SIDANLAB

Security Audit Report

Date: September 21, 2025

Project: FluidSwaps Protocol

Version: 1.0

Table of Content

1. Executive Summary

Regarding the review of Atomic Swap implementation, the Bitcoin script and Solidity smart contracts that are used are both fairly straightforward, and appear to be correctly implemented. While this audit ended up covering these two contracts, the reader should be made aware that the auditor's expertise is primarily in Cardano and its script system.

Regarding the review of global.ak, "Smart Accounts", the "Smart Account" scripts are used such that Bitcoin and Ethereum wallets can sign specific messages that are validated on the Cardano chain. These messages are instructions for the Cardano chain, as an example, "send x number of y tokens to z address" would be signed by the Bitcoin/Ethereum wallet, then the resulting signature would be used to approve a particular transaction on the Cardano blockchain. This effectively allows Bitcoin/Ethereum users to own any Cardano assets within these scripts.

1.1 Vulnerability Summary

Severity	Count	Resolved	Mitigated	Partially Resolved	Acknowledged/Declined
Critical	1	0	0	0	0
Major	0	0	0	0	0
Medium	0	0	0	0	0
Minor	0	0	0	0	0
Informational	2	0	0	0	0

Notes: Severity definitions are provided in Appendix A.

2. Project Overview

Protocol Type	Multi-chain Swap System
Key Components	The FluidSwaps protocol is a multi-chain system, allowing swaps between Cardano, Bitcoin and Ethereum tokens. There are a few elements of blockchain systems that FluidSwaps utilizes to allow this. Namely, the ability for cryptographic signatures to be verified across chains, and atomic swap implementations, which uses time-lock systems together with signature schemes to allow secure swaps. FluidSwaps also employs an idea they call "smart accounts" to allow Ethereum and Bitcoin private keys to sign Cardano transactions, effectively allowing Ethereum and Bitcoin wallets to own Cardano assets. Together with allowing swaps between the chains, the protocol effectively brings the ecosystems together.
User Roles	Users will be traders across multiple chains, the protocol is fully decentralized, there doesn't appear to be any admin roles.
External Integrations	No external integrations, atomic swap implementation is fully self sufficient

This audit is intended to review whether or not the smart contracts implemented in FluidSwaps effectively achieves its goals in a secure manner. While care has been taken to cover security issues, there is no guarantee that all security problems are uncovered.

3. Codebase & Audit Scope

List repositories, commits, files audited, and anything explicitly out of scope.

3.1 Repositories & Commits

Repository	Commit (hash)	Link
ft-cardano-fluidswaps-sc	d5668e085ed000a172cda022d6d4d0c978978adb	https://github.com/FluidTokens/ft-cardano-fluidswaps-sc

3.2 Files Audited

Path	Component	Commit
[validators/global.ak]	Smart accounts	d5668e085ed000a172cda022d6d4d0c978978adb
[validators/swap.ak]	Atomic swap (Cardano)	d5668e085ed000a172cda022d6d4d0c978978adb
[Bitcoin/swap.script]	Atomic Swap (Bitcoin)	d5668e085ed000a172cda022d6d4d0c978978adb
[Ethereum/swap.sol]	Atomic Swap (Ethereum)	d5668e085ed000a172cda022d6d4d0c978978adb

3.3 Out-of-Scope Dependencies & Assumptions

• The accuracy of the Aiken compiler to UPLC is out of scope of this audit
• The correctness of the Aiken standard library is out of the scope of this audit
Assumptions: The correctness of the Aiken, Solidity and Bitcoin script language is assumed.
Recommendations: For the most part, the protocol doesn't use any external libraries, and the used programming languages for smart contracts are most likely reliable.

The only chain that has any real options for a different language is Cardano, where Aiken is not the only option. However, Aiken is extremely popular and widely regarded as the recommended smart contract language for Cardano.

4. Privileged Roles & Centralization Review

The protocol is very well decentralized and there are no special privileged roles. Swaps are handled entirely by atomic swaps, and smart accounts are fully controlled by the user's keys.

5. Detailed Findings

5.1 [FND-01] Transaction Created Before The Deadline Can be Cancelled in Atomic Swap implementation

Severity	Critical
Status	Resolved

Description

The function **validate_tx_end** appears to be implemented incorrectly, the upper bound of the **transaction_validity_range** is asserted to be less than the deadline, this allows any transaction created BEFORE the deadline to be cancelled, which is opposite of the expected behaviour. Swaps should only be allowed to be cancelled AFTER the deadline.

Relevant code:

```
Cancel -> {

  expect validate_tx_end(self, deadline) == True

  list.has(self.extra_signatories, owner_ada_pubkey_hash)

}



pub fn validate_tx_end(tx: Transaction, deadline: Int) {

  trace cbor.diagnostic(deadline)

  when tx.validity_range.upper_bound.bound_type is {

    Finite(end) -> end < deadline

    _ -> False

  }

}
```

Recommended Action:

We should be checking that the transaction is created AFTER the deadline. In which case, checking the transaction's upper bound is actually incorrect. The script should instead be checking for the transaction's lower bound, ensuring that this lower bound is larger than the deadline. This ensures that the transaction cannot be valid until AFTER the deadline (the transaction only begins to be valid AFTER the deadline).

```
Cancel -> {
  expect validate_tx_end(self, deadline) == True
  list.has(self.extra_signatories, owner_ada_pubkey_hash)
}


pub fn validate_tx_end(tx: Transaction, deadline: Int) {
  trace cbor.diagnostic(deadline)
  when tx.validity_range.lower.bound_type is {
    Finite(begin) -> begin > deadline
    _ -> False
  }
}
```

Resolution:
The team updated the code to correctly check that the “Cancel” transaction has a validity range with a lower bound larger than the deadline. This correctly checks that the “Cancel” transaction occurs after the deadline has passed. This was resolved in commit [df76542b43f0b03d5d88ce41177ae328c26a52e2]

5.2 [FND-02] Absence of test cases related to “Smart Accounts”

Severity	Informational
Status	Resolved

Description

These "Smart Accounts" have a large amount of logic, but seemingly no test cases can be found within the repository, with such a large amount of logic, it is difficult to reason about the security of the scripts.

Recommended Action:

Add suitable testing for these scripts. Some basic testing with generated signatures at least would be recommended.

Resolution:

The team added some basic tests with signatures from generated messages, and asserted that they produced correct results. The tests can be found in commit [df76542b43f0b03d5d88ce41177ae328c26a52e2]

5.3 [FND-03] The code of “Smart Accounts” are inefficient

Severity	Informational
Status	Resolved

Description

The code within **global.ak** is quite inefficient, a large portion of it is fairly redundant. It appears that there are a large number of separate fields that are signed by the Bitcoin/Ethereum wallets, and the fields are validated by reconstructing what is assumed to be the user's intention.

However, the fields **policy, assetname, amount, spendingscriptflag, pubkey, stakingscriptflag, stakekey, datumflag, datum, payment_index** are all used to simply reconstruct a singular output at **payment_index** that fulfills all of these constraints. It is completely possible to make the user sign a single cbor encoded **Output** to encompass this entire logic. The only thing that would make this a little more difficult is the **min_utxo_value** required in this **Output**, but this can be accounted for by not checking the **lovelace** value of the **Output**.

Relevant code:

```
Send {
  signature,
  utxos,
  policy,
  assetname,
  amount,
  policyInput,
```

```
  assetnameInput,
  amountInput,
  spendingscriptflag,
  pubkey,
  stakingscriptflag,
  stakekey,
  datumflag,
  datum,
  payment_index,
  fee,
  signer,
} -> {
  let list_utxos =
    list.foldr(
      utxos,
      [],
      fn(utxo, xs) {
        [
          utxo.transaction_id,
          bytearray.from_int_big_endian(utxo.output_index, 1),
          ..xs
```

```
        ]
      },
    )
expect datum_converted: ByteArray = datum
let message_array: List<ByteArray> =
  list.push([], bytearray.from_int_big_endian(fee, 20))
    |> list.push(bytearray.from_int_big_endian(payment_index, 1))
    |> list.push(datum_converted)
    |> list.push(bytearray.from_int_big_endian(datumflag, 1))
    |> list.push(stakekey)
    |> list.push(bytearray.from_int_big_endian(stakingscriptflag, 1))
    |> list.push(pubkey)
    |> list.push(bytearray.from_int_big_endian(spendingscriptflag, 1))
    |> list.push(bytearray.from_int_big_endian(amountInput, 20))
    |> list.push(assetnameInput)
    |> list.push(policyInput)
    |> list.push(bytearray.from_int_big_endian(amount, 20))
    |> list.push(assetname)
    |> list.push(policy)
```

```
  let concat_array = concat_array(list.concat(list_utxos,
message_array))

  let message_hash_btc = get_hash_from_bytearray_btc(concat_array)

  let message_hash_eth = get_hash_from_bytearray_eth(concat_array)

  //TODO da checkare

  //verify I am sending the right amount to the right user from the data
i have in the first output

  //verify the second output goes to the starting one - the amount it
had - 1 ADA

  //address validation receiver is missing

  let sending_value = from_asset(policy, assetname, amount)

  let incoming_value =

    if amountInput > 0 {

      from_asset(policy, assetname, amount)

    } else {

      zero

    }


  expect Some(contract_input) = find_input(self.inputs, own_ref)

  expect Some(payment_output) = list.at(self.outputs, payment_index)


  let starting_value =
```

```
    find_inputs_value(self.inputs, contract_input.output.address)
  let final_value =
    find_outputs_value(self.outputs, contract_input.output.address)


  let flag_utxo_present = list.has(utxos, own_ref)


  let basic_validation = and {
      flag_utxo_present,
      or {
        validate_signature_key(pubKey, message_hash_btc, signature),
        validate_signature_key(pubKey, message_hash_eth, signature),
      },
    }
  let expected_datum = calculate_datum(datumflag, datum)
  let expected_address =
    calculate_address(
      spendingscriptflag,
      stakingscriptflag,
      pubkey,
      stakekey,
    )
```

```
and {
  quantity_of(payment_output.value, policy, assetname) == quantity_of(
    sending_value,
    policy,
    assetname,
  ),
  assets.match(
    final_value,
    merge(
      merge(
        merge(starting_value, negate(sending_value)),
        incoming_value,
      ),
      negate(from_lovelace(fee)),
    ),
    >=,
  ),
  basic_validation,
  payment_output.address == expected_address,
  payment_output.datum == expected_datum,
  all_inputs_present(utxos, self.inputs),
```

```
    only_external_signer_inputs_extra(utxos, self.inputs, signer),
  }
}
```

Recommended Action:

It is possible to reduce the amount of code significantly, and also increase the user's fine control over the outputs, it would even be possible to allow users to send multiple outputs with something like this:

```
Send { signature, utxos, outputs, signer } -> {
  let message_array: List<ByteArray> =
    list.push([], cbor.serialise(utxos))
      |> list.push(cbor.serialise(outputs))

  let message_hash_btc =
    get_hash_from_bytearray_btc(cbor.serialise(message_array))
  let message_hash_eth =
     get_hash_from_bytearray_eth(cbor.serialise(message_array))

  let basic_validation = or {
      validate_signature_key(pubKey, message_hash_btc, signature),
      validate_signature_key(pubKey, message_hash_eth, signature),
```

```
    }


  and {
    basic_validation,
    all_outputs_present(outputs, self.outputs),
    all_inputs_present(utxos, self.inputs),
    only_external_signer_inputs_extra(utxos, self.inputs, signer),
  }
}
```

Resolution:
Upon further discussion with the team, it appears that the recommended solution wasn't possible due to the presence of the "Outputs" type in a redeemer. This contains the "Value" type which cannot appear in datums or redeemers, due to their structure being unpredictable, and can cause security issues if forcibly used as a type in redeemers and datums. However, it was concluded that a similar behaviour can be achieved by using the hash of the outputs, which bypasses this limitation completely. The final implementation is significantly simplified and easier to reason about, and can be found in commit [d5668e085ed000a172cda022d6d4d0c978978adb](d5668e085ed000a172cda022d6d4d0c978978adb).

6. Disclaimer

This report reflects our assessment of the code provided during the audit window and does not constitute investment, legal, tax, or regulatory advice. Findings are based on limited

scope and may include false positives/negatives; subsequent code changes, third-party components, and deployment choices are out of scope. No warranty is given that the software is error-free or secure, and reliance by any person other than the Client is prohibited. Use of this report is governed by the Services Agreement, including confidentiality and limitations of liability.

7. About the Auditor

A technical solution provider, specializing in Cardano development. SIDAN Lab builds open-source toolings, decentralized applications for the Cardano community, and operates the SIDAN stake pool.

Website: https://sidan.io

Github: https://github.com/sidan-lab

Email Address: contact@sidan.io

Appendix A — Severity Definitions

Severity	Description
Critical	Issues that can directly jeopardize funds or protocol control—for example enabling theft, freezing user assets, or irrevocably locking value. These must be fixed before deployment or continued operation.
Major	High-impact problems that can damage users or the protocol, often within specific features or time windows. Exploitation can disrupt core behavior or allow significant misuse until corrected.
Medium	Vulnerabilities unlikely to cause immediate loss of funds but capable of degrading functionality, reliability, or user experience. Examples include conditions that enable transaction manipulation or temporary denial of intended actions.
Minor	Low-impact flaws with limited blast radius. Exploitation typically requires uncommon conditions, yields minimal advantage to an attacker, and poses little risk to protocol integrity or user funds.
Informational	Non-security or best-practice findings that don't introduce direct risk. These cover code clarity, documentation gaps, naming/style consistency, architectural suggestions, or optional performance refinements.

Appendix B — Change Log

Date	Version	Changes
2025-09-15	1.0	Initial Audit Report
2025-11-04	2.0	Resolutions from the FluidSwaps team