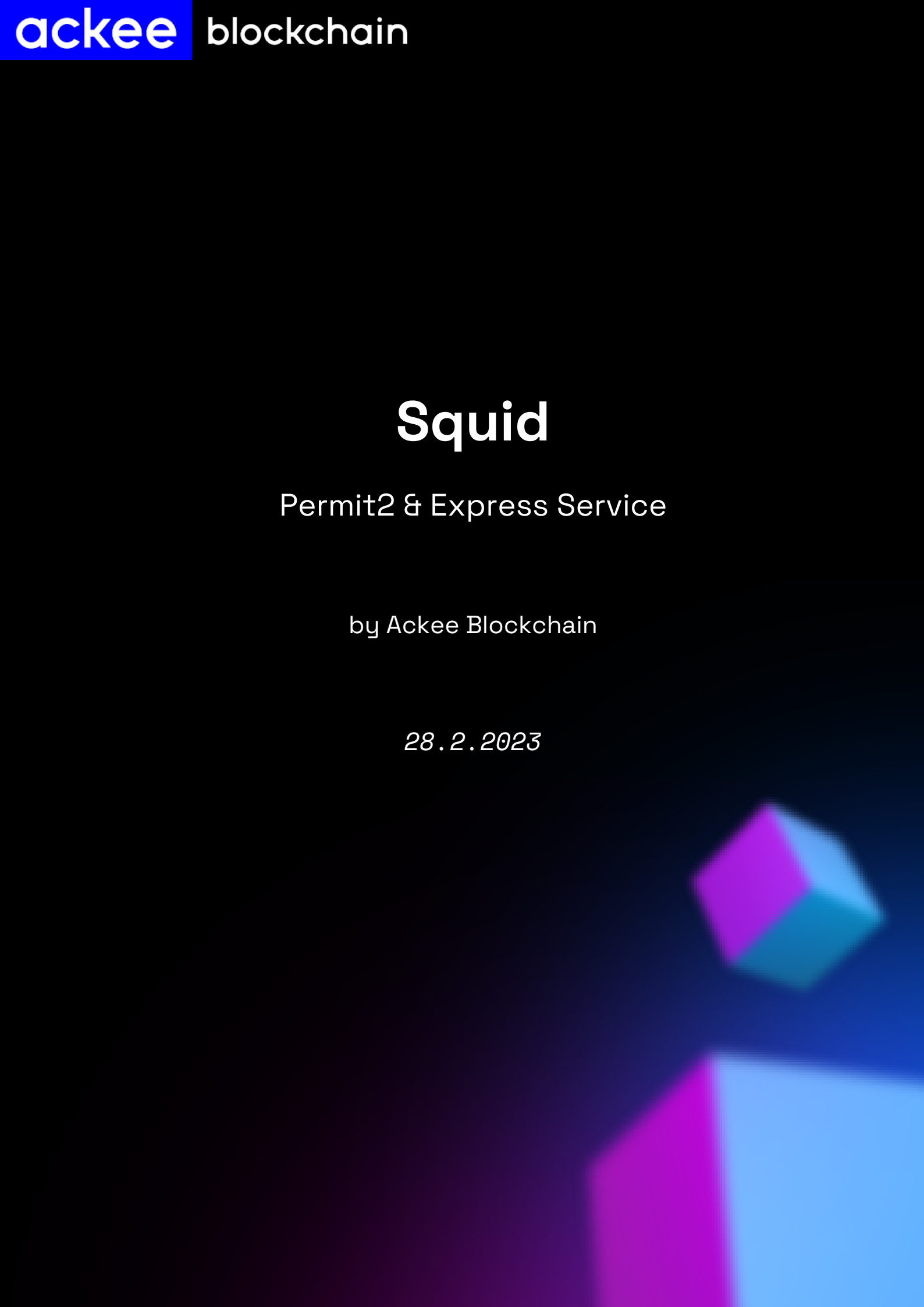
ackee blockchain

Squid

Permit2 & Express Service

by Ackee Blockchain

28.2.2023



Contents

1. Document Revisions

1.0	Final report	February 28, 2023

2. Overview

This document presents our findings in reviewed contracts.

2.1. Ackee Blockchain

Ackee Blockchain is an auditing company based in Prague, Czech Republic, specializing in audits and security assessments. Our mission is to build a stronger blockchain community by sharing knowledge – we run free certification courses School of Solana, Summer School of Solidity and teach at the Czech Technical University in Prague. Ackee Blockchain is backed by the largest VC fund focused on blockchain and DeFi in Europe, RockawayX.

2.2. Audit Methodology

1. **Technical specification/documentation** - a brief overview of the system is requested from the client and the scope of the audit is defined.
2. **Tool-based analysis** - deep check with automated Solidity analysis tools and Woke is performed.
3. **Manual code review** - the code is checked line by line for common vulnerabilities, code duplication, best practices and the code architecture is reviewed.
4. **Local deployment + hacking** - the contracts are deployed locally and we try to attack the system and break it.
5. **Unit and fuzzy testing** - run unit tests to ensure that the system works as expected, potentially write missing unit or fuzzy tests.

2.3. Finding classification

A *Severity* rating of each finding is determined as a synthesis of two sub-ratings: *Impact* and *Likelihood*. It ranges from *Informational* to *Critical*.

If we have found a scenario in which an issue is exploitable, it will be assigned an impact rating of *High*, *Medium*, or *Low*, based on the direness of the consequences it has on the system. If we haven't found a way, or the issue is only exploitable given a change in configuration (such as deployment scripts, compiler configuration, use of multi-signature wallets for owners, etc.) or given a change in the codebase, then it will be assigned an impact rating of *Warning* or *Info*.

Low to *High* impact issues also have a *Likelihood*, which measures the probability of exploitability during runtime.

The full definitions are as follows:

Severity

		Likelihood			
		High	**Medium**	**Low**	**-**
Impact	**High**	Critical	High	Medium	-
	Medium	High	Medium	Medium	-
	Low	Medium	Medium	Low	-
	Warning	-	-	-	Warning
	Info	-	-	-	Info

Table 1. Severity of findings

Impact

- **High** - Code that activates the issue will lead to undefined or catastrophic consequences for the system.
- **Medium** - Code that activates the issue will result in consequences of serious substance.
- **Low** - Code that activates the issue will have outcomes on the system that are either recoverable or don't jeopardize its regular functioning.
- **Warning** - The issue cannot be exploited given the current code and/or configuration (such as deployment scripts, compiler configuration, use of multi-signature wallets for owners, etc.), but could be a security vulnerability if these were to change slightly. If we haven't found a way to exploit the issue given the time constraints, it might be marked as a "Warning" or higher, based on our best estimate of whether it is currently exploitable.
- **Info** - The issue is on the borderline between code quality and security. Examples include insufficient logging for critical operations. Another example is that the issue would be security-related if code or configuration (see above) was to change.

Likelihood

- **High** - The issue is exploitable by virtually anyone under virtually any circumstance.
- **Medium** - Exploiting the issue currently requires non-trivial preconditions.
- **Low** - Exploiting the issue requires strict preconditions.

2.4. Review team

Member's Name	Position
Jan Kalivoda	Lead Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

2.5. Disclaimer

We've put our best effort to find all vulnerabilities in the system, however our findings shouldn't be considered as a complete list of all existing issues. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

3. Executive Summary

Squid is a protocol that allows cross-chain swaps. The new Squid Router implements Axelar's Express Service and Permit2 approach for token transfers.

Revision 1.0

Axelar engaged Ackee Blockchain to perform a security review of the new features in Squid Router with a total time donation of 3 engineering days in a period between February 22 and February 27, 2023, and the lead auditor was Jan Kalivoda.

The audit has been performed on the commit `4fcb32f` and the scope was the following:

- SquidRouter.sol

We began our review by using static analysis tools, namely Woke. We then took a deep dive into the logic of the contracts. During the review, we paid special attention to:

- correct implementation of the Express Service,
- correct usage of the Permit2 approach,
- checking if nobody can maliciously replay signatures,
- ensuring the arithmetic of the system is correct,
- detecting possible reentrancies in the code,
- ensuring access controls are not too relaxed or too strict,
- looking for common issues such as data validation.

Our review resulted in 7 findings, ranging from Info to High severity. The most

severe one is H1: Non-standard ERC20 tokens can not be used that is causing DoS for some tokens.

Ackee Blockchain recommends Squid:

- add NatSpec documentation to all functions,
- address all other reported issues.

See Revision 1.0 for the system overview of the codebase.

4. Summary of Findings

The following table summarizes the findings we identified during our review. Unless overridden for purposes of readability, each finding contains:

- a *Description*,
- an *Exploit scenario*,
- a *Recommendation* and if applicable
- a *Solution*.

There might often be multiple ways to solve or alleviate the issue, with varying requirements regarding the necessary changes to the codebase. In that case, we will try to enumerate them all, clarifying which solves the underlying issue better (albeit possibly only with architectural changes) than others.

	Severity	Reported	Status
H1: Non-standard ERC20 tokens can not be used	High	1.0	Reported
W1: Usage of `solc` optimizer	Warning	1.0	Reported
W2: Fees are bypassed if the contract's balance is zero	Warning	1.0	Reported
W3: Pitfalls of Express Service	Warning	1.0	Reported
W4: Permit2 address can not be changed after deploy	Warning	1.0	Reported
I1: Contract id based validation	Info	1.0	Reported

	Severity	Reported	Status
I2: Missing NatSpec documentation	Info	1.0	Reported

Table 2. Table of Findings

5. Report revision 1.0

5.1. System Overview

This section contains an outline of the audited contracts. Note that this is meant for understandability purposes and does not replace project documentation.

Contracts

Contracts we find important for better understanding are described in the following section.

SquidRouter

The contract inherits from ExpressExecutable which is a requirement to use Axelar's Express Service. Also, the contract is Upgradable (Axelar's contract for upgradeability) and RoledPausable (Pauser role can pause specific functions). The contract can perform various calls important for cross-chain swaps.

SquidRouter optionally performs a user-defined multicall on a source chain (through the SquidMulticall contract), sends a token through the Axelar Gateway, and then optionally performs another user-defined multicall on a destination chain. It is also supported to perform a multicall without sending tokens through the Axelar Gateway.

Actors

This part describes actors of the system, their roles, and permissions.

Pauser

A pauser of the SquidRouter contract can pause all public functions intended for swaps.

Owner

An owner of the SquidRouter contract (proxy) can deploy a new implementation of the SquidRouter contract.

5.2. Trust model

Users have to trust Owner that he/she will correctly set up the contract parameters and will not change maliciously the contract implementation.

H1: Non-standard ERC20 tokens can not be used

High severity issue

Impact:	Medium	Likelihood:	High
Target:	SquidRouter.sol	Type:	DoS

Description

The contract is doing token approvals for the gateway in the following way:

```
function _approve(address tokenAddress, address spender, uint256 amount)
private {
    if (IERC20(tokenAddress).allowance(address(this), spender) < amount) {
        // Not a security issue since the contract doesn't store tokens
        IERC20(tokenAddress).approve(spender, type(uint256).max);
    }
}
```

For example, USDT has a different `approve` function than is specified in the IERC20 interface, so **the transaction reverts always with an incorrect ABI decoding** because IERC20 expects a return value (bool).

```
function approve(address spender, uint256 amount) external returns (bool);
```

Moreover, the approval is not restored to zero in any case. This can cause an issue for non-standard ERC20 tokens when `approve` needs to be called, like USDT. USDT needs an allowance set to zero to approve it to another non-zero value and thus, in this case, the transaction fails.

Exploit scenario

The `bridgeCall` function is called but the transaction reverts with an incorrect ABI decoding (see Appendix C).

Recommendation

Refactor the approval system to be always set only for the needed value to transfer and use the ”safe” functions from OpenZeppelin. For example, the `_approve` function can be refactored to:

```
function _approve(address tokenAddress, address spender, uint256 amount)
private {
    IERC20(tokenAddress).safeApprove(spender, 0);
    IERC20(tokenAddress).safeApprove(spender, amount);
}
```

This will ensure the most compatibility with other tokens.

Go back to Findings Summary

W1: Usage of `solc` optimizer

Impact:	Warning	Likelihood:	N/A
Target:	`**/*`	Type:	Compiler configuration

Description

The project uses `solc` optimizer. Enabling `solc` optimizer may lead to unexpected bugs.

The Solidity compiler was audited in November 2018, and the audit concluded that the optimizer may not be safe.

Vulnerability scenario

A few months after deployment, a vulnerability is discovered in the optimizer. As a result, it is possible to attack the protocol.

Recommendation

Until the `solc` optimizer undergoes more stringent security analysis, opt-out using it. This will ensure the protocol is resilient to any existing bugs in the optimizer.

Go back to Findings Summary

W2: Fees are bypassed if the contract's balance is zero

Impact:	Warning	Likelihood:	N/A
Target:	SquidRouter.sol	Type:	Logic error

Description

In the `_bridgeCall` function is call to the Axelar Gateway and in prior of that, fees should be paid to the Axelar Gas Service. However, if the contract's balance is zero, the fees are not paid.

```
if (address(this).balance > 0) {
    if (express) {
        gasService.payNativeGasForExpressCallWithToken{value: address(
this).balance}(
            address(this),
            destinationChain,
            destinationContractAddress,
            payload,
            bridgedTokenSymbol,
            bridgedTokenBalance,
            refundRecipient
        );
    } else {
        gasService.payNativeGasForContractCallWithToken{value: address(
this).balance}(
            address(this),
            destinationChain,
            destinationContractAddress,
            payload,
            bridgedTokenSymbol,
            bridgedTokenBalance,
            refundRecipient
        );
    }
}
```

So any user can choose to send zero value to the contract and bypass the fees.

Moreover, there is no difference if `express` is true or false.

Recommendation

Ensure this is wanted behavior.

Go back to Findings Summary

W3: Pitfalls of Express Service

Impact:	Warning	Likelihood:	N/A
Target:	SquidRouter.sol	Type:	Logic error

Description

SquidRouter inherits from the ExpressExecutable contract that is itself vulnerable to insufficient data validation. The safety of the logic contract depends on a proxy contract. The proxy contract has to shadow publicly-accessible entrypoints for executions on a source chain (`execute` and `executeWithToken`).

Recommendation

Ensure that you will use Axelar's ExpressProxy contract as a proxy.

Go back to Findings Summary

W4: Permit2 address can not be changed after deploy

Impact:	Warning	Likelihood:	N/A
Target:	SquidRouter.sol	Type:	Logic error

Description

Permit2 address can be set only in the constructor. So if the contract is deployed with Permit2 address as a zero-address, the Permit2 feature can not be later enabled. Or respectively, if the Permit2 address is set to the correct address from the beginning, it can not be disabled later.

Recommendation

Ensure the address doesn't need to be changed after the contract is deployed.

Go back to Findings Summary

I1: Contract id based validation

Impact:	Info	Likelihood:	N/A
Target:	`**/*`	Type:	Data validation

Description

The project uses zero-address checks for addresses data validation, however, validation can be more stringent if contract ids are used.

Recommendation

To each component that is passed to another add a constant variable that contains the contract id and use it for validation.

For example, the SquidMulticall contract will contain a variable named `CONTRACT_ID`:

```
bytes32 public constant CONTRACT_ID = keccak256("Squid Multicall");
```

and the constructor in SquidRouter will contain a check for the value of this variable:

```
require(
    ISquidMulticall(_multicall).CONTRACT_ID() == keccak256("Squid
Multicall"),
    "Invalid multicall address"
);
```

This will help to reduce the risk of passing incorrect values.

Go back to Findings Summary

I2: Missing NatSpec documentation

Impact:	Info	Likelihood:	N/A
Target:	`**/*`	Type:	Best practices

Description

The NatSpec documentation is missing. Functions should contain for example an explanation for function parameters and return values. Proper documentation is important for code reviews and further development.

Recommendation

Cover all contracts and functions with the NatSpec documentation.

Go back to Findings Summary

Appendix A: How to cite

Please cite this document as:

Ackee Blockchain, Squid: Permit2 & Express Service, 28.2.2023.

Appendix B: Glossary of terms

The following terms might be used throughout the document:

Superclass/Ancestor of C

A contract that C inherits/derives from.

Subclass/Child of C

A contract that inherits/derives from C.

Syntactic contract

A Solidity contract. May have an inheritance chain, and may be deployed.

Deployed contract

An EVM account with non-zero code. If its source was written in Solidity, it was created through at least one syntactic contract. If that contract had superclasses (parents), it would be composed of multiple syntactic contracts.

Init/initialization function

A non-constructor function that serves as an initializer. Often used in upgradeable contracts.

External entrypoint

A `public` or `external` function.

Public/Publicly-accessible function/entrypoint

An `external` or `public` function that can be successfully executed by any network account.

Mutating function

A non-`view` and non-`pure` function.

Appendix C: Woke tests

The following test shows example of reverting the USDT on bridge call:

```
def test_usdt():
    default_chain.tx_callback = lambda tx: print(tx.console_logs)

    # set accounts
    owner = default_chain.accounts[0]
    alice = default_chain.accounts[1]
    default_chain.default_tx_account = owner

    # deploy contracts
    p2 = Permit2.deploy()
    gw = MockGateway.deploy()
    mc = SquidMulticall.deploy()
    sr = SquidRouter.deploy(gw, Address(3), mc, p2)
    sr.unpause()

    # set usdt
    usdt = Account("0xdac17f958d2ee523a2206206994597c13d831ec7")
    usdt = IUSDT(usdt)
    gw.setTokenAddress("USDT", usdt.address)
    # some address with enough USDT what we are going to impersonate
    usdt.transfer(owner, 10**8,
from_=Address("0x9696f59E4d72E237BE84fFD425DCaD154Bf96976"))
    usdt.approve(sr.address, 10**8)

    # this transaction reverts
    tx = sr.bridgeCall("Ethereum", "USDT", 10**8, [], alice, False, bytes
(b'salt'), return_tx=True)
    print(tx.call_trace)
    print(tx.console_logs)
    print(tx.error)
```

ackee

Thank You

Ackee Blockchain a.s.


Prague, Czech Republic


hello@ackeeblockchain.com


https://discord.gg/z4KDUbuPxq