CERTIK

MiL.k

Security Assessment

CertiK Assessed on Mar 3rd, 2025

CertiK Assessed on Mar 3rd, 2025

MiL.k

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
DeFi	Ethereum (ETH)	Manual Review, Static Analysis
LANGUAGE	**TIMELINE**	**KEY COMPONENTS**
Solidity	Delivered on 03/03/2025	N/A

CODEBASE

base

update_2025_03_03

View All in Codebase Page

COMMITS

641aebef6a019aa62a379007d10aa5a2e588009d

e161e8019d2b4b1e201c0785a4e56d3a82f0456a

View All in Codebase Page

Highlighted Centralization Risks

Initial owner token share is 100%

Vulnerability Summary

9	9	0	0	0	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined

0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
0	Major		Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
4	Medium	4 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
4	Minor	4 Resolved	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
1	Informational	1 Resolved	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | MiL.k

CODEBASE | MiL.k

Repository

base

update_2025_03_03

Commit

641aebef6a019aa62a379007d10aa5a2e588009d

e161e8019d2b4b1e201c0785a4e56d3a82f0456a

AUDIT SCOPE | MiL.k

14 files audited ● 5 files with Resolved findings ● 9 files without findings

ID	Repo	File	SHA256 Checksum
● ACC	key-inside/milk-contracts	src/Account.sol	9c62f254ba8bbfe8335c1cae75d988a1246a5a97486c138b19a474019a31debb
● ERC	key-inside/milk-contracts	src/ERC4337NaiveUtils.sol	89f70e37f1d902766c87bb0e6f1dc15a906327b159aa01b6203cb0793331e1a0
● EPB	key-inside/milk-contracts	src/EntryPoint.sol	9129422758e3f0593a99cb492376b690d2580a6974c0ec3d797b42cf75799cf8
● MSM	key-inside/milk-contracts	src/MultiSigManager.sol	34ebf1a82c8cb157be7e76cdf26993dca6e9a4188dc44c875e3afadee2501d9c
● TLW	key-inside/milk-contracts	src/TimeLockWallet.sol	c0c75d757c3e31a6ca86bd61b18ce6e13ca4c8eb2c9508365a037245d39afdcc
○ AFB	key-inside/milk-contracts	src/AccountFactory.sol	fb420427a885b905f4d2c89c44406ee32bfac4e7c7de06809d6205b963d79b2e
○ IER	key-inside/milk-contracts	src/IERC4337Naive.sol	12855affa8d3ae1dae0e285714ce87fb01a243ca14cca84f2518b3dcc5639fcb
○ ACO	key-inside/milk-contracts	src/Account.sol	3ee90b4eb3f7023ae07fe667b0102e99f3ef334434cfca1176a8682002c8a6b3
○ AFU	key-inside/milk-contracts	src/AccountFactory.sol	fb420427a885b905f4d2c89c44406ee32bfac4e7c7de06809d6205b963d79b2e

ID	Repo	File	SHA256 Checksum
ERN	key-inside/milk-contracts	src/ERC4337NaiveUtils.sol	cbb1935f3cafef1c6978412362acf8d0e3fcc553b6c3a1d3a62a73cc2ab42cf1
EPU	key-inside/milk-contracts	src/EntryPoint.sol	a9f10b42bdfbcd3f80fa6fd2f3fe3136564b5133cbdbfee13195b68951c9fa6c
IEC	key-inside/milk-contracts	src/IERC4337Naive.sol	4897a8d5f59c6a194381912c6107f1287efc9bcf8ca0234df4a4f58ed23be57d
MUL	key-inside/milk-contracts	src/MultiSigManager.sol	dfec33cdb643e88fec34d08d144db8f6ff40fa26411370aceebb32d69a9b0811
TIM	key-inside/milk-contracts	src/TimeLockWallet.sol	a7feb430ddb683c9ad7852f727082d4c74fd6c356b740d6007fecf95d0f610d8

APPROACH & METHODS | MiL.k

This report has been prepared for MiL.k to discover issues and vulnerabilities in the source code of the MiL.k project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review and Static Analysis techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

FINDINGS | MiL.k

9	0	0	4	4	1
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for MiL.k. Through this audit, we have uncovered 9 issues ranging from different severity levels. Utilizing the techniques of Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
ACC-01	Contract / Function Can Not Receive Native Tokens As Intended	Volatile Code	Medium	Resolved
MSM-03	Incorrect Owner Linking When No Items In `_owners`	Logical Issue	Medium	Resolved
MSM-04	Bypassing Signature Validation Due To Uninitialized Threshold	Logical Issue	Medium	Resolved
TLW-01	Front-Running Vulnerability In `depositFrom()` Allowing Unauthorized Deposits	Logical Issue	Medium	Resolved
ACC-02	Contract Inheritance Issue: Missing Interface Inheritance	Coding Issue	Minor	Resolved
EPB-01	Missing Validation Of `userOp.sender` In `executeUserOp` Method	Logical Issue	Minor	Resolved
ERC-01	EIP712 Standard Not Followed	Volatile Code	Minor	Resolved
MSM-05	Potential Gas Exhaustion In Owner Removal Due To Inefficient Iteration	Denial of Service	Minor	Resolved
ERC-02	Assembly Usage	Coding Issue	Informational	Resolved

ACC-01 CONTRACT / FUNCTION CAN NOT RECEIVE NATIVE TOKENS AS INTENDED

Category	Severity	Location	Status
Volatile Code	Medium	src/Account.sol (base): 45, 51	Resolved

Description

The issue in the `executeUserOp` function arises because the contract does not have a mechanism to receive native tokens, yet the function attempts to execute a call with a value. Suppose the contract lacks a `receive` or `fallback` function and is not designed to accept native tokens. In that case, any transaction involving a nonzero value will fail, making the function unusable for operations requiring Ether transfers.

Recommendation

To mitigate the issue, we recommend:

1. Implementing a `receive` or `fallback` function in the contract. This will enable the contract to handle and accept incoming native token transactions properly.
2. Adding the `payable` modifier to any functions that are intended to receive native tokens and process transactions involving Ether.

Alleviation

[**MiL.k Token Team** 12 Feb 2025]: The `Account` contract is not payable. So, modified codes not to transfer native coins. Removed `value` part from `callData` of user operations.

[**CertiK Team** 12 Feb 2025]: The change was made in commit e327da32.

MSM-03 | INCORRECT OWNER LINKING WHEN NO ITEMS IN `_owners`

Category	Severity	Location	Status
Logical Issue	Medium	src/MultiSigManager.sol (base): 94~101	Resolved

Description

The issue in the `_addOwner` function arises when there are no existing items in the `_owners` mapping, and the `PIVOT` has not been added. In this case, the function incorrectly sets `_owners[owner] = 0` and `_owners[PIVOT] = owner`. However, for the list to be consistent, the owner should point to the `PIVOT`, not the other way around. This results in a broken owner linkage, where the new owner does not correctly reference the `PIVOT` in the list, leading to potential issues with owner management and contract logic.

Recommendation

We recommend updating the logic in the `_addOwner` function to ensure that when there are no existing items, the owner correctly points to the `PIVOT`, i.e., `_owners[owner] = PIVOT`; and `_owners[PIVOT] = owner`;. This will maintain the integrity of the owner list and avoid any linkage errors.

Alleviation

[**MiL.k Token Team** 12 Feb 2025]: Changed `_initOwners` function to the constructor. So `_owners` always has at least 1 item.

[**CertiK Team** 12 Feb 2025]: The change was made in commit e327da32.

MSM-04 BYPASSING SIGNATURE VALIDATION DUE TO UNINITIALIZED THRESHOLD

Category	Severity	Location	Status
Logical Issue	Medium	src/MultiSigManager.sol (base): 152~165	Resolved

Description

The issue in the `_validateSignature` function arises because the threshold `_threshold` is set to `0` by default, and it is only properly initialized when `_initOwners` is called. If `_initOwners` is not invoked, the threshold remains `0`, which means the `require(count >= _threshold)` condition will always fail. This allows anyone to bypass the signature validation process, as the threshold check is effectively ignored. This vulnerability can lead to unauthorized users being able to bypass critical security checks and potentially manipulate the system.

Recommendation

We recommend ensuring that `_threshold` is properly initialized before any calls to `_validateSignature`. This can be done by either adding a check in the function to enforce that `_threshold > 0` or ensuring that `_initOwners` is always called early in the contract's lifecycle to prevent this bypass.

Alleviation

[**MiL.k Token Team** 12 Feb 2025]: Due to the processing of MSM-03, `_threshold` is always greater than 0

[**CertiK Team** 12 Feb 2025]: The change was made in commit e327da32.

TLW-01 FRONT-RUNNING VULNERABILITY IN `depositFrom()` ALLOWING UNAUTHORIZED DEPOSITS

Category	Severity	Location	Status
Logical Issue	Medium	src/TimeLockWallet.sol (update_2025_02_21): 96, 98, 115~116	Resolved

Description

The `depositFrom()` can be called by anyone, and there is no check for `msg.sender`, allowing an attacker to front-run a legitimate deposit by calling depositFrom first with a minimal amount (e.g., 1 wei) and the same `timestamp`. This can disrupt the contract's logic by preventing the legitimate user's deposit, as the `timestamp` is now already occupied. In a more severe case, an attacker could deposit with a far-future `timestamp`, effectively locking the approved tokens for an extended period (e.g., 100 years), making them unrecoverable. This vulnerability undermines the contract's intended functionality and token management.

Recommendation

We recommend adding a check to ensure that only the from address or an approved spender can call `depositFrom()`, preventing unauthorized deposits and front-running attacks that disrupt the contract's logic.

Alleviation

[**MiL.k team** 03 Mar 2025]: Updated the source code that only approved spender can make deposit.

ACC-02 CONTRACT INHERITANCE ISSUE: MISSING INTERFACE INHERITANCE

Category	Severity	Location	Status
Coding Issue	Minor	src/Account.sol (base): 10	Resolved

Description

The issue in the Account contract is that while it inherits from `MultiSigManager`, it does not inherit from the `IAccount` interface, even though it implements the functionality expected from `IAccount`. According to best practices, contracts should inherit from relevant interfaces to provide clear and explicit contract structure, ensuring that the contract adheres to the defined interface methods. By not inheriting from `IAccount`, the `Account` contract may lead to confusion, lack of transparency, and potential compatibility issues with other contracts or external systems that rely on the `IAccount` interface.

Recommendation

We recommend modifying the Account contract to inherit from the `IAccount` interface in addition to `MultiSigManager`. This would clearly define the contract's expected interface, improve readability, and ensure better interoperability with other contracts or external applications expecting the `IAccount` interface.

Alleviation

[**MiL.k Token Team** 12 Feb 2025]: Modified the `Account` contract to inherit from `IAccount`

[**CertiK Team** 12 Feb 2025]: The change was made in commit e327da32.

EPB-01 MISSING VALIDATION OF `userOp.sender` IN `executeUserOp` METHOD

Category	Severity	Location	Status
Logical Issue	Minor	src/EntryPoint.sol (base): 61	Resolved

Description

In the `executeUserOp` method of the `Account` contract, the `userOp.sender` is expected to be the `Account` contract itself. However, there is no validation to ensure that `userOp.sender` is indeed the correct contract address, which could lead to unexpected behavior. To prevent such conditions, a check should be added to verify that `userOp.sender == address(this)` . This ensures that the method is only executed within the intended contract context, protecting the contract from unintended invocations.

Recommendation

We recommend adding the condition `require(userOp.sender == address(this), "Invalid sender");` in the `executeUserOp` method to enforce that the function can only be called from the `Account` contract itself.

Alleviation

[**MiL.k Token Team** 12 Feb 2025]: Added a line `require(userOp.sender == address(this), InvalidOpSender(userOp.sender));`

[**CertiK Team** 12 Feb 2025]: The change was made in commit e327da32.

ERC-01 | EIP712 STANDARD NOT FOLLOWED

Category	Severity	Location	Status
Volatile Code	Minor	src/ERC4337NaiveUtils.sol (base): 24	Resolved

Description

`Account._validateUserOp()` calculates the `ERC4337NaiveUtils.hash()` to summarize the operation executed and `_validateSignature()` of this hash. However, the message hashed does not follow EIP712 standard. In particular, it doesn't contain `typeHash`, like `bytes32 constant EXECUTE_USEROP_TYPEHASH = keccak256("executeUserOp(NaiveUserOperation userOp,bytes32 userOpHash)")`. The `typeHash` allows the signer to authorize the specific action, not the data, and thus minimizes the risk the signature will be used in another context.

Recommendation

We recommend following EIP712 standard.

Alleviation

[**MiL.k Token Team** 12 Feb 2025]:

- Modified the `Account` contract to inherit from `EIP712`.
- Added `typeHash` to a user operation hash.
- Removed codes that pass `userOpHash` argument from `EntryPoint` to `Account`.

[**CertiK Team** 12 Feb 2025]: The change was made in commit e327da32.

MSM-05 POTENTIAL GAS EXHAUSTION IN OWNER REMOVAL DUE TO INEFFICIENT ITERATION

Category	Severity	Location	Status
Denial of Service	Minor	src/MultiSigManager.sol (base): 94~101, 135	Resolved

Description

The issue in the Solidity code arises from the `_prevOwnerOf` function, which iterates through `_owners` to find the previous owner in a linked list structure. If the list of owners grows too long, this loop could consume excessive gas, making it impractical or even impossible to execute due to block gas limits. Consequently, any function that relies on `_prevOwnerOf`, such as `_removeOwner`, may fail if the owner list becomes too large. This can result in an inability to remove owners, leading to potential governance or security risks where inactive or malicious owners cannot be removed.

Recommendation

We recommend limiting the number of owners to a manageable size to prevent excessive gas consumption when iterating through the list. By enforcing an upper bound on the number of owners, the contract can ensure that functions relying on `_prevOwnerOf` remain executable within the gas limits, preventing situations where owner removal becomes impossible.

Alleviation

[**MiL.k Token Team** 12 Feb 2025]: The upper bound `MAX_OWNER_COUNT` was set.

[**CertiK Team** 12 Feb 2025]: The change was made in commit e327da32.

ERC-02 | ASSEMBLY USAGE

Category	Severity	Location	Status
Coding Issue	Informational	src/ERC4337NaiveUtils.sol (base): 29~34	Resolved

Description

The smart contract contains an assembly code block, which enables direct interaction with the Ethereum Virtual Machine (EVM) and permits developers to write low-level code that manipulates the EVM. This low-level access is error-prone and can introduce complexity, vulnerabilities, and logic errors.

```
30          assembly ("memory-safe") {
31              result.offset := 0
32              result.length := 0
33          }
```

Recommendation

It is recommended to refactor the code to use higher-level Solidity constructs instead of EVM assembly, improving readability, maintainability, and reducing the risk of potential issues.

Alleviation

[**MiL.k Token Team** 12 Feb 2025]: Removed low-level codes

[**CertiK Team** 12 Feb 2025]: The change was made in commit e327da32.

OPTIMIZATIONS | MiL.k

ID	Title	Category	Severity	Status
MSM-01	Redundant Threshold Validation In Require Statement	Gas Optimization	Optimization	Resolved
MSM-02	Redundant Condition In Signature Verification Loop	Gas Optimization	Optimization	Resolved

MSM-01 REDUNDANT THRESHOLD VALIDATION IN REQUIRE STATEMENT

Category	Severity	Location	Status
Gas Optimization	Optimization	src/MultiSigManager.sol (base): 74	Resolved

Description

The issue in the Solidity code is that the require statement checking `threshold_ > 0 && threshold_ <= owners_.length` is redundant because the same validation is performed again inside the `_changeThreshold(threshold_)` function. This results in unnecessary gas consumption since every require call incurs a cost. Instead of checking the condition twice, the initial check should be removed, relying solely on `_changeThreshold(threshold_)` to enforce the constraint. This would optimize execution by reducing redundant operations while maintaining the same security guarantees.

Recommendation

We recommend removing the redundant require statement before calling `_changeThreshold(threshold_)`. Instead, the validation should be handled solely within `_changeThreshold()`, ensuring that the contract enforces the correct threshold without incurring unnecessary gas costs.

Alleviation

[**MiL.k Token Team** 12 Feb 2025]: Removed redundant checks.

[**CertiK Team** 12 Feb 2025]: The change was made in commit e327da32.

MSM-02 REDUNDANT CONDITION IN SIGNATURE VERIFICATION LOOP

Category	Severity	Location	Status
Gas Optimization	Optimization	src/MultiSigManager.sol (base): 158	Resolved

Description

The issue in the Solidity code is the redundant loop condition `i < signature.length && count < _threshold`. Since each iteration of the loop increments `i` by `65` (the length of a single signature) and the number of iterations is inherently limited by `_threshold` (due to `count < _threshold`), the `i < signature.length` check is unnecessary. Removing this redundant condition simplifies the code and slightly optimizes gas usage.

Recommendation

We recommend modifying the loop condition to only check `count < _threshold`, as it inherently ensures that the loop does not exceed the required number of valid signatures. This change eliminates unnecessary operations, improving efficiency while maintaining correctness.

Alleviation

[**MiL.k Token Team** 12 Feb 2025]: Removed unnecessary checks.

[**CertiK Team** 12 Feb 2025]: The change was made in commit e327da32.

APPENDIX | MiL.k

Finding Categories

Categories	Description
Gas Optimization	Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
Denial of Service	Denial of Service findings indicate that an attacker may prevent the program from operating correctly or responding to legitimate requests.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

Elevating Your Entire Web3 Journey

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.