Warfront Legends: A Hybrid Web2/Web3 Idle Auto-Battler War Game

Warfront Legends Team
contact@warfrontlegends.example
www.warfrontlegends.example

Draft, May 12, 2026

Abstract

A purely server-authoritative game would still require players to trust a central operator with custody of their cosmetic identity, earned achievements, and tradable items. We propose a design in which the gameplay loop remains entirely off-chain — low-latency, free-to-play, and indistinguishable from a conventional Web2 action title — while a separate, optional settlement layer on Base records ownership of cosmetic assets, soldier identities, and verifiable competitive results. Players need not interact with the chain to play, progress, or compete. Players who do interact with the chain receive custody of their items as ERC-721 and ERC-1155 tokens, and competitive rewards are distributed through Merkle-attested snapshots signed by the game server, verified on-chain at claim time. The system separates *gameplay state*, which must be fast and authoritative, from *ownership state*, which must be permissionless and persistent. We describe the network model, the replay verification protocol, the attestation pipeline, and the token sinks that bound emissions. We argue that this separation produces a game that is competitive without requiring trust in the operator, and onboardable without requiring a wallet.

1. Introduction

Online action games rely on a central operator for two reasons: to run an authoritative simulation (so that physics, hit detection, and scoring agree across clients) and to act as a custodian for player progress (cosmetics, ranks, statistics). The first responsibility is technical and largely unavoidable in latency-sensitive games. The second is a matter of trust: the operator decides what players own, what can be transferred, and what persists if the operator stops running the service.

Web3 games have attempted to address the trust problem by moving gameplay itself on-chain. This approach is sound for turn-based and asynchronous genres but fails for real-time action: blockchains do not provide the millisecond-tick determinism that side-scrolling combat demands, and attempts to bridge the gap typically introduce latency, cost, or new trust assumptions that defeat the original goal.

We propose a different separation. The simulation runs off-chain, exactly as in a conventional online game, with the operator as the authority over gameplay state. Ownership of player-facing artifacts — soldier identity, cosmetic gear, seasonal reward entitlements — is recorded on Base, a Layer-2 network on Ethereum. The two layers communicate through signed attestations, never through synchronous calls. A player who never connects a wallet experiences a free-to-play action game. A player who connects a wallet additionally gains custody of their items and the ability to verify that competitive rewards were issued correctly.

This paper describes the protocol that connects the two layers, the replay mechanism that allows the chain to verify off-chain results without re-running the simulation, and the token model that bounds reward issuance to measurable in-game activity.

2. System Overview

The system has four components. The *client* is a browser-based game running on the player's device. The *simulation server* executes the authoritative game loop and resolves all combat outcomes. The *attestation service* produces signed statements about completed matches. The *settlement contracts*, deployed on Base, hold ownership records and accept attestations as inputs to reward distribution.

A match proceeds as follows. The client connects to the simulation server and exchanges inputs and state updates over a WebSocket. At match end, the simulation server produces a record of the match: the seed, the sequence of inputs, and the resulting score. The attestation service signs a hash of this record together with the player's identifier. If the player has linked a wallet, the attestation may later be claimed on-chain; if not, the record is stored server-side and may be claimed retroactively if the player links a wallet at any future time. The chain is never on the critical path of gameplay.

3. Identity

Each player has a soldier — a cosmetic identity consisting of a callsign, a visual configuration, and an accumulated history of matches. For unlinked players, the soldier is a row in the operator's database. For linked players, the soldier is additionally an ERC-721 token whose owner is the linked wallet.

Linking is one-directional and revocable. A player connects a wallet and signs a typed message containing their server-side identifier. The operator records the binding. To dissolve the binding, the player signs a revocation message, which is recorded on-chain in the identity contract; after revocation, attestations issued for the prior identifier are no longer claimable by the wallet that signed the revocation.

Minting a soldier as an NFT does not change gameplay. The mint records the soldier's current cosmetic configuration on-chain and freezes its callsign. Subsequent cosmetic changes update the token's metadata; the token's identity persists across changes.

4. Gameplay State

Gameplay state is held by the simulation server. The server receives input events from clients — movement, aim, fire, jump — and advances the simulation at a fixed tick rate. Clients perform local prediction to mask network latency and reconcile against authoritative state when corrections arrive. This is the standard architecture used by competitive online games and is unrelated to the on-chain layer.

The server is responsible for cheat prevention. Inputs that violate physical constraints (impossible movement speed, fire rate exceeding the weapon's specification) are rejected. The simulation is deterministic given a seed and an input sequence, which is the property the attestation layer depends on.

Gameplay state is ephemeral. After a match concludes, only the *result record* is retained: the seed, the compressed input sequence, the final score, and the durations of significant events. The moment-to-moment state is discarded.

5. Replay Attestation

Distributing rewards based on match outcomes raises a verification problem. The chain cannot run the game simulation, but it must not accept arbitrary claims of "I scored 42,000." We resolve this by having the operator sign attestations and by making those attestations independently re-verifiable.

For each completed match, the simulation server produces a tuple $(s, \mathbf{i}, r)$ where s is the random seed, $\mathbf{i}$ is the ordered sequence of player inputs, and r is the result (score, duration, checkpoints). The attestation service computes

$$h = H(s \,\|\, \mathbf{i} \,\|\, r \,\|\, p)$$

where H is a collision-resistant hash function and p is the player's identifier. The service signs h with the operator's attestation key and publishes the signature alongside the result record.

A match is independently verifiable: any party may run the simulation locally on $(s, \mathbf{i})$ and confirm that the resulting score equals r. This is the same property that allows speedrun communities to verify runs from input recordings. Tampering with $\mathbf{i}$ or r produces a different hash and invalidates the signature; tampering with the simulation's behavior (a divergent client) produces a different r when re-simulated and is detectable by any auditor.

The attestation does not commit the chain to the result. It commits only to the statement: "the operator claims this player produced this result from this input sequence." Reward distribution depends on this claim being correct, which is enforced by the social and reputational consequences of being caught issuing false attestations, and by the technical possibility of re-simulation by any sufficiently motivated party.

6. Settlement

Rewards are distributed in seasons. At the end of each season, the operator computes the leaderboard from the season's attested results and constructs a Merkle tree whose leaves are tuples $(\mathit{wallet}, \mathit{amount})$ for each rewarded player. The Merkle root is published on-chain in the rewards contract together with the total amount of $WARC being distributed.

A player claims their reward by submitting their leaf and a Merkle proof. The contract verifies the proof against the published root and transfers the reward to the wallet. Unclaimed rewards from a season expire after a fixed window and return to the treasury.

This pattern is borrowed from existing airdrop infrastructure and has two properties relevant here. First, settlement gas cost is constant per claimer regardless of the size of the leaderboard, so even seasons with hundreds of thousands of participants are tractable. Second, the Merkle root is the only piece of season data the chain stores, which keeps on-chain footprint small.

A player who has not linked a wallet at season end may still link one later and claim, provided the season's claim window has not expired. Attestations carry the server-side identifier; the rewards contract accepts a claim if the linked wallet is the current binding for the identifier in the leaf.

7. Tradable Items

Cosmetic gear — helmets, fatigues, weapon skins — is represented as ERC-1155 tokens. Items are minted into the operator's inventory at manufacture time and transferred to the player's wallet when the player acquires them, either through gameplay rewards, direct purchase, or secondary market trades.

Items are strictly cosmetic. The protocol does not permit gameplay-affecting items to be tokenized, in order to preserve competitive integrity: an attacker cannot purchase an advantage. This constraint is enforced at the client and server level by limiting the set of tokenizable item types.

The marketplace is permissionless. The operator runs an indexer and a front-end for convenience, but trades execute against an open exchange contract on Base, and any third party may build alternative interfaces. Royalties on trades are split between a creator allocation and a burn address, the latter providing one of the token sinks described in Section 9.

8. The Token

$WARC is an ERC-20 token on Base. It is a utility token within the Warfront Legends ecosystem. It does not represent equity, profit-sharing, or any claim against the operator. Its functions are: governance over cosmetic content schedules and treasury allocations, payment for optional cosmetic services, and denomination of seasonal rewards.

The maximum supply is fixed at 10^8 tokens (one hundred million). Allocation is as follows: ninety-five percent to liquidity, and five percent to marketing and development. There is no team allocation, no presale, and no insider distribution at launch.

The token is not necessary to play the game and is not necessary to hold soldier or gear NFTs. It is necessary only to participate in governance votes and to receive seasonal leaderboard rewards.

9. Token Sinks

Without sinks, emissions accumulate indefinitely and the token's purchasing power within the ecosystem decays. The protocol introduces four sinks. First, minting a soldier as an NFT requires a small $WARC payment, of which a fraction is burned and the remainder accrues to the treasury. Second, custom callsigns and clan tags require $WARC payment, fully burned. Third, secondary market trades incur a fee, partially burned. Fourth, optional cosmetic upgrades — recolors, particle effects, animation variants — are priced in $WARC, with a configurable burn ratio set by governance.

Sink magnitudes are calibrated to absorb a target fraction of yearly emissions. The exact calibration is subject to revision and is governed by the same process that controls

cosmetic content schedules.

10. Anti-Cheat

Because the protocol distributes monetary rewards, anti-cheat is treated as a primary concern rather than an operational afterthought.

The simulation runs server-side, so client modifications cannot directly fabricate game events. Inputs that imply impossible behavior are rejected at the input layer. Statistical detection runs over completed matches: distributions of accuracy, reaction time, and movement coherence are monitored, and outliers are flagged for review.

For seasonal reward eligibility, a sample of high-ranking matches is re-simulated by an independent verifier service from the published $(s, \mathbf{i})$ before the Merkle root is committed. Discrepancies between the original r and the re-simulated result invalidate the attestation and cause the player to be excluded from the season's reward set. This re-simulation is bounded: it scales with the number of high-ranking matches, not the total match volume.

Manual review handles edge cases. Bug bounties, paid in $WARC and cash, incentivize external researchers to find vulnerabilities in both the client-server protocol and the smart contracts.

11. Privacy

The system stores the minimum identifying information necessary for gameplay. Wallet linking is opt-in; an unlinked player has no on-chain record. Match attestations contain only an opaque server-side identifier. Wallet binding is recorded on-chain at the player's request but does not reveal the player's gameplay history; the relationship between the identifier and the wallet is observable only to the operator and to the wallet holder themselves.

Replays are stored to permit verification but are not made public by default. A player may opt to publish their replays for community review or speedrun submission. Published replays disclose only the input sequence and the seed, not the player's identity beyond their callsign.

12. Failure Modes

Operator compromise is the dominant failure mode. If the attestation key is exfiltrated, an attacker can issue false attestations for the remainder of the key's validity window. Mitigations include short key rotation periods, threshold signing for high-value attestations, and post-hoc re-simulation of any attestation as a check. None of these prevent a compromise; they bound its impact.

Operator shutdown is the second concern. If the operator ceases to run the service, the gameplay layer disappears and no new attestations can be produced. The on-chain layer survives: existing soldier and gear NFTs remain owned by their holders, the rewards contract continues to honor unclaimed Merkle proofs until the claim window expires, and the token continues to trade on permissionless exchanges. The game is not recoverable

from the chain alone; the gameplay code, assets, and server are required. We do not claim otherwise.

Smart contract bugs are the third concern. Contracts will be audited prior to deployment, and bug bounties will remain active in perpetuity. The contract suite is small and uses well-understood primitives (ERC-721, ERC-1155, ERC-20, Merkle distribution), which limits the surface area for novel vulnerabilities but does not eliminate it.

13. Conclusion

We have described a hybrid game architecture in which the gameplay loop remains off-chain and the ownership layer is on-chain, connected by signed attestations and Merkle-proven settlement. The architecture preserves the latency and feel of a conventional online action game while granting players custody of their cosmetic identity and verifiable entitlement to competitive rewards. The chain is never on the critical path of play, and the wallet is never required to begin playing. The operator retains the responsibilities that operators are good at — running a simulation, balancing a game, distributing content — and sheds the responsibilities that have historically been a source of distrust: arbitrary control over what players own and unilateral authority over what counts as a valid result.

References

[1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008.

[2] W. Entriken et al., "EIP-721: Non-Fungible Token Standard," 2018.

[3] W. Radomski et al., "EIP-1155: Multi Token Standard," 2018.

[4] F. Vogelsteller and V. Buterin, "EIP-20: Token Standard," 2015.

[5] Base, "Base Documentation," `docs.base.org`.

[6] R. C. Merkle, "A Digital Signature Based on a Conventional Encryption Function," *CRYPTO '87*, 1988.