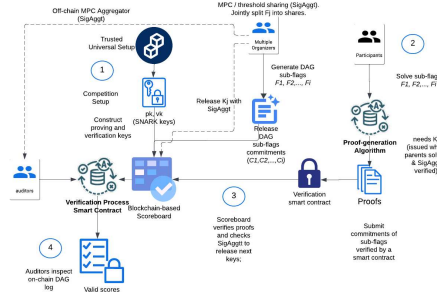


Fig. 2: High-level zk-MPSFV architecture: PLONK setup (PK, VK); organizers MPC-split sub-flags and release keys via SigAgg_t ; teams submit proofs π_j checked against the DAG; all proofs/timestamps are on-chain, and proving all nodes yields the master flag F_M and bonus.

for every vertex j the organizers jointly sample a fresh secret F_j , such as a 256-bit string, inside an MPC. They immediately split it into n Shamir shares with threshold t ; $\text{share}_{j,k}$ is kept private by organizer_k . Organizers then jointly publish team-bound commitments of the sub-flags as a DAG in lines (11–15). For every team T and every vertex j , compute a commitment that binds the sub-flag to that team’s public key: $C_{j,\text{teamID}} = \text{Hash}(F_j \parallel pk_{\text{teamID}})$. All such commitments are posted on-chain, making them immutable and publicly auditable. The same MPC is also used to generate a global master flag F_M and secretly share it in (lines 16–18). Its commitment, $C_{M,\text{teamID}} = \text{Hash}(F_M \parallel pk_{\text{teamID}})$, is computed by the organizers’ MPC as the Keccak-256 (a cryptographic hash function assumed to be collision-resistant and preimage-resistant) and published on-chain so that, later, validating its proof constitutes proof-of-completion (lines 19–20). To enforce DAG-gated-progress (line 21): for each vertex j , the organizers derive a symmetric key K_j using threshold cryptography, encrypt the puzzle S_j as $E_{K_j}(S_j)$, and upload every ciphertext to IPFS (lines 22–26). Lines (27–32) enable every team to start solving the root sub-flags by initialising for every team T $\text{teamProgress}[\text{teamID}]$ to $\emptyset$. Then, for every root vertex $P(j) = \emptyset$, immediately invoke $\text{releaseKey}(j, \text{teamID}, \text{sigAgg})$ to emit K_j only after a valid aggregated organizer signature SigAgg_t . zk-MPSFV prevents the replay attack by creating a Boolean map proofHashUsed indexed by the team-bound commitment $C_{j,\text{teamID}}$ and setting all entries to false; when a proof is accepted, the corresponding entry is flipped to true, preventing replay (lines 33–37). Finally, the scheme runs the universal SNARK setup (PLONK) [38] to obtain the proving key PK and verification key VK and publish VK on chain (lines 38–40), and set on-chain the public scoring parameters: points per sub-flag X and the final-bonus value Y .

Phase 2: Sub-flag Proof Generation and Submission. As depicted in algorithm 2 lines (1–10), a team iterates (off-chain) over every sub-challenge index j currently unlocked (line 2). If the decrypt key K_j is available and all parent indices $P(j)$ are recorded in the team’s progress set (line 3), the player decrypts puzzle S_j , solves it, and obtains the hidden sub-flag

Algorithm 1: System Setup

```

1 Phase 1: System Setup
2 //Global parameters
3  $(n, t)$  // number of organizers and threshold
4  $\mathcal{G} = (V, E)$  // DAG over indices  $V = \{1, \dots, N\}$ 
5 AllTeams // list of registered teams each with public and secret keys
    $pk_{teamID}, sk_{teamID}$  used for EIP-712 signing and ECIES decryption

6 //MPC generation of sub-flags and shares
7 for  $j \in V$  do
8   jointly sample  $F_j$  under an MPC protocol;
9    $(share_{j,1}, \dots, share_{j,n}) \leftarrow \text{ShamirSplit}(F_j, t, n)$ ;
10  each organizer  $k$  stores  $share_{j,k}$  securely;

11 //Publish team-bound commitments storedCommit[j][teamID] on-chain
12 foreach  $teamID \in \text{AllTeams}$  do
13   for  $j \in V$  do
14      $C_{j,teamID} \leftarrow \text{Hash}(F_j \parallel pk_{teamID})$ ;
15     storedCommit[j][teamID] =  $C_{j,teamID}$ ;

16 //Master flag and Commitment (also secret-shared)
17 jointly generate  $F_M$  via MPC;
18 split into  $(share_{M,1}, \dots, share_{M,n})$ ;
19  $C_{M,teamID} \leftarrow \text{Hash}(M \parallel pk_{teamID})$  via MPC;
20 publish  $C_{M,teamID}$  on chain;
21 //Encrypt puzzles
22 for  $j \in V$  do
23   organizers derive  $K_j$  (threshold operation);
24    $E(S_j) \leftarrow E_{K_j}(S_j)$ ;

25 //  $K_j$  is stored only in organiser MPC store; never on chain;
26 upload  $\{E(S_j)\}_{j \in V}$  to IPFS;
27 //Initial key release for root nodes
28 foreach  $teamID \in \text{AllTeams}$  do
29   teamProgress[teamID]  $\leftarrow \emptyset$ ;
30   foreach  $j \in V$  where  $P(j) = \emptyset$  do
31      $sigAgg = \text{sigAgg}_t(j, teamID)$  call
       releaseKey( $j, teamID, sigAgg$ );
32     // A function in algorithm 2 (lines 21-25)

33 //One-time-use guard initialisation
34 foreach  $teamID \in \text{AllTeams}$  do
35   for  $j \in V$  do
36     proofHashUsed[ $C_{j,teamID}$ ]  $\leftarrow \text{false}$ ;
37     // A Boolean map stored in the Scoreboard contract state

38 //SNARK setup and scoring
39  $(PK, VK) \leftarrow \text{Setup}(\lambda)$ ; publish VK on chain;
40 set basePointsPerFlag =  $X$ , finalBonus =  $Y$ ;

```

F_j (line 4). If the puzzle is solved correctly, then the contract-stored commitment specific to the team, $C_{j,T}$ (line 6), is fed into the SNARK prover to generate its zero-knowledge proof π_j (line 7). The team packages the tuple $(j, \pi_j, \text{nonce}, \text{teamID})$ into a typed message m_j [41], signs it with their private key sk_{teamID} (lines 8–9), and submits the signed payload to the blockchain (line 10) [39].

Phase 3: On-chain Verification and Points Assignment. Upon receiving the team’s submission from phase 2 (algorithm 2 line 12), the contract first validates the EIP-712 signature and nonce (line 13), rejecting submissions if any parent index remains unsolved for the submitting team (line 14). It retrieves the appropriate commitment $C_{j,teamID}$ (line 15), ensuring it has not been previously spent; otherwise, it rejects the submission. The proof π_j is verified using $\text{Verify}(VK, \pi_j, C_{j,teamID})$ (line 16). When valid, the contract marks the commitment as used, updates the team’s progress set with j , credits *basePointsPerFlag*, and records a timestamp (line 17). For every child node k whose parents are now fully proven, the contract emits a

KeyEligibilityRequest($k, teamID$) tuple (lines 18). Then, an off-chain MPC service gathers t organizer signatures, aggregates them into $\sigma_{Agg} = \text{SigAgg}_t(k, teamID)$ (lines 19–20), and calls the function *releaseKey*($k, teamID, \sigma_{Agg}$). The contract’s *releaseKey* function (lines 21–25) verifies this aggregated signature (*VerifyAgg*), retrieves the cleartext key K_k from organizer storage, encrypts it for the team’s public key via ECIES, and emits *ReleasedKey*($teamID, k, C_k^{enc}$) tuple, ensuring only the intended team can decrypt the subsequent puzzle [38].

Phase 4: Master Flag and Bonus. As shown in algorithm 2, the contract first checks if a team has proven every vertex in the DAG, and all vertices are recorded in *teamProgress*[$teamID$]. When the team already captures the master flag F_M (lines 26-27), the team retrieves back its stored team-bound commitment $C_{M,teamID}$ (line 28), to feed to the SNARK prover so that to generate its proofs π_M , then the team constructs the typed message $(\pi_M, M, \text{nonce}, \text{teamID})$, signs it with their sk_{teamID} , and submits M along with the signature (lines 29–32). The contract verifies the signature, nonce, and the proof (lines 33–34). If valid, then the contract awards the *finalBonus* to the team’s score and emits *MasterFlagRevealed*($teamID, C_{M,teamID}$) tuple, which permanently records the team’s completion on-chain (lines 35–36).

A. Security Definitions and Proofs

Definition 1 (Sub-flag Zero-Knowledge). *Let F_i be a sub-flag and let pk_{teamID} be the public key of team $teamID$. Define the commitment $C_{i,teamID} = \text{Hash}(F_i \parallel pk_{teamID})$. Let $\pi_i \leftarrow \text{Prove}(PK, F_i, C_{i,teamID})$ be a proof produced with witness F_i , and let $\text{Verify}(VK, \pi_i, C_{i,teamID}) \rightarrow \{0, 1\}$ be the corresponding verifier. We say the scheme is zero-knowledge if the proof π_i reveals no information about F_i (or any hidden data) beyond the fact that the prover knows a pre-image of $C_{i,teamID}$. Formally, there exists a probabilistic polynomial-time simulator $\text{Sim}(td, C_{i,teamID}) \rightarrow \pi^*$, where td is the trapdoor output by the trusted or universal setup, such that for every PPT distinguisher D*

$$|\Pr[D(\pi_i) = 1] - \Pr[D(\pi^*) = 1]| \leq \text{negl}(\lambda).$$

Definition 2 (Soundness). *Let $\{F_1, \dots, F_N\}$ be the sub-flags and let pk_{teamID} be the public key registered for team $teamID$. For every index i define the team-specific commitment $C_{i,teamID} = \text{Hash}(F_i \parallel pk_{teamID})$. Consider any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ that outputs an index i and a proof π ; let $\text{Verify}(VK, \pi, C_{i,teamID}) \rightarrow \{0, 1\}$ be the SNARK verifier. The scheme is sound if, except with negligible probability in the security parameter λ ,*

$$\Pr \left[\begin{aligned} &\text{Verify}(VK, \pi, C_{i,teamID}) = 1 \\ &\wedge (\nexists F_i^* : \text{Hash}(F_i^* \parallel pk_{teamID}) = C_{i,teamID}) \\ &\mid (i, \pi) \leftarrow \mathcal{A} \end{aligned} \right] \leq \text{negl}(\lambda).$$

Definition 3 (Completeness). *Let $\{F_1, \dots, F_N\}$ be the sub-flags of a challenge and let pk_{teamID} denote the public key of team $teamID$. For every index i define the*

Algorithm 2: Proof Generation and Validation

```

1 Phase 2: Sub-Flag Proof Generation (off-chain)
2 foreach unlocked index  $j$  for  $teamID$  do
3   if  $K_j$  available and  $P(j) \subseteq teamProgress[teamID]$  then
4     decrypt  $S_j$  with  $K_j$ ; solve puzzle and capture  $F_j$ ;
5     if solve succeeds then
6        $C_{j,teamID} \leftarrow storedCommit[j][teamID]$ ;
7        $\pi_j \leftarrow Prove(PK, F_j, C_{j,teamID})$ ;
8        $m_j \leftarrow (j, \pi_j, nonce, teamID)$  // nonce is a fresh value
          chosen by the team for replay protection
9        $\sigma \leftarrow Sign(sk_{teamID}, m_j)$ ;
10      submit  $(j, \pi_j, nonce, teamID, \sigma)$ ;

11 Phase 3: On-Chain Verification (contract)
12 Input:  $(j, \pi_j, nonce, teamID, \sigma)$ 
13 Verify EIP-712 signature and nonce; reject on failure.
14 Reject if any parent  $p \in P(j)$  is missing from  $teamProgress[teamID]$ .
15  $C_{j,teamID} \leftarrow storedCommit[j][teamID]$ . Reject if
    proofHashUsed[ $C_{j,teamID}$ ] is true.
16  $valid \leftarrow Verify(VK, \pi_j, C_{j,teamID})$ ; reject if  $valid = false$ .
17 proofHashUsed[ $C_{j,teamID}$ ]  $\leftarrow true$ ; add  $j$  to
    teamProgress[teamID];
    teamPoints[teamID] += basePointsPerFlag;
    solveTime[teamID][j]  $\leftarrow block.timestamp$ .
18 For each child  $k$  with  $j \in P(k)$  and all  $P(k)$  proven, emit
    KeyEligibilityRequest( $k, teamID$ ) tuple.

19 Off-chain MPC Service
20 Upon KeyEligibilityRequest( $k, teamID$ ): collect  $t$  partial
    signatures; combine into  $sigAgg = SigAgg_t(k, teamID)$ ;
21 call releaseKey( $k, teamID, sigAgg$ ) on-chain.

22 Function releaseKey( $k, teamID, sigAgg$ )
23 if VerifyAgg( $aggPK, k, teamID, sigAgg$ ) then
24   retrieve  $K_k$  from organizer store;
25    $C_k^{enc} \leftarrow ECIES\_Encrypt(pk_{teamID}, K_k)$ ;
26   emit ReleasedKey( $teamID, k, C_k^{enc}$ ) tuple;

27 Phase 4: Master Flag & Bonus
28 if  $\forall j \in V_M : j \in teamProgress[teamID]$  then
29    $C_{M,teamID} \leftarrow storedCommit[M][teamID]$ ;
30    $\pi_M \leftarrow Prove(PK, F_M, C_{M,teamID})$ ;
31    $m_M \leftarrow (M, \pi_M, nonce, teamID)$ ;
32    $\sigma \leftarrow Sign(sk_{teamID}, m_M)$ ;
33   submit  $(M, \pi_M, nonce, teamID, \sigma)$ ;
34   Verify EIP-712 signature and nonce; reject on failure;
35    $valid \leftarrow Verify(VK, \pi_M, C_{M,teamID})$ ; reject if  $valid = false$ ;
36   teamPoints[teamID] += finalBonus;
37   emit MasterFlagRevealed( $teamID, C_{M,teamID}$ ) tuple.

```

commitment that is specific to that team $C_{i,teamID} = Hash(F_i || pk_{teamID})$.

Let $Prove(PK, F_i, C_{i,teamID}) \rightarrow \pi_i$ and $Verify(VK, \pi_i, C_{i,teamID}) \rightarrow \{0, 1\}$ be, respectively, the prover and verifier algorithms of the underlying zk-SNARK. The scheme is complete if an honest participant who genuinely knows F_i such that $C_{i,teamID} = Hash(F_i || pk_{teamID})$ can produce a proof that will be accepted except with negligible probability in the security parameter λ :

$$\Pr \left[Verify(VK, \pi_i, C_{i,teamID}) = 1 \mid \pi_i \leftarrow Prove(PK, F_i, C_{i,teamID}) \right] \geq 1 - \text{negl}(\lambda).$$

Definition 4 (Public Verifiability). Let $\{F_i\}_{i=1}^N$ be the sub-flags and, for each team identifier $teamID$ with public key pk_{teamID} , let $C_{i,teamID} = Hash(F_i || pk_{teamID})$ be the corresponding team-bound commitment. Let (PK, VK) be the SNARK keys and let $\pi_i \leftarrow Prove(PK, F_i, C_{i,teamID})$ denote a proof posted on chain together with an organiser aggregate signature $SigAgg_t(k, teamID)$ on the key-release message for

node k . Let $aggPK$ be the public aggregate verification key of the organisers. We say the scheme is publicly verifiable if, for every $(i, teamID)$ and every party $\mathcal{V}$ that is given only the public parameters $(VK, aggPK)$ and the on-chain transcript $(\pi_i, C_{i,teamID}, SigAgg_t(k, teamID))$, it holds that

$$Verify(VK, \pi_i, C_{i,teamID}) = 1 \quad \text{and} \quad VerifyAgg(aggPK, SigAgg_t(k, teamID), k, teamID) = 1$$

if and only if both of the following statements are true: (1) there exists a witness F_i such that $C_{i,teamID} = Hash(F_i || pk_{teamID})$, and (2) at least t organisers have produced valid partial signatures authorising the release of the decrypt key for node k to team $teamID$.

Definition 5 (DAG-Gated Progress). Let $\mathcal{G} = (V, E)$ be a directed acyclic graph whose vertices index the sub-flags $\{F_j\}_{j \in V}$ and puzzles $\{S_j\}_{j \in V}$. For each team $teamID$ with public key pk_{teamID} and each $j \in V$, define the team-bound commitment $C_{j,teamID} = Hash(F_j || pk_{teamID})$, and let $\pi_j \leftarrow Prove(PK, F_j, C_{j,teamID})$ be the corresponding zero-knowledge proof submitted by $teamID$.

Let $SC(j, \pi_j, teamID) \in \{accept, reject\}$ denote the (deterministic) scoreboard procedure which, on input $(j, \pi_j, teamID)$ and the current on-chain state, performs:

- 1) verification of the EIP-712 signature under pk_{teamID} ,
- 2) a check that $P(j) \subseteq teamProgress[teamID]$, and
- 3) a call to $Verify(VK, \pi_j, C_{j,teamID})$.

We say that $SC(j, \pi_j, teamID) = accept$ if and only if all three checks succeed; otherwise it outputs reject. For each team $teamID$ and vertex $j \in V$, let $Unlocked[teamID][j] \in \{true, false\}$ be the on-chain predicate indicating whether $teamID$ is allowed to access the ciphertext $E_{K_j}(S_j)$. Let $SigAgg_t(j, teamID)$ denote an aggregated threshold signature on the canonical release message for $(j, teamID)$, produced from at least t organisers and publicly verifiable under the aggregate key $aggPK$.

The protocol achieves DAG-gated progress if, for every team $teamID$ and every vertex $j \in V$, the following equivalence holds in every reachable on-chain state:

$$Unlocked[teamID][j] = true \iff \forall p \in P(j) : SC(p, \pi_p, teamID) = accept, \quad \text{and} \quad VerifyAgg(aggPK, SigAgg_t(j, teamID)) = 1.$$

and $Unlocked[teamID][j] = false$ otherwise. In particular, the contract emits only the team-specific ciphertext $C_{j,teamID}^{enc} = ECIES_Encrypt(pk_{teamID}, K_j)$ when $Unlocked[teamID][j]$ first becomes true, so no team can decrypt S_j or attempt to solve it without first satisfying all DAG prerequisites and obtaining a valid organiser threshold release.

Definition 6 (Chain-of-Proof Consistency). Let $G = (V, E)$ be the DAG of sub-flags and let T range over team identifiers. For each team $teamID$, let $teamProgress[teamID] \subseteq V$ denote the set of vertices j for which the scoreboard has already accepted a proof from $teamID$, and

let $\text{proofHashUsed}[C_{j,\text{teamID}}] \in \{\text{false}, \text{true}\}$ be the one-time-use flag associated with the team-bound commitment $C_{j,\text{teamID}} = \text{Hash}(F_j \parallel pk_{\text{teamID}})$. Upon receiving a submission $(j, \pi_j, \text{nonce}, \text{teamID}, \sigma)$, the scoreboard computes

$$\text{SC}(j, \pi_j, \text{teamID}) = \begin{cases} \text{accept,} & \text{if } P(j) \subseteq \text{teamProgress}[\text{teamID}] \\ & \wedge \text{proofHashUsed}[C_{j,\text{teamID}}] = \text{false} \\ & \wedge \text{Verify}(\text{VK}, \pi_j, C_{j,\text{teamID}}) = 1 \\ \text{reject,} & \text{otherwise.} \end{cases}$$

We say the protocol satisfies chain-of-proof consistency if for every probabilistic polynomial-time adversary $\mathcal{A}$, for every team teamID and every vertex $j \in V$,

$$\Pr \left[\text{SC}(j, \pi_j, \text{teamID}) = \text{accept} \wedge \exists p \in P(j) : (p \notin \text{teamProgress}[\text{teamID}] \vee \text{proofHashUsed}[C_{j,\text{teamID}}] = \text{true}) \right] \leq \text{negl}(\lambda).$$

where the probability is taken over the randomness of $\mathcal{A}$, the cryptographic setup, and the protocol execution.

Definition 7 (Proof-of-Completion). Let $\mathcal{G} = (V, E)$ be the DAG of sub-flags $\{F_j\}_{j \in V}$ and let pk_{teamID} be the public key of team teamID . For each $j \in V$, define the team-bound commitment $C_{j,\text{teamID}} = \text{Hash}(F_j \parallel pk_{\text{teamID}})$. Assume the system also fixes a master flag F_M and stores the commitment $C_{M,\text{teamID}} = \text{Hash}(F_M \parallel pk_{\text{teamID}})$ on chain during setup. Let $\text{teamProgress}[\text{teamID}] \subseteq V$ denote the set of vertices for which the scoreboard has accepted a proof from teamID . In Phase 4 of the protocol, the contract exposes a procedure

$$\text{SC}_M(\text{teamID}, \pi_M) \in \{\text{accept}, \text{reject}\},$$

which, on input a master-flag submission $(F_M, \pi_M, \text{nonce}, \text{teamID}, \sigma)$, performs: (1) a check of the EIP-712 signature σ and freshness of nonce ; (2) a check that $\text{teamProgress}[\text{teamID}] = V$; (3) a check that $\text{Verify}(\text{VK}, \pi_M, C_{M,\text{teamID}}) = 1$. The procedure outputs accept if and only if all three checks hold; otherwise it outputs reject . We say the scheme achieves proof-of-completion if, for every probabilistic polynomial-time adversary $\mathcal{A}$ and every security parameter λ ,

$$\Pr \left[\exists \text{teamID} : \text{SC}_M(\text{teamID}, \pi_M) = \text{accept} \wedge (\text{teamProgress}[\text{teamID}] \neq V \vee \text{Verify}(\text{VK}, \pi_M, C_{M,\text{teamID}}) = 0) \right] \leq \text{negl}(\lambda).$$

Definition 8 (Key-Privacy). Let pk_{teamID} be the public key of team teamID and write $C^{\text{enc}} = \text{Enc}_{pk_{\text{teamID}}}(K)$ for the ECIES ciphertext that the contract stores on-chain when releasing a decrypt key K . The scheme is key-private if, for every probabilistic polynomial-time adversary $\mathcal{A}$ that is given $(pk_{\text{teamID}}, C^{\text{enc}})$ and all auxiliary on-chain data, and that may adaptively query the decryption oracle on any ciphertext

except C^{enc} , the advantage in distinguishing two equally-long keys remains negligible:

$$\left| \Pr[\mathcal{A}(pk_{\text{teamID}}, \text{Enc}_{pk_{\text{teamID}}}(K_0)) = 0] - \Pr[\mathcal{A}(pk_{\text{teamID}}, \text{Enc}_{pk_{\text{teamID}}}(K_1)) = 0] \right| \leq \text{negl}(\lambda).$$

Under the IND-CCA2 security of ECIES [34], this probability bound holds, ensuring that observers learn nothing about the released key without the corresponding secret key sk_{teamID} .

Definition 9 (Anti-Replay). For every team teamID and vertex index j the contract stores a Boolean flag $\text{proofHashUsed}[C_{j,\text{teamID}}]$, where $C_{j,\text{teamID}} = \text{Hash}(F_j \parallel pk_{\text{teamID}})$. The flag is initialised to false and is set to true the first time a proof for $C_{j,\text{teamID}}$ is accepted. A submission is automatically rejected if the flag is already true . The scheme is anti-replay if, for any probabilistic polynomial-time adversary $\mathcal{A}$,

$$\Pr \left[\exists j, \text{teamID} : \text{SC}(j, \pi^{(1)}, \text{teamID}) = \text{accept} \wedge \text{SC}(j, \pi^{(2)}, \text{teamID}) = \text{accept} \right] \leq \text{negl}(\lambda),$$

where $\pi^{(1)}$ and $\pi^{(2)}$ are two (possibly identical) proofs, submitted in the same or different transactions, both referencing the commitment $C_{j,\text{teamID}}$.

Definition 10 (Threshold-Robust Organiser Security). Let n be the total number of organisers and t the reconstruction threshold ($1 < t \leq n$). Assume that each sub-flag F_j , decrypt key K_j and master flag F_M is shared among the organisers using a (t, n) Shamir secret-sharing scheme, and that key-release authorisations are issued as BLS threshold signatures $\text{SigAgg}_t(\cdot)$ under an aggregate public key aggPK .

The scheme satisfies threshold-robust organiser security if the following three properties hold for every probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ that may adaptively corrupt at most $n - t$ organisers.

(i) **Secrecy.** Let $\text{view}_{\text{real}}(\mathcal{A})$ denote the random variable consisting of all information observed by $\mathcal{A}$ in an execution (in particular, all corrupted Shamir shares for $\{F_j, K_j, F_M\}$ and all public messages), and let $\text{view}_{\text{sim}}(\mathcal{A})$ be the output of a PPT simulator Sim that is given only the public parameters and no secret values. Secrecy holds if for every such $\mathcal{A}$ there exists a PPT Sim such that

$$|\Pr[\mathcal{D}(\text{view}_{\text{real}}(\mathcal{A})) = 1] - \Pr[\mathcal{D}(\text{view}_{\text{sim}}(\mathcal{A})) = 1]| \leq \text{negl}(\lambda)$$

for every PPT distinguisher $\mathcal{D}$. In particular, any coalition of fewer than t organisers gains negligible information about F_j, K_j or F_M .

(ii) **Unforgeability.** Let $\mathcal{A}$ interact with the threshold-signature oracle and obtain at most $n - t$ partial signing keys. Consider the event that $\mathcal{A}$ outputs a pair (m, σ) such that

$\text{VerifyAgg}(\text{aggPK}, \sigma, m) = 1$ and for which σ cannot be written as the aggregation of t valid partial signatures on m . Unforgeability requires that

$$\Pr[\text{VerifyAgg}(\text{aggPK}, \sigma, m) = 1 \wedge \sigma \notin \text{Agg}(\{\sigma_1, \dots, \sigma_t\})] \leq \text{negl}(\lambda)$$

which follows from the EUF-CMA security of the underlying BLS threshold scheme.

(iii) **Availability.** Fix any honestly generated key-release request m and any execution in which at least t organisers remain honest and online. Availability requires that, except with negligible probability in λ , there exists a time at which a valid aggregated signature $\sigma^* = \text{SigAgg}_t(m)$ is produced and accepted by $\text{VerifyAgg}(\text{aggPK}, \sigma^*, m)$.

Security Proof Considering the desired security properties defined in subsection IV-A, we prove the following theorem in Appendix A

Theorem 1 (Security of zk-MPSFV). Assume that (i) $\text{Hash}(\cdot)$ is collision-resistant; (ii) $\Psi = (\text{Setup}, \text{Prove}, \text{Verify})$ is a sound, complete and zero-knowledge SNARK; (iii) Σ is an EUF-CMA secure BLS threshold-signature scheme; (iv) ECIES is IND-CCA2 secure; and (v) Shamir (t, n) secret sharing is perfectly hiding. Suppose further that the scoreboard contract implements System Setup algorithm 1 and Proof Generation and Validation algorithm 2 on a public, immutable blockchain, and that the adversary model is as in subsection III-A (Types $\mathcal{A}_1$ – $\mathcal{A}_3$). Then, for every probabilistic polynomial-time adversary $\mathcal{A}$ of Type $\mathcal{A}_1$ – $\mathcal{A}_3$ and every security parameter λ , the zk-MPSFV protocol satisfies Definitions 1–10, i.e. the ten security goals: sub-flag zero-knowledge, soundness, completeness, public verifiability, DAG-gated progress, chain-of-proof consistency, proof-of-completion, key-privacy, anti-replay, and threshold-robust organiser security, except with negligible probability $\text{negl}(\lambda)$.

V. PERFORMANCE EVALUATION AND ANALYSIS

We instantiate our scheme with PLONK [25]. PLONK provides a *universal* trusted setup: a single (pk, vk) suffices for any circuit up to 2^{28} gates. Proofs usually fall in the 700 to 900 bytes range and on-chain verification costs $\approx 350\,000$ gas [38]

Experimental Setup: Our evaluation uses one organizer server hosting the $(t=2, n=3)$ MPC daemons, BLS key store, and a local zkSync Era node, and a separate host running 30 Ubuntu 22.04 VMs, one per team, each with *Circom 2.1.5*, *snarkJS 0.7.1*, and an ECIES wallet. Benchmarks run on an Intel i7-11800H (8 cores), 32 GB RAM, Ubuntu 22.04, with Solidity compiled using hardhat 2.22.0. The Scoreboard contract is deployed on the zkSync Era testnet and periodically finalised to Ethereum L1 (depth $k=64$). L1 deployments target Goerli [44]; for comparability, we report gas and normalize costs to 30 gwei and ETH/USD = \$3,435.06 (July 15, 2025).²

²Goerli and zkSync testnets do not reflect stable mainnet fees; USD values are indicative only, as actual costs vary with congestion and ETH price.

We evaluate a Jeopardy-style CTF where each team completes a 5-step gated challenge with one proof per step; larger replayed proof batches are used to stress-test on-chain verification capacity. Scripts, scenario files, logs, model validation, and benchmark files are publicly archived.³

Results and Discussion: We report performance from two complementary viewpoints. Table III provides *baseline microbenchmarks* for the main components of the prototype (off-chain setup, contract deployments, proof generation, and the L1 proof-submission transaction). Table IV then provides a focused *ablation/optimization breakdown* for the *on-chain proof-submission path*, starting from the same baseline proof transaction measured in Table III (row “Proof tx”) and applying successive optimizations cumulatively. Generating 1000 sub-flags, hashing with `keccak256`, and producing a $(t, n) = (2, 3)$ Shamir share set takes 1.25 s (Table III), i.e., 1.25 ms per sub-flag, of which 40% is synchronous disk I/O. A single buffered write reduces this cost to 0.45 ms. Moreover, the PLONK verifier dominates byte-code size and consumes 1.86 M gas, while the *Scoreboard* consumes another 0.89 M gas; under our 30 gwei normalization these correspond to \$191.7 and \$91.7, respectively, i.e., \$283.4 total. Moving the same byte-code to a roll-up such as zkSync Era [45] or Arbitrum One can reduce user fees by roughly an order of magnitude without compromising L1-backed finality guarantees [43]. With the three-chunk PLONK circuit⁴ shipped in the artifact, a single proof requires, on average, 5.34 s and 180 MB RAM; 90.5% of the latency is spent in the FFT-heavy proving phase. By using all eight CPU cores, we reduce the end-to-end elapsed time for 8 batches of proofs by a factor of 6.5 $\times$, and the maximum RAM footprint during that run was 1.8 GB. For proof submission on Goerli, the transaction `submitSubFlagProof` verifies the proof, updates internal state, and emits `KeyEligibilityRequest`: it consumes 1.73 M gas. Together with the subsequent `releaseKey` call (210 k gas), an end-to-end “proof+release” step consumes ≈ 1.94 M gas; thus, under the 30 M gas block limit, at most $\lfloor 30/1.73 \rfloor \approx 17$ proof verifications (or $\lfloor 30/1.94 \rfloor \approx 15$ proof+release pairs) fit in a single L1 block. On zkSync, the same calldata uses ≈ 170 k L2 gas and typically settles in < 1 s in our test-net runs.

TABLE III: Raw Performance Stats (Goerli, L1). L1 costs are computed under a normalized gas price of 30 gwei and ETH/USD = \$3,435.06 (July 15, 2025).

Task	Time	Gas /\$	Note
Flag+share (1k)	1.25 s	—	1.25 ms/flag
Verifier deploy	1.42 s	1.86M / \$191.7	68% of total
Scoreboard deploy	0.38 s	0.89M / \$91.7	—
PLONK proof (1)	5.34 s	—	90.5% FFT/prove
Proof tx	8.52 s	1.73M / \$178.3	L1 finality $k=64$

Our functional benchmark was run on a *30-VM distributed test-bed* (one VM per team, three for the or-

³<https://anonymous.4open.science/r/ZKSNARKS-CTF-B052/>

⁴A *k-chunk verifier* is a Solidity verifier where the verifying key is split into k linked helper contracts to stay within EVM code-size limits; smaller k reduces deployment size and verification gas.

ganizers). The complete round of commitment hashing, PLONK proving, BLS key-release, and on-chain verification finishes in ≈ 25 s on zkSync Era, confirming that the prototype is lightweight enough for classroom-size events. To check headroom, we replayed ever-larger batches of signed `submitSubFlagProof` (a function in the `scoreboard.sol` smart contract) transactions against the same contracts from a single driver process. Throughput held steady at ≈ 7 tx s⁻¹ until the roll-up’s batch limit was reached, and no event ordering errors or state reverts were observed up to 5000 proofs. In our zkSync Era deployment, the on-chain call to `submitSubFlagProof` uses approximately 170,000 gas per accepted sub-flag. For budgeting, however, we report *fees measured directly from transaction receipts* rather than assuming an L2 gas price. Over our replay run ($N = 5000$ accepted proofs), the median fee paid per proof transaction was $f_{\text{proof},50} \approx 1.33 \times 10^{-6}$ ETH, and the observed fee range was $[f_{\text{proof},\min}, f_{\text{proof},\max}] = [0.70 \times 10^{-6}, 3.50 \times 10^{-6}]$ ETH.⁵ This corresponds to $\approx \$0.0046$ per proof at $\$3,435/\text{ETH}$ (or $\approx \$0.0040$ at $\$3,000/\text{ETH}$). If we consider a Jeopardy-style event in which each team submits at most m proven sub-flags in total (across all challenges), then total verification fees scale linearly as

$$\text{fee}_{\text{total}}^{(\text{ETH})} \approx N_{\text{teams}} \cdot m \cdot f_{\text{proof},50}.$$

For example, taking $N_{\text{teams}} = 1000$ and $m = 50$ yields $\text{fee}_{\text{total}} \approx 0.0665$ ETH ($\approx \$228$ at $\$3,435/\text{ETH}$; $\approx \$200$ at $\$3,000/\text{ETH}$). At the measured throughput of ≈ 7 tx s⁻¹, 50,000 proofs would clear in ≈ 2.0 hours. This extrapolation assumes comparable fee conditions to our measurement window and approximately constant per-proof gas usage; its purpose is to provide an order-of-magnitude estimate rather than a precise budget forecast. While Table III reports several end-to-end components, Table IV isolates the `submitSubFlagProof` verification path (and its verifier/contract code) and reports how its gas/fee changes as we apply optimizations. The first row of Table IV (“Baseline”) corresponds to the “Proof tx” entry in Table III. In Table IV, “2-chunk” reduces the verifier-key footprint (fewer constant chunks), “release-build” compiles the verifier with production compiler optimizations, and “BLS threshold verify” enables the intended threshold-signature check in the release logic. The final row reports deploying the same verification path on zkSync Era [45], consistent with roll-up scaling arguments [43]. These results show that zk-MPSFV comfortably handles a 30-team prototype on commodity hardware while scaling, with only linear cost growth, to a 1,000-team public event. The data confirm that proof generation is the actual bottleneck; hence, parallel provers and minor circuit pruning repay handsomely. Even so, single-proof PLONK verification remains gas-heavy; Layer-2 settlement provides an order-of-

magnitude relief while preserving on-chain auditability [43, 38].

TABLE IV: Ablation of successive optimisations on the `submitSubFlagProof` verification path. The first row (Baseline) corresponds to the “Proof tx” row in Table III. L1 costs are computed under a normalized gas price of 30 gwei and ETH/USD = $\$3,435.06$ (July 15, 2025). zkSync fees are taken directly from transaction receipts (ETH), with USD shown only as an indicative conversion. “2-chunk” refers to packaging the verifier key constants into two Solidity chunks rather than three (see text).

Optimisation step	Gas (L1)	L1 cost (\$)	Gas (L2)	zkSync fee
Baseline (3-chunk / L1)	1.73 M	178.3	—	—
2-chunk circuit	1.25 M	128.8	—	—
Release-build PLONK	0.98 M	100.9	—	—
BLS threshold verify (real)	0.77 M	79.3	—	—
Deploy to zkSync Era (L2)	—	—	0.17 M	1.33×10^{-6} ETH ($\approx \$0.0046$)

Compared to the Ed25519-based NIZKCTF backend [23], Fig. 3(a–b) compares *client-side* attestation cost on an 8-core host. NIZKCTF completes a signature-style proof in ≈ 0.35 ms, while zk-MPSFV produces a PLONK proof in ≈ 5.34 s, reflecting the FFT- and polynomial-commitment workload intrinsic to modern zk-SNARKs. Accordingly, NIZKCTF sustains ≈ 23 k proofs/s, whereas zk-MPSFV achieves ≈ 1.5 proofs/s, limited primarily by witness computation and memory pressure. Fig. 3(c–d) isolates *submission* $\rightarrow$ *acceptance* overhead *excluding local proof generation*. A Git-based NIZKCTF push completes in ≈ 1.5 s and sustains 0.67 proofs/s per worker. For zk-MPSFV, the baseline L1 deployment confirms a proof transaction in ≈ 8.52 s (0.117 proofs/s), while the optimized L2 configuration (2-chunk verifier, release build, and real threshold verification, deployed on zkSync Era) reduces confirmation time to ≈ 1 s (≈ 1 proof/s) with receipt-measured fees $f_{\text{proof},50} \approx 1.33 \times 10^{-6}$ ETH ($\approx \$0.004$ – $\$0.005$ depending on ETH/USD). Overall, the remaining performance gap to signature-only backends is dominated by client-side proving rather than chain overhead: zk-MPSFV trades speed for zero-knowledge privacy, DAG-gated progress, and on-chain auditability. Teams prioritizing minimal latency can use a signature-only backend, while higher-stakes events can adopt cheaper L2s and further prover optimizations (e.g., circuit pruning, aggregation, or alternative proof systems) without changing the front-end workflow. The reduction from ≈ 8.52 s (L1 base) to ≈ 1 s (L2 opt) confirms that the on-chain portion of the gap is not fundamental and is directly improved by the optimizations in Table IV. By contrast, the remaining gap in panels (a–b) stems from the cost of generating general-purpose zero-knowledge proofs and is therefore intrinsic to stronger privacy guarantees, though it can be narrowed with prover-side optimizations (circuit pruning/aggregation and hardware acceleration). We emphasize that zk-MPSFV targets audit-ready progression CTFs (e.g., classroom exams, qualifiers, or prize-bearing events) where organizers require verifiable, step-wise completion evidence and reduced reliance on any single trusted backend, rather than minimizing latency at all costs. In this setting, a ≈ 5 s proof-generation delay per step is typically small relative to human solve times (minutes to hours), while the on-chain audit trail and DAG-gated enforcement provide stronger integrity guarantees than signature-only backends.

⁵Fees are taken from transaction receipts using effective fee fields. Since deployment uses the zkSync Era testnet, ETH and USD values are only indicative; mainnet fees vary with L2 congestion, L1 data-availability costs, and ETH price. Unless noted otherwise, USD conversions assume ETH/USD = $\$3,435.06$ (July 15, 2025).

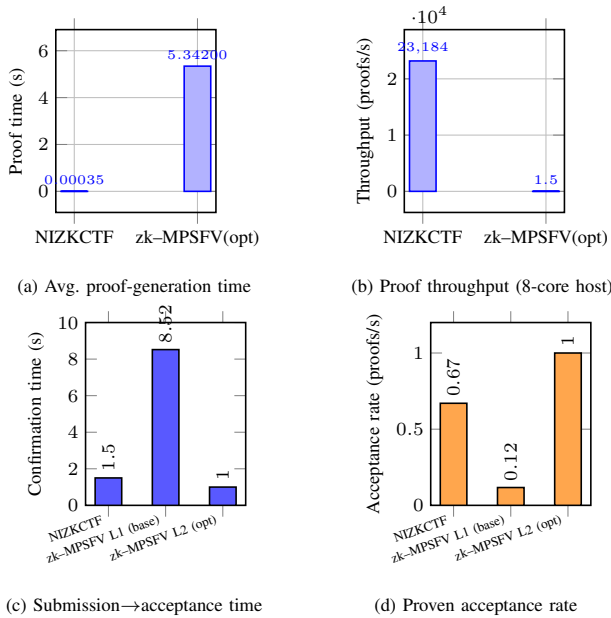


Fig. 3: Comparison of NIZKCTF [23] and zk-MPSFV after applying our optimizations. Panels (a,b) report client-side proof generation on an 8-core host (zk-MPSFV uses parallel proving). Panels (c,d) report submission-to-acceptance overhead *excluding local proof generation*; the L2 bar uses the optimized verification path (2-chunk verifier, release build, and real BLS threshold verification) deployed on zkSync Era (Table IV).

VI. CONCLUSION

zk-MPSFV shows that zero-knowledge proofs, DAG-gated challenges, and threshold key release can enable CTF scoring without reliance on a single trusted organizer. With sub-flag proofs costing about 170k gas (approximately \$0.004 on zkSync Era), the system can support fast clearance even at 1000-team scale, making it practical for small to mid-scale events. The design remains modular (e.g., PLONK/Groth16/STARKs) and can disable the MPC secrecy layer when full flag confidentiality is unnecessary, without changing the DAG state machine or one-time commitment ledger. We are extending zk-MPSFV to attack-defense settings, evaluating alternative proof stacks and batching, and porting to other L2s to study cost–security trade-offs, moving verifiable CTFs closer to mainstream deployment.

REFERENCES

- [1] Wall Street Journal, The cybersecurity talent shortage: WSJ readers dissect the problem. <https://shorturl.at/6M9f6> last accessed July, 2025.
- [2] Tay, A., Hayes, S.M., Wilson, D., Hall, E., Kaufman, D.: Gamified Cybersecurity Education Through the Lens of the Information Search Process: An Exploratory Study of Capture-the-Flag Competitions [Research-in-Progress]. *Issues in Informing Science and Information Technology* 21, 001 (2024)
- [3] Vigna, G., Borgolte, K., Corbetta, J., Doupé, A., Fratan-tonio, Y., Invernizzi, L., Kirat, D., Shoshitaishvili, Y.: Ten Years of iCTF: The Good, The Bad, and The Ugly. In: 2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14). USENIX Association (2014).
- [4] Chung, K., Cohen, J.: Learning obstacles in the capture the flag model. In: 2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14). USENIX Association (2014). <https://www.usenix.org/conference/3gse14/summit-program/presentation/chung>, last accessed July 2025.
- [5] Burket, J., Chapman, P., Becker, T., Ganas, C., Brumley, D.: Automatic problem generation for Capture-the-Flag competitions. In: 2015 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 15). USENIX Association (2015)
- [6] Švábenský, V., Čeleda, P., Vykopal, J., Brišáková, S.: Cybersecurity knowledge and skills taught in capture the flag challenges. *Computers & Security* **102**, 102154 (2021)
- [7] CTFtime.org, *Community reports raising concerns about competition integrity*, CTFtime Event #285, 2018. Online discussion archived at: <https://ctftime.org/event/285/>.
- [8] CTFtime.org, *Community concerns regarding fairness and transparency in CTF scoring mechanisms*, CTFtime Event #281, 2018. Online discussion archived at: <https://ctftime.org/event/281/>.
- [9] Gynvael Coldwind, *Discussion of organizer authority and trust assumptions in CTF competitions*, Blog post, 2018. Online at: <https://gynvael.coldwind.pl/?id=750>.
- [10] CTFtime.org, *Community feedback highlighting perceived unfair advantage in a CTF competition*, CTFtime Event #2876, 2023. Online discussion archived at: <https://ctftime.org/event/2876/>.
- [11] Groth, J.: On the size of pairing-based non-interactive arguments. In: *Advances in Cryptology – EUROCRYPT 2016*, pp. 305–326. Springer (2016)
- [12] Facebook Inc. (2019). *FBCTF* [Software]. <https://github.com/facebook/fbctf>, last accessed July 2025.
- [13] Kucek, S., & Leitner, M. (2020). An empirical survey of functions and configurations of open-source capture the flag (CTF) environments. *Journal of Network and Computer Applications*, 151, 102470. <https://doi.org/10.1016/j.jnca.2019.102470>
- [14] Joe, D. (Alias: Moloch). (2019). *RootTheBox* [Software]. <https://github.com/moloch-/RootTheBox>, last accessed July 2025.
- [15] Chung, K. (2019). *CTFd* [Software]. <https://github.com/CTFd/CTFd>, last accessed July 2025.
- [16] Nakiami. (2019). *Mellivora* [Software]. <https://github.com/Nakiami/mellivora>, last accessed July 2025.
- [17] DEF CON. *DEF CON CTF Official Rules*. <https://defcon.org/html/links/dc-ctf.html> (last accessed July 2025).
- [18] Google. *Google CTF Rules*. <https://capturetheflag.withgoogle.com/> (last accessed July 2025).
- [19] European Cyber Security Challenge (ECSC). Retrieved from <https://ecsc.eu/>. Last accessed June 2024
- [20] Kara, M., Karampidis, K., Sayah, Z., Laoudi, A., Pa-

- padourakis, G., & Abid, M. N. (2023). A password-based mutual authentication protocol via zero-knowledge proof solution. In *International Conference on Applied Cyber-security* (pp. 31–40). Cham: Springer Nature Switzerland.
- [21] Abdolmaleki, B., Asadi, A. R., Asadi, V. R., Köpsell, S., Mohee, B., Roustaeifar, N., & Zarezadeh, M.: *VERIDP: Verifiable Differentially Private Training*. Cryptology ePrint Archive, Paper 2026/542 (2026). <https://eprint.iacr.org/2026/542>
- [22] Zarinjouei, F., Abdolmaleki, B., Zarezadeh, M., Mohee, B., Abidin, A., & Köpsell, S.: *zkRNN: Zero-Knowledge Proofs for Recurrent Neural Network Inference*. Cryptology ePrint Archive, Paper 2026/073 (2026). <https://eprint.iacr.org/2026/073>
- [23] Matias, P., Barbosa, P., Cardoso, T.N.C., Campos, D.M., Aranha, D.F.: NIZKCTF: A noninteractive zero-knowledge capture-the-flag platform. *IEEE Security & Privacy* **16**(6), 42–51 (2018)
- [24] Shamir, A.: How to share a secret. *Communications of the ACM* **22**(11), 612–613 (1979)
- [25] Gabizon, A., Williamson, Z.J., and Ciobotaru, O.: Plonk: Permutations over Lagrange-bases for oecumenical non-interactive arguments of knowledge. *Cryptology ePrint Archive*, Report 2019/953 (2019).
- [26] Bacho, R., Loss, J.: On the adaptive security of the threshold BLS signature scheme. In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pp. 193–207 (2022)
- [27] Boneh, D., Lynn, B., & Shacham, H. (2001, November). Short signatures from the Weil pairing. In *International Conference on the Theory and Application of Cryptology and Information Security*, pp. 514–532, Springer, Berlin, Heidelberg.
- [28] R. A. Chetwyn and L. Erdődi, “Cheat detection in cyber security capture the flag games—an automated cyber threat hunting approach,” in *Proceedings of the 28th Conference on Computer & Enterprise Security Architecture (C&ESAR)*, 2021, p. 175.
- [29] KITCTF, “Report: Flag Sharing Incidents During GPN CTF 2025,” Jul. 1, 2025. [Online]. Available: <https://kitctf.de/gpnctf-23/gpn-ctf-flagshare>. Accessed: Jan. 1, 2026.
- [30] CTFtime.org, “BlackHat MEA CTF Qualification 2025” (event page including prize information and participant comments), Sept. 2025. [Online]. Available: <https://ctftime.org/event/2876/>. Accessed: Jan. 1, 2026.
- [31] amon (j. heng), “HackIM 2016 Case Study,” *Nandy Narwhals CTF Team (blog)*, updated Feb. 3, 2016. [Online]. Available: <https://nandynarwhals.org/hackim2016-casestudy/>. Accessed: Jan. 1, 2026.
- [32] G. Coldwind, “An informal review of CTF abuse,” Jul. 23, 2022. [Online]. Available: <https://gynvael.coldwind.pl/?id=750>. Accessed: Jan. 1, 2026.
- [33] picoCTF (Carnegie Mellon University), “Competition Rules (Spring 2024),” Mar. 2024. [Online]. Available: <https://picoctf.org/competitions/2024-spring-rules.html>. Accessed: Jan. 1, 2026.
- [34] Abdalla, M., Bellare, M., Rogaway, P.: The oracle Diffie-Hellman assumptions and an analysis of DHIES. In: *Topics in Cryptology—CT-RSA 2001: The Cryptographers’ Track at RSA Conference 2001, San Francisco, CA, USA, April 8–12, 2001 Proceedings*, pp. 143–158. Springer, Berlin, Heidelberg (2001)
- [35] Katz, J., Lindell, Y.: *Introduction to Modern Cryptography: Principles and Protocols*. Chapman and Hall/CRC (2007)
- [36] Salehi, M., Clark, J., Mannan, M.: Not so immutable: Upgradeability of smart contracts on Ethereum. In: *International Conference on Financial Cryptography and Data Security*, pp. 539–554. Springer, Cham (2022)
- [37] Asif, R., Hassan, S.R.: Shaping the future of Ethereum: Exploring energy consumption in Proof-of-Work and Proof-of-Stake consensus. *Frontiers in Blockchain* **6**, 1151724 (2023)
- [38] Guo, H., Feng, Y., Wu, C., Li, Z., Xu, J.: Benchmarking ZK-Friendly Hash Functions and SNARK Proving Systems for EVM-compatible Blockchains. *arXiv preprint arXiv:2409.01976* (2024)
- [39] Zhang, J., Li, Y., Gao, J., Guan, Z., Chen, Z.: Siguard: Detecting signature-related vulnerabilities in smart contracts. In: *Proceedings of the 2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pp. 31–35. IEEE (2023)
- [40] Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. *Cryptology ePrint Archive*, Report 2018/046 (2018). <https://eprint.iacr.org/>
- [41] Bloemen, R., Logvinov, L., & Evans, J.: EIP-712: Typed structured data hashing and signing. *Ethereum Improvement Proposal (EIP)*, last accessed June 17, 2023. <https://eips.ethereum.org/EIPS/eip-712> (2017)
- [42] CTFd: Capture The Flag Framework. <https://github.com/CTFd/CTFd>. Last accessed June 2024.
- [43] V. Buterin, A rollup centric road map, *Ethereum Blog*, 2020. <https://ethereum-magicians.org/t/a-rollup-centric-ethereum-roadmap/4698>. Last accessed July 2025.
- [44] Goerli Testnet. *Ethereum Goerli Test Network*. Available at: <https://goerli.net/> [last accessed July 2025].
- [45] zkSync. *zkSync Era Documentation*. Matter Labs. Available at: <https://docs.zksync.io/zksync-era> [Last accessed July 2025].
- [46] Standards for Efficient Cryptography Group (SECG), *SECG Home Page*, <https://www.secg.org/>, accessed Jan. 24, 2026.

APPENDIX

A. Theorem 1 Proof

Proof overview. Theorem 1 is proven by Lemmas 1–10, one per security goal. Lemmas 1–4 rely on standard

SNARK/commitment properties (zero-knowledge, soundness, completeness, and public verifiability); Lemmas 5–9 establish protocol/state-machine properties (DAG gating, chain-of-proof consistency, anti-replay, and proof-of-completion); Lemma 10 covers threshold-robust organizer security and key-release correctness. Theorem 1 follows by composition.

Assume that (i) $\text{Hash}(\cdot)$ is collision-resistant, (ii) $\Psi = (\text{Setup}, \text{Prove}, \text{Verify})$ is a sound, complete, and zero-knowledge SNARK [35], [40], (iii) Σ is a BLS threshold-signature scheme that is EUF-CMA secure [27], (iv) ECIES is IND-CCA2 secure [34], and (v) Shamir (t, n) secret sharing is perfectly hiding [24]. Following the threat-model premises of subsection III-A, the scoreboard contract runs on a public, immutable Ethereum blockchain, where it verifies SNARK proofs on chain [38], enforces DAG-gated progress and team-bound commitments, releases decrypt keys only after receiving a valid organizer aggregate signature SigAgg_t , publishes each K_j as an ECIES ciphertext addressed to the requesting team, and applies replay protection through EIP-712 signed messages and the one-time flag *proofHashUsed*. Under these assumptions and runtime guarantees, we prove that zk-MPSFV attains all ten security goals against the adversaries defined in subsection III-A.

Lemma 1 (Sub-flag Zero-Knowledge). *Consider any probabilistic polynomial-time adversary $\mathcal{A}$ of Type $\mathcal{A}_1$ – $\mathcal{A}_3$ from Section III-A. Let pk_{teamID} be the public key of an honest team $teamID$ and let $C_{i,teamID} = \text{Hash}(F_i \parallel pk_{teamID})$ be its team-bound commitment to a sub-flag F_i . Then $\mathcal{A}$ has at most negligible advantage in distinguishing a real proof $\pi_i \leftarrow \text{Prove}(\text{PK}, F_i, C_{i,teamID})$ from a simulated proof $\pi_i^* \leftarrow \text{Sim}(td, C_{i,teamID})$.*

Proof. We use a standard sequence of hybrid experiments.

Game 0 (Real experiment). The challenger samples F_i and computes $C_{i,teamID} = \text{Hash}(F_i \parallel pk_{teamID})$. It generates a proof $\pi_i \leftarrow \text{Prove}(\text{PK}, F_i, C_{i,teamID})$ and gives π_i together with all on-chain data allowed by the Type $\mathcal{A}_1$ – $\mathcal{A}_3$ threat model to $\mathcal{A}$.

Game 1. As in Game 0, except that the proof is generated by the simulator: $\pi_i^* \leftarrow \text{Sim}(td, C_{i,teamID})$. By the zero-knowledge property of the SNARK Ψ , the distinguishing advantage between Game 0 and Game 1 is $\text{negl}(\lambda)$.

Game 2. We now change the committed value while still using simulated proofs. Instead of committing to F_i , the challenger sets $C'_{i,teamID} = \text{Hash}(0^{|F_i|} \parallel pk_{teamID})$ and runs the simulator on $C'_{i,teamID}$. Since the commitment $C_{i,teamID}$ is computationally hiding and pk_{teamID} is fixed, $\mathcal{A}$'s advantage in distinguishing Game 1 from Game 2 is at most $\text{negl}(\lambda)$.

Game 3. Finally, we replace the random-oracle answers by fresh uniform values: for every new query x to Hash the challenger samples $h \leftarrow \{0, 1\}^\kappa$, returns h , and records (x, h) in the oracle table. By the random-oracle idealisation, the transcript distribution in Game 3 is identical to that of Game 2, so the distinguishing gap is again negligible.

In Game 3, the commitment value and the simulated proof are independent of F_i , and therefore $\mathcal{A}$'s view contains no

information about the sub-flag beyond its existence. By the triangle inequality over the three hybrid transitions, $\mathcal{A}$'s overall advantage in Game 0 is bounded by a negligible function in λ , which proves sub-flag zero-knowledge. $\square$

Lemma 2 (Soundness). *Consider any adversary $\mathcal{A}$ of Type $\mathcal{A}_1$ – $\mathcal{A}_3$ from subsection III-A. Let pk_{teamID} be the public key of an honest team $teamID$ and define the team-bound commitment*

$$C_{i,teamID} = \text{Hash}(F_i \parallel pk_{teamID}).$$

Assume that (i) the SNARK Ψ is sound, (ii) $\text{Hash}(\cdot)$ is collision-resistant, and (iii) the commitment scheme is binding. Then the probability that $\mathcal{A}$ outputs an index i and a proof π such that the verifier accepts π for $C_{i,teamID}$ while no pre-image F_i^ satisfying $\text{Hash}(F_i^* \parallel pk_{teamID}) = C_{i,teamID}$ exists in $\mathcal{A}$'s internal state is negligible in λ ; i.e.,*

$$\Pr \left[\text{Verify}(\text{VK}, \pi, C_{i,teamID}) = 1 \wedge \left(\nexists F_i^* : \text{Hash}(F_i^* \parallel pk_{teamID}) = C_{i,teamID} \mid (i, \pi) \leftarrow \mathcal{A}(1^\lambda) \right) \right] \leq \text{negl}(\lambda).$$

Proof. Suppose, towards contradiction, that there exists a PPT adversary $\mathcal{A}$ for which the above probability is non-negligible. We show how to use $\mathcal{A}$ to violate at least one of the three underlying assumptions.

Case 1: If there is no value F_i^* such that $\text{Hash}(F_i^* \parallel pk_{teamID}) = C_{i,teamID}$ at all, then the statement proved by π is false, yet $\text{Verify}(\text{VK}, \pi, C_{i,teamID}) = 1$. An algorithm that runs $\mathcal{A}$ and outputs $(C_{i,teamID}, \pi)$ in this case breaks the soundness of the SNARK Ψ .

Case 2: Suppose there exists at least one such pre-image F_i^* , but $\mathcal{A}$ can produce two different openings $F_i^* \neq F'_i$ (possibly in different executions) that are both accepted as valid witnesses for the same commitment $C_{i,teamID}$. Then $\mathcal{A}$ violates the binding property of the commitment scheme, since it has found two distinct openings to a single commitment.

Case 3: If $\mathcal{A}$ can find distinct values $F_i^* \neq F'_i$ such that $\text{Hash}(F_i^* \parallel pk_{teamID}) = \text{Hash}(F'_i \parallel pk_{teamID}) = C_{i,teamID}$, then it has found a collision for Hash, contradicting the assumption that Hash is collision-resistant.

In all three cases we obtain a PPT algorithm that breaks one of: SNARK soundness, commitment binding, or hash collision-resistance. By the assumptions stated at the beginning of this section, each of these events occurs with probability at most $\text{negl}(\lambda)$. Therefore the probability that $\mathcal{A}$ can cause the scoreboard to accept a proof for $C_{i,teamID}$ without effectively knowing a valid pre-image is also bounded by $\text{negl}(\lambda)$, establishing the soundness claim. $\square$

Lemma 3 (Completeness). *Consider any adversary $\mathcal{A}$ of Type $\mathcal{A}_1$ – $\mathcal{A}_3$ as in Section III-A. Let pk_{teamID} be the public key of an honest team $teamID$ and define the team-bound commitment*

$$C_{i,teamID} = \text{Hash}(F_i \parallel pk_{teamID}).$$

Let $\pi_i \leftarrow \text{Prove}(\text{PK}, F_i, C_{i, \text{teamID}})$ be the proof output by the honest prover of the underlying SNARK Ψ . If F_i is a correct sub-flag for commitment $C_{i, \text{teamID}}$, then

$$\Pr[\text{Verify}(\text{VK}, \pi_i, C_{i, \text{teamID}}) = 1] = 1 - \text{negl}(\lambda),$$

where the probability is over the randomness of Setup and Prove and any internal randomness of the verifier.

Proof. By assumption, the SNARK $\Psi = (\text{Setup}, \text{Prove}, \text{Verify})$ is complete: for any statement–witness pair (x, w) in the language, an honest prover running $\text{Prove}(\text{PK}, w, x)$ produces a proof that is accepted by $\text{Verify}(\text{VK}, \cdot, x)$ with probability $1 - \text{negl}(\lambda)$. In our setting, the public statement is the commitment $C_{i, \text{teamID}} = \text{Hash}(F_i \parallel pk_{\text{teamID}})$ and the witness is F_i . Since pk_{teamID} is fixed and public, the mapping $(F_i, pk_{\text{teamID}}) \mapsto C_{i, \text{teamID}}$ is deterministic and part of the statement definition; it does not affect the completeness guarantee of Ψ . Therefore, whenever F_i is a valid pre-image of $C_{i, \text{teamID}}$, an honest team running $\pi_i \leftarrow \text{Prove}(\text{PK}, F_i, C_{i, \text{teamID}})$ obtains a proof that satisfies $\text{Verify}(\text{VK}, \pi_i, C_{i, \text{teamID}}) = 1$ except with negligible probability in λ . Adversaries of Type $\mathcal{A}_1$ – $\mathcal{A}_3$ may delay, reorder or inject transactions on the network, but they do not influence the internal correctness of Verify once a proof and statement are fixed. Hence, completeness holds in the full threat model: an honest participant who truly knows F_i almost surely produces a proof that the on-chain verifier accepts. $\square$

Lemma 4 (Public Verifiability). *Let $i \in V$ and let teamID be any team identifier with public key pk_{teamID} . For that pair, denote by $C_{i, \text{teamID}} = \text{Hash}(F_i \parallel pk_{\text{teamID}})$ the on-chain commitment to the sub-flag, by π_i the proof posted on-chain, and by $\text{SigAgg}_t(i, \text{teamID})$ the organiser aggregate signature attached to the release of K_i for teamID . Assume that the blockchain is append-only and publicly readable, and that the public parameters $(\text{VK}, \text{aggPK})$ are fixed and known to all parties. Then for any (possibly unbounded) verifier $\mathcal{V}$ that has read-only access to the blockchain state and to $(\text{VK}, \text{aggPK})$, the following holds: for every recorded submission $(\pi_i, C_{i, \text{teamID}}, \text{SigAgg}_t(i, \text{teamID}))$, $\mathcal{V}$ can determine its validity by evaluating*

$$b_1 \leftarrow \text{Verify}(\text{VK}, \pi_i, C_{i, \text{teamID}}) \quad \text{and} \\ b_2 \leftarrow \text{VerifyAgg}(\text{aggPK}, \text{SigAgg}_t(i, \text{teamID}))$$

and accepting the submission if and only if $b_1 = b_2 = 1$. No additional secret state or interaction with the original prover or organisers is required.

Proof. By construction of the protocol, whenever team teamID submits a proof for index i , the scoreboard contract stores on-chain the tuple $(\pi_i, C_{i, \text{teamID}}, \text{SigAgg}_t(i, \text{teamID}))$ together with any auxiliary metadata (e.g. timestamps). The public verification key VK and the aggregate organiser key aggPK are also

deployed on-chain (or are distributed as part of the system parameters) and are therefore known to any third party.

Let $\mathcal{V}$ be a verifier with read-only access to the ledger and to $(\text{VK}, \text{aggPK})$. Given any on-chain record $(\pi_i, C_{i, \text{teamID}}, \text{SigAgg}_t(i, \text{teamID}))$, $\mathcal{V}$ can:

- 1) Compute $b_1 \leftarrow \text{Verify}(\text{VK}, \pi_i, C_{i, \text{teamID}})$, which, by definition of the SNARK verifier, depends only on the public key VK and the public inputs $(\pi_i, C_{i, \text{teamID}})$.
- 2) Compute $b_2 \leftarrow \text{VerifyAgg}(\text{aggPK}, \text{SigAgg}_t(i, \text{teamID}))$, which, by definition of the aggregate signature scheme, depends only on the public aggregate key aggPK , the message (i, teamID) encoded in the signature, and the aggregated signature value itself.

Both computations are deterministic and require no secret keys or off-chain witnesses. Because the blockchain is append-only and publicly readable, every party sees the same $(\pi_i, C_{i, \text{teamID}}, \text{SigAgg}_t(i, \text{teamID}))$ and obtains the same outputs (b_1, b_2) . Therefore any observer can independently decide to accept the record if and only if $b_1 = b_2 = 1$, which shows that the protocol satisfies public verifiability in the sense of Definition 4. $\square$

Lemma 5 (DAG–Gated Progress). *Fix an honest team teamID and let $\mathcal{A}$ be any adversary of Type $\mathcal{A}_1$ – $\mathcal{A}_3$ as in subsection III-A. For every vertex j in the challenge DAG $\mathcal{G} = (V, E)$, let $P(j)$ denote the set of parents of j . Let $C_{p, \text{teamID}} = \text{Hash}(F_p \parallel pk_{\text{teamID}})$ and $\pi_p \leftarrow \text{Prove}(\text{PK}, F_p, C_{p, \text{teamID}})$ for each $p \in P(j)$. Let the contract emit the ciphertext*

$$C_{j, \text{teamID}}^{\text{enc}} = \text{ECIES_Encrypt}(pk_{\text{teamID}}, K_j)$$

whenever a key-release event for (j, teamID) occurs. Then, under the assumptions of subsection A, for every PPT adversary $\mathcal{A}$ we have

$$\Pr[C_{j, \text{teamID}}^{\text{enc}} \text{ is emitted} \wedge (\exists p \in P(j) : \text{Verify}(\text{VK}, \pi_p, C_{p, \text{teamID}}) = 0)] \leq \text{negl}(\lambda).$$

Equivalently, except with negligible probability in λ , a team obtains access to K_j (and hence to S_j) only after valid proofs for all parents $p \in P(j)$ have been accepted.

Proof. By algorithm 2, lines 21–25, the contract calls $\text{releaseKey}(j, \text{teamID}, \text{SigAgg}_t)$ only if the DAG condition $P(j) \subseteq \text{teamProgress}[\text{teamID}]$ holds, i.e. only after every parent $p \in P(j)$ has previously yielded an accepted submission for teamID . The function emits $C_{j, \text{teamID}}^{\text{enc}}$ only if $\text{VerifyAgg}(\text{aggPK}, \text{SigAgg}_t(j, \text{teamID})) = 1$. A Type- $\mathcal{A}_1$ or $\mathcal{A}_2$ adversary may front-run, reorder or inject transactions, but cannot alter the contract code or its state transition guards. To emit $C_{j, \text{teamID}}^{\text{enc}}$ while some $p \in P(j)$ lacks a valid proof, $\mathcal{A}$ must either (a) forge an accepted SNARK proof for a false statement, or (b) forge a valid aggregate organiser signature. By Lemma 2, the probability of (a) is $\text{negl}(\lambda)$. By EUF–CMA security of the BLS threshold scheme and the bound of Lemma 10, point (ii), the probability of (b) is also $\text{negl}(\lambda)$ even for a Type- $\mathcal{A}_3$ adversary that corrupts at most $n - t$

organisers. Furthermore, observing a ciphertext $C_{j,teamID}^{enc}$ for some other team $teamID'$ does not help, since by Lemma 8 and IND-CCA2 security of ECIES, no adversary without $sk_{teamID'}$ can recover K_j from that ciphertext with more than negligible advantage. By a union bound over these negligible events, the probability that $C_{j,teamID}^{enc}$ is emitted while some parent $p \in P(j)$ has not been proven is at most $\text{negl}(\lambda)$. Hence, with overwhelming probability, a team can access K_j and attempt S_j only after all prerequisites in the DAG have been satisfied. $\square$

Lemma 6 (Chain-of-Proof Consistency). *Consider any probabilistic polynomial-time adversary $\mathcal{A}$ of Type $\mathcal{A}_1$ – $\mathcal{A}_3$ from subsection III-A. For each team identifier $teamID$ the contract maintains a set $\text{teamProgress}[teamID] \subseteq V$ and, for every vertex $j \in V$, a Boolean flag $\text{proofHashUsed}[C_{j,teamID}]$, where $C_{j,teamID} = \text{Hash}(F_j \parallel pk_{teamID})$. Upon receiving a submission $(j, \pi_j, \text{nonce}, teamID, \sigma)$ the scoreboard:*

- 1) *verifies the EIP-712 signature σ under the public key pk_{teamID} ;*
- 2) *checks that $P(j) \subseteq \text{teamProgress}[teamID]$; and*
- 3) *checks that $\text{proofHashUsed}[C_{j,teamID}] = \text{false}$.*

If all checks succeed and $\text{Verify}(\text{VK}, \pi_j, C_{j,teamID}) = 1$, the contract sets $\text{proofHashUsed}[C_{j,teamID}] \leftarrow \text{true}$ and adds j to $\text{teamProgress}[teamID]$; otherwise it rejects.

Under the EUF-CMA security of the signature scheme Σ and the one-time-use rule for proofHashUsed , the probability that $\mathcal{A}$ causes the contract to accept a submission for vertex j that either (i) is not signed by the corresponding team key or (ii) violates the DAG dependencies or reuses a previously accepted commitment is at most $\text{negl}(\lambda)$.

Proof. We show that any successful attack on chain-of-proof consistency would contradict either the EUF-CMA security of Σ or the contract's explicit checks. First, consider impersonation. To attribute a valid proof for vertex j to an honest team $teamID$, $\mathcal{A}$ must produce a signature σ that verifies under pk_{teamID} on the EIP-712 encoded message $(j, \pi_j, \text{nonce}, teamID)$. By the EUF-CMA security of Σ , the probability that $\mathcal{A}$ forges such a signature without knowing the corresponding secret key sk_{teamID} is at most $\text{negl}(\lambda)$. Hence, except with negligible probability, every accepted submission is genuinely authorised by the claimed team. Second, consider out-of-order or replayed submissions by a team that does control sk_{teamID} . If $\mathcal{A}$ submits a proof for vertex j while there exists a parent $p \in P(j)$ with $p \notin \text{teamProgress}[teamID]$, then line 14 of Algorithm algorithm 2 (Phase 3) forces rejection: the contract explicitly checks the inclusion $P(j) \subseteq \text{teamProgress}[teamID]$ before accepting. Thus no child vertex can be accepted before all of its parents have been accepted for the same team. Similarly, if $\mathcal{A}$ reuses a previously accepted commitment $C_{j,teamID}$ in a new transaction, the flag $\text{proofHashUsed}[C_{j,teamID}]$ is already set to *true* from the first successful submission. Line 15 of algorithm 2 causes the contract to reject any subsequent submission referencing the same commitment. Therefore each commitment $C_{j,teamID}$ can be

consumed at most once. Combining these facts, any accepted proof for j must (i) be signed by the team that owns pk_{teamID} , (ii) respect the DAG order through the teamProgress check, and (iii) correspond to a commitment that has not been used before. The probability that an adversary of Type $\mathcal{A}_1$ – $\mathcal{A}_3$ violates any of these conditions is bounded by $\text{negl}(\lambda)$, which establishes chain-of-proof consistency. $\square$

Lemma 7 (Proof-of-Completion). *Let $C_{M,teamID} = \text{Hash}(F_M \parallel pk_{teamID})$ be the team-bound master-flag commitment fixed during setup. In Phase 4 of algorithm 2 the scoreboard adds finalBonus to $\text{teamPoints}[teamID]$ only if all vertices $j \in V$ satisfy $j \in \text{teamProgress}[teamID]$ and $\text{Verify}(\text{VK}, \pi_M, C_{M,teamID}) = 1$ for the submitted master-flag proof π_M . Hence, except with negligible probability, no team that has not solved every sub-flag can obtain the final bonus.*

Proof. In Phase 4 of algorithm 2 the contract first checks that $\{j : j \in V\} \subseteq \text{teamProgress}[teamID]$ (lines 26–27); otherwise the call is rejected and no bonus is paid. If this condition holds, the contract retrieves the stored commitment $C_{M,teamID}$ (line 28) and accepts the master-flag submission only if $\text{Verify}(\text{VK}, \pi_M, C_{M,teamID}) = 1$ (lines 29–34). By completeness and soundness of the SNARK, any accepted π_M must, except with probability $\text{negl}(\lambda)$, certify knowledge of a preimage F_M such that $\text{Hash}(F_M \parallel pk_{teamID}) = C_{M,teamID}$. Therefore a team that has not first produced valid proofs for every vertex in V cannot reach the branch where finalBonus is added, and a team that does reach it can only succeed by presenting a correct master-flag proof for its own commitment. Hence, with all but negligible probability, the final bonus is awarded only upon full completion of the DAG by that team. $\square$

Lemma 8 (Key-Privacy). *Let $\mathcal{A}$ be any probabilistic polynomial-time adversary of Type $\mathcal{A}_1$ – $\mathcal{A}_3$ from subsection III-A. Fix an honest team $teamID$ with public key pk_{teamID} , and let $C_j^{enc} = \text{ECIES_Encrypt}(pk_{teamID}, K_j)$ be the on-chain ciphertext used to deliver the decrypt key K_j . Assume that the underlying ECIES scheme is IND-CCA2 secure in the sense of Abdalla et al. [34]. Then for any two equal-length keys K_0, K_1 we have*

$$\left| \Pr[\mathcal{A}(pk_{teamID}, \text{Enc}_{pk_{teamID}}(K_0)) = 1] - \Pr[\mathcal{A}(pk_{teamID}, \text{Enc}_{pk_{teamID}}(K_1)) = 1] \right| \leq \text{negl}(\lambda).$$

In particular, observing C_j^{enc} and all other on-chain data gives $\mathcal{A}$ only negligible advantage in learning any information about K_j without the secret key sk_{teamID} .

Proof. In our setting, a Type $\mathcal{A}_1$ – $\mathcal{A}_3$ adversary has full read access to the blockchain state, and hence to the ciphertext C_j^{enc} and all associated metadata. It may also obtain decryptions of arbitrary ciphertexts under pk_{teamID} except for the challenge ciphertext C_j^{enc} itself, because the contract never exposes a decryption oracle for that particular value. This is exactly the IND-CCA2 game for ECIES as defined in [34]: $\mathcal{A}$ chooses (K_0, K_1) of equal length, receives $\text{Enc}_{pk_{teamID}}(K_b)$

for a hidden bit b , and may adaptively query a decryption oracle on any other ciphertext. By the IND-CCA2 security of ECIES, the distinguishing advantage of any PPT adversary in this game is at most $\text{negl}(\lambda)$. Substituting C_j^{enc} for the challenge ciphertext and pk_{teamID} for the challenge public key yields exactly the bound stated in the lemma. Hence, from the adversary's point of view, C_j^{enc} is computationally indistinguishable from the encryption of any other key of the same length, and in particular leaks no useful information about the true K_j without sk_{teamID} . $\square$

Lemma 9 (Anti-Replay). *For every vertex j and team teamID let $C_{j,\text{teamID}} = \text{Hash}(F_j \parallel pk_{\text{teamID}})$ be the corresponding team-bound commitment. The contract maintains a Boolean flag $\text{proofHashUsed}[C_{j,\text{teamID}}] \in \{\text{false}, \text{true}\}$, initialised to false and set to true upon the first acceptance of a proof for $C_{j,\text{teamID}}$. Then, for any probabilistic polynomial-time adversary $\mathcal{A}$ of Type $\mathcal{A}_1$ – $\mathcal{A}_3$ (as in subsection III-A) and any security parameter λ , the probability that the scoreboard accepts two distinct transactions referencing the same commitment is negligible:*

$$\Pr \left[\exists j, \text{teamID} : \text{SC}(j, \pi^{(1)}, \text{teamID}) = \text{accept} \wedge \text{SC}(j, \pi^{(2)}, \text{teamID}) = \text{accept} \right] \leq \text{negl}(\lambda).$$

where SC denotes the on-chain verification procedure of Algorithm 2.

Proof. By Algorithm 2, line 15, any submission for which $\text{proofHashUsed}[C_{j,\text{teamID}}] = \text{true}$ is immediately rejected. Hence, once the first proof for $C_{j,\text{teamID}}$ has been accepted and the flag set to *true*, a second acceptance for the same commitment can only occur if either: (i) the first state update is later reverted on-chain, or (ii) the adversary forges a new valid submission that bypasses the flag check. Event (i) is excluded by the blockchain assumptions in subsection III-A: after k confirmations the block containing the first acceptance is immutable, so the flag value cannot be rolled back. For event (ii), note that proofHashUsed is a deterministic mapping inside the contract state. Any second transaction that reaches the verification code for the same $C_{j,\text{teamID}}$ will observe the flag as *true* and be rejected, regardless of the proof or signature content. The only way to circumvent this behaviour is to forge an alternative contract execution that ignores or overwrites the flag, which would require either breaking the underlying consensus or altering the deployed bytecode, both ruled out by the model. Cryptographic forgery (e.g. of signatures) cannot help, because the rejection condition depends solely on the stored flag. Therefore each commitment $C_{j,\text{teamID}}$ can be associated with at most one accepted proof in any execution. The probability that $\mathcal{A}$ causes two distinct accepted submissions for the same commitment is thus bounded by $\text{negl}(\lambda)$. $\square$

Lemma 10 (Threshold-Robust Organisation Security). *Let n be the total number of organisers and t the reconstruction threshold with $1 < t \leq n$. Suppose that (i) sub-flags F_j , decrypt keys K_j and the master flag F_M are shared*

using (t, n) -Shamir secret sharing, and (ii) key release is authorised via a BLS threshold-signature scheme Σ which is EUF-CMA secure [27]. Let $\mathcal{A}$ be any Type $\mathcal{A}_3$ adversary (subsection III-A) that corrupts at most $n - t$ organisers. Then, for every security parameter λ :

- (1) (*Secrecy*) *The joint view of the corrupted organisers—including all their Shamir shares and local randomness—is computationally indistinguishable from a view generated without knowledge of the underlying secrets. In particular, $\mathcal{A}$ obtains no information about any F_j , K_j or F_M .*
- (2) (*Unforgeability*) *The probability that $\mathcal{A}$ outputs a message m and an aggregate signature $\text{SigAgg}_t(m)$ that verifies under the public aggregate key aggPK without having obtained t valid partial signatures on m is at most $\text{negl}(\lambda)$.*
- (3) (*Availability*) *If at least t organisers remain honest and online, then for every correctly formed key-release request m a valid $\text{SigAgg}_t(m)$ is eventually produced and accepted by the contract, except with negligible probability.*

Proof. For *Secrecy*, perfect hiding of Shamir (t, n) sharing implies that any set of at most $n - t$ shares is statistically independent of the shared secret [24]. Thus the joint distribution of the corrupted organisers' views can be simulated without access to F_j , K_j or F_M , and $\mathcal{A}$ gains no information about these values. For *Unforgeability*, the EUF-CMA security of the BLS threshold scheme [27] guarantees that any PPT adversary that controls fewer than t signing keys has at most $\text{negl}(\lambda)$ probability of producing a verifying aggregate signature on a new message without collecting t valid partial signatures. Since $\mathcal{A}$ corrupts at most $n - t$ organisers, it controls at most that many partial keys and therefore cannot forge $\text{SigAgg}_t(m)$ except with negligible probability. For *Availability*, the threat model in subsection III-A assumes that at least t organisers remain honest and responsive. For any well-formed key-release request m , each honest organiser eventually contributes a correct partial signature; the aggregation procedure then combines these into a valid $\text{SigAgg}_t(m)$, which the contract accepts. Network adversaries may delay individual messages, but cannot indefinitely prevent their inclusion on chain once posted to L1 (forced-inclusion assumption). Combining these three points, we conclude that with fewer than t corrupted organisers the adversary cannot learn, forge, or prematurely release organiser-protected secrets, and the protocol maintains both confidentiality and liveness up to negligible failure probability. $\square$

Lemmas 1–10 establish, under assumptions (i)–(v) at the beginning of this section and for every adversary of Type $\mathcal{A}_1$ – $\mathcal{A}_3$ from Section III-A, that: sub-flag zero-knowledge (Lemma 1), soundness (Lemma 2), completeness (Lemma 3), public verifiability (Lemma 4), DAG-gated progress (Lemma 5), chain-of-proof consistency (Lemma 6), proof-of-completion (Lemma 7), key-privacy (Lemma 8), anti-replay (Lemma 9), and threshold-robust organisation security (Lemma 10) all hold except with negligible probability $\text{negl}(\lambda)$. Since these ten properties are exactly the security goals stated in Theorem 1, the theorem follows.