Instruction Specific Address (ISA) Framework

Shubham Singh
Vatsal Gupta
Abhishek Somani

Abstract

Traditional Web3 user flows require connecting wallets, approving token allowances, and signing numerous transactions. This approach can be cumbersome and intimidating for newcomers, ultimately limiting the wider adoption of blockchain. The mechanism proposed in this paper reimagines this process by introducing Instruction Specific Address (ISA), a one-time address generated specifically for each user action. Through deterministically generated addresses, ISAs decouple user wallets from DApps, enabling approval-less, friction-less transactions. This reduces the complexity of approvals, mitigates phishing vectors, and allows a more familiar Web2-like user experience while retaining the trustless nature of blockchain interactions.

Contents

1 Introduction **2**
1.1 Problem Statement 2
1.1.1 Poor UX 2
1.1.2 Security Issues 2
1.2 Defining an ISA 2
1.3 How ISA Solves the Problems 3
1.3.1 Security 3
1.3.2 UX 3

2 ISA Architecture **3**
2.1 Overview 3
2.1.1 ISAFactory Contract: 3
2.1.2 CallOps Structure 3
2.1.3 ExecutionContract (Executor) 3
2.1.4 Relayers 3
2.2 Deterministic Address Generation 4

3 Core Mechanics **4**
3.1 CallOps Data Structure 4

4 Lifecycle of an ISA Interaction **5**

5 Legacy DApp Interaction Flow vs ISA Flow **6**
5.1 Legacy DApp Interaction Flow 6
5.1.1 Wallet Connection 6
5.1.2 Token Approvals 6
5.1.3 Executing the DApp Transaction 6
5.1.4 Result 6
5.2 ISA Flow 7
5.2.1 Instruction Specific Address (ISA) Generation 7
5.2.2 Token Transfer to the ISA 7
5.2.3 Trigger DApp Transaction 7
5.2.4 DApp Transaction Execution 7
5.2.5 Result 8

6 Use Cases **8**
6.1 eCommerce and Merchant Gateways 8
6.2 Decentralized Exchanges (DEXs) 8
6.3 Telegram Mini Apps 8
6.4 Chain-Abstracted DApps 9

7 Security and Risk Management **9**
7.1 Minimal Exposure 9
7.2 Refund Mechanics 9
7.3 Target Contract Considerations 9

8 Roadmap & Future Developments **9**

9 Conclusion **9**

1 Introduction

1.1 Problem Statement

1.1.1 Poor UX

Despite dynamic and visually appealing interfaces, Web 3.0 inherently suffers from poor user experience due to the number of steps required to perform a single transaction. Interacting with a DApp involves connecting a wallet, signing an approval transaction, and then signing the main transaction (such as a swap, stake, etc.).

1.1.2 Security Issues

Interacting with DApps using wallets is highly risky due to approval vulnerabilities, phishing attacks, and other forms of hacks exploited within Web 3.0.

1.2 Defining an ISA

An Instruction-specific Address (ISA) is a deterministically generated single-purpose contract address generated with minimal context needed to execute a specific user operation.

Once the operation completes or fails, the ISA self-destructs, refunding any remaining balance to the designated recipient and freeing up blockchain resources.

1.3 How ISA Solves the Problems

1.3.1 Security

By eliminating direct wallet connections from DApps, ISA makes users far less susceptible to phishing and malicious contract calls. Each ISA is ephemeral (self-destructs after use), thereby limiting the maximum risk of the user to the specific amount transferred for that interaction.

1.3.2 UX

Users can 'scan and pay' in a Web2-like manner, simply transferring assets to an address for immediate use without plug-in-based wallet connections. This method is also compatible with centralized exchanges (CEX) wallets, bridging the gap between custodial and non-custodial experiences.

2 ISA Architecture

2.1 Overview

The architecture of ISA consists of the following components:

2.1.1 ISAFactory Contract:

This contract generates deterministic addresses via `CREATE2` and enforces a consistent initialization code. It exposes a `generateAddress()` function that takes a unique `salt` and ISA contract bytecode.

2.1.2 CallOps Structure

This is a standardized object that contains the target contract , source token, execution data, potential approvals and a fallback `refundRecipient`. The CallOps structure ensures a consistent interface that wallets, DApps, and relayers can rely upon.

2.1.3 ExecutionContract (Executor)

As the name suggests, the ExecutionContract executes the desired logic and refunds unspent balances.

2.1.4 Relayers

These are entities that process ISA operations and maintain fluid user interactions.

2.2 Deterministic Address Generation

ISA relies on the `CREATE2` opcode to guarantee unique, predictable addresses. This process hashes together:

- The sender (factory) address
- A user-provided `salt`
- The initialization code's hash

3 Core Mechanics

3.1 CallOps Data Structure

```
struct CallOps {
    address targetAddress;
    address approvalAddress;
    bytes executionData;
    address sourceToken;
    address refundRecipient;
    uint256 amount;
    }
```

- **targetAddress:** Contract to interact with.
- **approvalAddress:** If token transfers need an approval step.
- **executionData:** Encoded call data for logic execution.
- **sourceToken:** The token used in the operation.
- **refundRecipient:** Address receiving leftover funds upon failure.
- **amount:** Amount of the source tokens to be sent by the user.

4 Lifecycle of an ISA Interaction

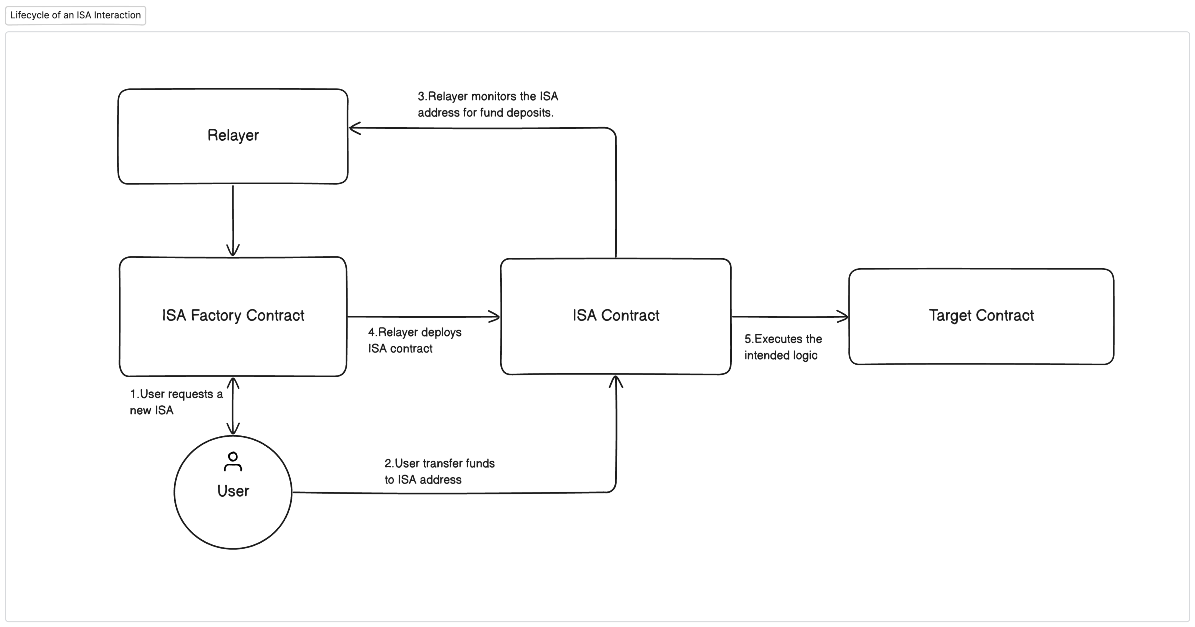
Lifecycle of an ISA Interaction
Relayer
3.Relayer monitors the ISA address for fund deposits.
ISA Factory Contract
4.Relayer deploys ISA contract
ISA Contract
5.Executes the intended logic
Target Contract
1.User requests a new ISA
User
2.User transfer funds to ISA address

Figure 1: ISA Lifecycle Diagram

Below is a high-level sequence of a typical ISA transaction lifecycle:

1. **User Input and Address Generation:** The user specifies the action to perform and requests a new ISA from the ISAFactory.
2. **Transfer of Funds:** The user sends the required assets to the ISA address.
3. **Execution:** A relayer monitors ISA deposits; once a deposit is detected to the ISA, it deploys the contract, and executes the logic.
4. **Refund on Failure:** If the operation fails, the ISA returns the unspent assets to the `refundRecipient`.
5. **Self-Destruct:** Regardless of success or failure, the ISA self-destructs, ensuring one-time usage and preventing future exploit vectors.

5 Legacy DApp Interaction Flow vs ISA Flow

5.1 Legacy DApp Interaction Flow

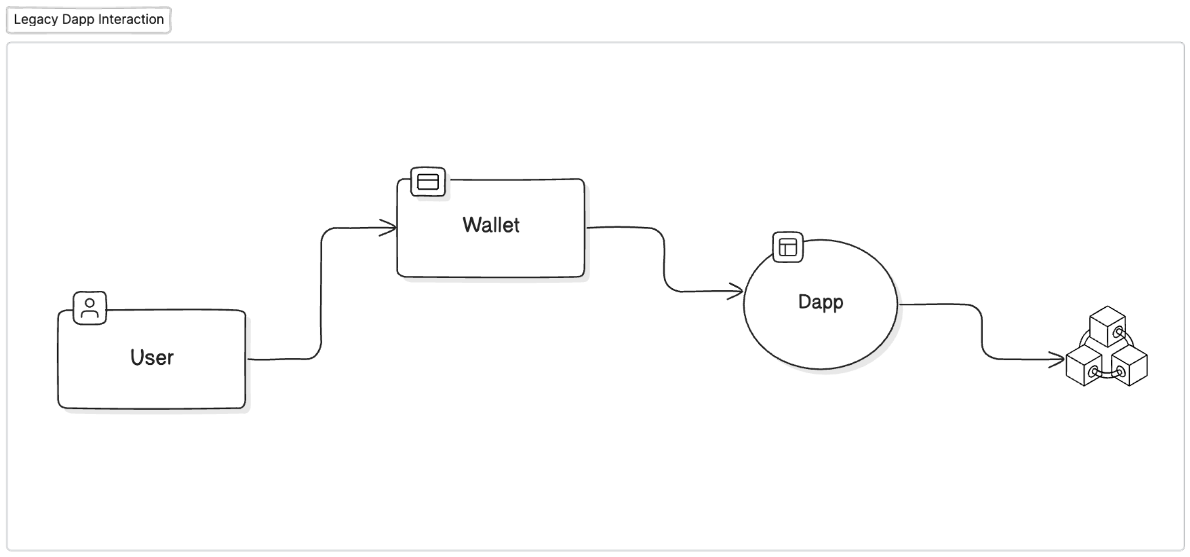

Figure 2: Legacy Flow Diagram

5.1.1 Wallet Connection

The user first has to connect their wallet (e.g., MetaMask, Coinbase wallet) to the DApp. This often involves navigating through a browser plug-in or a mobile app and granting the DApp access to the wallet address.

5.1.2 Token Approvals

Before using any DApp functionality (e.g., swapping tokens, purchasing NFTs), the user must approve the DApp's smart contract to spend tokens on their behalf. These approvals are often infinite, covering an entire token balance unless manually limited. Approvals involve an additional on-chain transaction that the user must confirm in their wallet, adding friction.

5.1.3 Executing the DApp Transaction

Once approvals are set, the user can initiate the main DApp transaction (e.g., token swap, mint, deposit) through the wallet. The DApp transaction is broadcasted to the blockchain, and the user must confirm the transaction details again in their wallet.

5.1.4 Result

If everything succeeds, the DApp updates user balances or triggers the desired action on-chain. If the DApp requires more interactions, the user may need additional approvals or confirmations.

5.2 ISA Flow

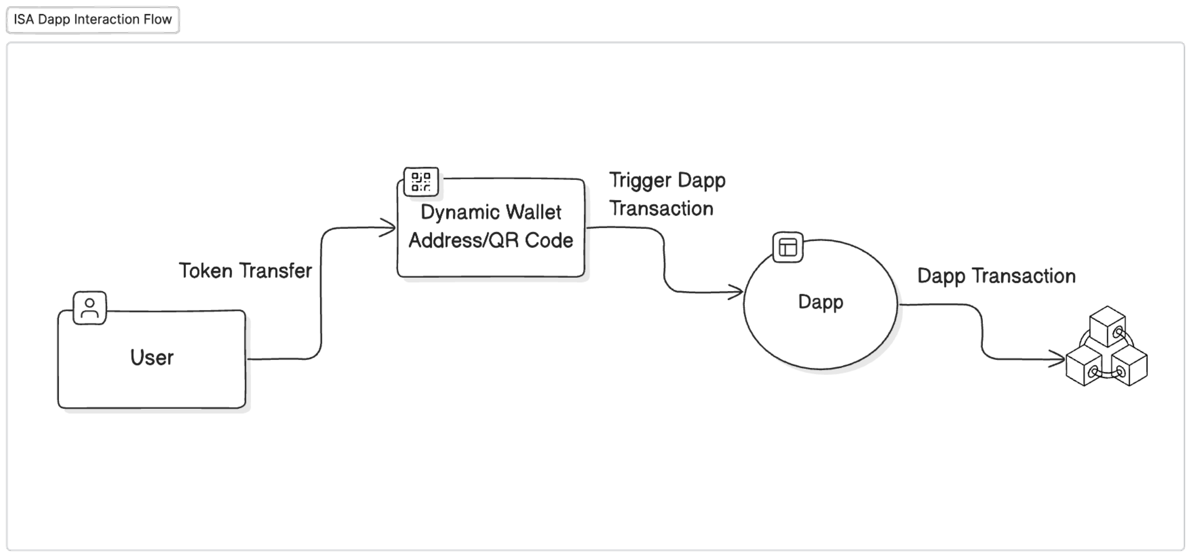

Figure 3: ISA Flow Diagram

5.2.1 Instruction Specific Address (ISA) Generation

Instead of connecting a wallet directly to the DApp, the user is provided a single-use contract address - this address can be displayed on the UI as a QR code or as a plain address.

5.2.2 Token Transfer to the ISA

The user sends the exact amount of tokens required for the intended DApp interaction directly to the ISA. No DApp-specific “approval” transaction is needed. The transferred tokens are effectively “pre-approved” for the DApp logic by virtue of living inside this dedicated address.

5.2.3 Trigger DApp Transaction

Once the ISA receives the tokens, a relayer deploys the ISA contract, and the designated function or operation is automatically triggered. The ISA contract acts as the executor of the transaction, sending the tokens to the target contract according to the user’s instructions. Since the relayer is typically responsible for deploying the ISA contract, there is a risk of failure if the relayer does not trigger it. To circumvent this issue and ensure trustlessness, anyone can trigger the factory contract function to deploy the ISA contract and consequently execute the designated calldata.

5.2.4 DApp Transaction Execution

The DApp contract receives the tokens from the ISA and completes the user’s request. On completion, the ISA self-destructs, returning any unused funds to a designated refund recipient.

5.2.5 Result

The user's desired action is carried out on-chain, all without requiring 'token approvals' from a personal wallet. The entire user experience is streamlined to a single deposit action.

Advantages of ISA	**Description**
Approval-less	Users do not have to grant indefinite token allowances; each ISA interaction involves only the specific assets needed for that transaction.
Enhanced Security	The maximum capital at risk is confined to the assets transferred into the ISA. DApps have no direct access to user wallets, drastically reducing the impact of phishing or malicious calls.
Frictionless UX	Feels like a Web2 payment flow ("scan and pay"). Reduces the barrier for non-crypto-native users who prefer a simple transfer to an address, rather than installing plugins or signing multiple transactions.
Broad Compatibility	Works with centralized exchange (CEX) wallets. Operates seamlessly across EVM-based blockchains and can be extended to non-EVM chains with the same principle.

Table 1: Advantages of ISA

6 Use Cases

6.1 eCommerce and Merchant Gateways

- **Seamless Crypto Payments:** Accept payments in crypto (native or tokens) without requiring customers to manage multiple approval steps.
- **Automated Workflows:** The merchant's DApp or business logic can fetch real-time status updates by polling the status from the contract or from a third party service. Once it receives the confirmation of deposit, order fulfillment or other workflow automations can be triggered.

6.2 Decentralized Exchanges (DEXs)

Users can perform token swaps by transferring tokens to an ISA that automatically handles execution logic (e.g., routing, slippage management). Since ISA eliminates the need for prior token allowances, user friction is reduced.

6.3 Telegram Mini Apps

ISA can underpin Telegram mini apps allowing users to perform crypto transactions directly from lightweight interfaces. By enabling in-app transactions, projects can ensure their users never have to leave their interface even for complex DeFi actions like staking and yield mining.

6.4 Chain-Abstracted DApps

Projects can design front-ends that abstract away specific chain interactions. Users need only specify their 'intent' (e.g., swap, purchase) and send assets to an ISA.

7 Security and Risk Management

7.1 Minimal Exposure

The temporary nature of ISAs ensures that any vulnerabilities or exploits cannot persist. Once the transaction completes, the address self-destructs. Users' primary wallets remain protected. If a DApp is malicious, it only receives the specific funds in the ISA.

7.2 Refund Mechanics

Each ISA has a fallback address in case of failures, ensuring that the unused assets are sent back to the user. The `ExecutionFailed` and `RefundIssued` events provide an auditable trail of any returned assets.

7.3 Target Contract Considerations

Target contracts that rely on `msg.sender` for core logic or ownership checks should ensure they handle calls from any address (in this case, the ISA). DApps integrating the ISA framework should also be audited to ensure robust parameter validation and secure fund-handling logic.

8 Roadmap & Future Developments

- **Relayer Ecosystem:** Building standardized frameworks for multiple relayers to compete or collaborate in providing gas and execution services.
- **Developer Tooling & SDK:** Providing out-of-the-box libraries and developer tools to make ISA deployments straightforward, including minimal code for factories, initialization logic, and UI components.
- **Security Modules & Insurance:** Potential add-on modules for additional coverage or bonded relayers that guarantee execution fidelity.

9 Conclusion

The Instruction Specific Address (ISA) standard offers a simple approach to enable user interactions within the decentralized ecosystem. By limiting user risk, removing the need for approvals, and providing a frictionless payment-flow experience, ISAs align Web3 with a more intuitive "Web2-like" product philosophy—while retaining trustlessness and composability. Furthermore, the standard is designed with a clear path to bridge the gap between custodial and non-custodial wallets, benefiting the entire ecosystem.

References

- Ethereum Magicians Discussions: ERC-7838 Discussion Thread
- RFC 2119: Key Words for Use in RFCs
- EIP-20: ERC-20 Token Standard
- CREATE2 Opcode Documentation: Ethereum Yellow Paper / EVM Opcodes Reference