Portkey - zklogin implementation

Security Assessment

CertiK Assessed on Dec 5th, 2024

CertiK Assessed on Dec 5th, 2024

Portkey - zklogin implementation

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

Executive Summary

TYPES	ECOSYSTEM	METHODS
Zero Knowledge	PortKey	Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 12/05/2024	N/A

CODEBASE

https://github.com/Portkey-Wallet/zkLogin-circuit/

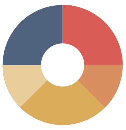
View All in Codebase Page

COMMITS

- ee1a9ee620dae6e1d68d95f7d0d626fd5930cfdb
- a86ee05a46a5dc0b706a45487c1e9485e65af218
- a90c6efbdd19a8ed2263cc9cea9a1941aa126683

View All in Codebase Page

Vulnerability Summary



8	7	1	0	0	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined

Count	Severity	Status	Description
2	Critical	2 Resolved	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.
1	Major	1 Mitigated	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.
2	Medium	2 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
1	Minor	1 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.
2	Informational	2 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 | PORTKEY - ZKLOGIN IMPLEMENTATION

CODEBASE | PORTKEY - ZKLOGIN IMPLEMENTATION

Repository

https://github.com/Portkey-Wallet/zkLogin-circuit/

Commit

- ee1a9ee620dae6e1d68d95f7d0d626fd5930cfdb
- a86ee05a46a5dc0b706a45487c1e9485e65af218
- a90c6efbdd19a8ed2263cc9cea9a1941aa126683

AUDIT SCOPE | PORTKEY - ZKLOGIN IMPLEMENTATION

21 files audited ● 5 files with Resolved findings ● 16 files without findings

ID	Repo	File	SHA256 Checksum
● SH2	Portkey-Wallet/zkLogin-circuit	circuits/helpers/sha256.circom	2378ec440f0fa7e3befa7c8797623969eebe470c3a79dd5f77e875e9eafa77a7
● UTI	Portkey-Wallet/zkLogin-circuit	circuits/helpers/utils.circom	7b299eba3fb9f76895f71e3339b876d85f841d2084e38858e257912bccf1eff5
● HML	Portkey-Wallet/zkLogin-circuit	circuits/idHashMapping.circom	0d182d28ea7218ed63b6c619baeec36d4e4bb2822f670b6cadc15b829566fdb2
● LLP	Portkey-Wallet/zkLogin-circuit	circuits/zkLogin.circom	bfc56b4f4ef403635850b1afd56f77517ef9d3ffdbe0feba163d06a8b4755b09
● LSL	Portkey-Wallet/zkLogin-circuit	circuits/zkLoginSha256.circom	9913e47dc3aa03fa197213e0711be62f5d0d9d01b7934c13eb00fa895838904f
● LPW	Portkey-Wallet/zkLogin-circuit	circuits/helpers/base64.circom	edf5e0dff4412a22e6ea1da02bf4582f9a9f51810d61dc7da0b7722342536ff6
● BIG	Portkey-Wallet/zkLogin-circuit	circuits/helpers/bigint.circom	405fff88f22f5a5b4de40cc45146ff9d4ef5b29ef8d7cb6b654b9689fb26e792
● BII	Portkey-Wallet/zkLogin-circuit	circuits/helpers/bigint_func.circom	a39caa50d6a2a870675c6b5192ce8faa6680867c6175b49493f4cb26697031d4
● FPH	Portkey-Wallet/zkLogin-circuit	circuits/helpers/fp.circom	41e131159c21eb2fba458e1fb26e7406967f4b1a7583e9a283028b7326612307

ID	Repo	File	SHA256 Checksum
HAS	Portkey-Wallet/zkLogin-circuit	circuits/helpers/hashtofield.circom	416a2731518388b9d613da447cc7e1b24fb5990cfc712c0bca21ddff87d38fa2
IDH	Portkey-Wallet/zkLogin-circuit	circuits/helpers/idhash_poseidon.circom	96d294e1eb11458d7e42a83a7e7a32584b50ea21fd5874214b51c76a8d891de5
IDA	Portkey-Wallet/zkLogin-circuit	circuits/helpers/idhash_sha256.circom	df46bca4a59a6a6c159225d4cc964d04040b2d5540635bb99706d9fd6b116b35
JWT	Portkey-Wallet/zkLogin-circuit	circuits/helpers/jwt-new.circom	60dca887bb24ab556c6d8835edc153cdadc41f80e7a3d362a8ca5c6666e8a046
JWC	Portkey-Wallet/zkLogin-circuit	circuits/helpers/jwtchecks.circom	2b582c6b76df074057f81360969c19ce8900317a9d2dc7f41187ba65ae93d74a
MIS	Portkey-Wallet/zkLogin-circuit	circuits/helpers/misc.circom	608c717bba18db6fa6d55897a1f41a2fdf40633c05e14b3f7e192911140cdf8b
RSA	Portkey-Wallet/zkLogin-circuit	circuits/helpers/rsa-new.circom	089c2265eda8cda097a4f3f8d065fc8c4787d1da9209a70275e9bc5f273f4159
SHA	Portkey-Wallet/zkLogin-circuit	circuits/helpers/sha256-new.circom	f005c7aee064ee86cc89a142bef9e08ed6a4175fa6c6406825d328f239842ece
SH5	Portkey-Wallet/zkLogin-circuit	circuits/helpers/sha256general.circom	d13f9808457faac966526117ae12e472f47f64bfe93fb1c19eba77fd72e198ae
SH6	Portkey-Wallet/zkLogin-circuit	circuits/helpers/sha256partial.circom	71ae376ef787585b19f8f82fbefce8b4b9e3d44bca5aaa63eaf09b370958117a
STR	Portkey-Wallet/zkLogin-circuit	circuits/helpers/string.circom	ebb6cc22bcac4509b9014619d3db8459c1d3e176cdadf32c56de1d520a589003

ID	Repo	File	SHA256 Checksum
STI	Portkey-Wallet/zkLogin-circuit	circuits/helpers/strings.circom	55b4b5005310d95e83b0832aeadb24a761767cca92ca0ec9e1c8427634268a06

APPROACH & METHODS | PORTKEY - ZKLOGIN IMPLEMENTATION

This report has been prepared for Portkey to discover issues and vulnerabilities in the source code of the Portkey - zklogin implementation 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.

REVIEW NOTES | PORTKEY - ZKLOGIN IMPLEMENTATION

Overview

The **zkLogin** project for Portkey concerns the zero-knowledge (ZK) circuits of a login system. The main component of this project is the circuit for logins. Users are meant to provide a JSON Web Token (JWT) as a private input, the public key that signed the JWT as a public input, and a possible `salt` value as a private input. The signature check is conducted within the circuit.

The circuit then extracts the `sub` and `nonce` claims of the JWT. The `nonce` and a hash of the `sub` with the possible `salt` are produced as public outputs. This hash is meant to be an ID.

There are three main circuits:

1. `ZkLogin` : The main login circuit described above where the hash function used is the Poseidon hash function.
2. `ZkLoginSha256` : The same as the `ZkLogin` circuit, but the hash function is instead the SHA256 hash function.
3. `IdHashMapping` : This circuit takes in a `sub` and `salt` as private inputs to produce the corresponding Poseidon hash and SHA256 hash that would have been produced if they were used in the above login circuits.

This audit only concerns the circuits themselves and not how the proofs generated from the circuits will be used.

External Dependencies

In **zkLogin**, the project relies on the circomlib library for many utility circuits.

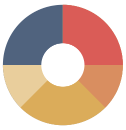
Although some checks were made, such as known issues with circuits like the `LessThan` circuit, the scope of the audit treats the library as a black box and assumes their functional correctness.

The public key meant to be used for the login circuits is also assumed to be well-formed and trusted by the project.

Centralization

In the **zkLogin** project, the project requires a common reference string (CRS) to be able to generate and validate zero-knowledge proofs. Compromises to the generation of the CRS can allow one to generate valid proofs for false statements. Details of this are stated in the finding `Use of Trusted Setup` .

FINDINGS | PORTKEY - ZKLOGIN IMPLEMENTATION



8	2	1	2	1	2
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Portkey - zklogin implementation. Through this audit, we have uncovered 8 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
SH2-01	Underconstrained Circuit `Sha256BytesOutputBytes` Allows Arbitrary SHA256 Hash	Logical Issue	Critical	Resolved
UTI-01	Underconstrained Circuit `BitsToBytes` Allows Arbitrary SHA256 Hash	Logical Issue	Critical	Resolved
GLOBAL-02	**Use Of Trusted Setup**	**Centralization**	**Major**	**Mitigated**
SH2-02	Lack Of Check On Padded Message	Logical Issue	Medium	Resolved
SH2-03	Underconstrained Circuit `Sha256PadAndHash`	Logical Issue	Medium	Resolved
GLOBAL-01	Unused Circuits	Code Optimization	Minor	Resolved
LPW-02	Inconsistency In Claim Lengths	Inconsistency	Informational	Resolved
LPW-03	Unimplemented JWT Claims	Design Issue	Informational	Resolved

SH2-01 | UNDERCONSTRAINED CIRCUIT `Sha256BytesOutputBytes` ALLOWS ARBITRARY SHA256 HASH

Category	Severity	Location	Status
Logical Issue	● Critical	circuits/helpers/sha256.circom: 48	● Resolved

Description

The output of the circuit `Sha256BytesOutputBytes` is not constrained.

```
48      out <-- B2B.out;
```

This circuit is used within the circuits `ZkLoginSha256` and `IdHashMapping` and no additional checks are placed on the output, allowing the verifier to verifiy a malicious witness with an arbitrary output. This is dangerous as this circuit is meant to produce a specific SHA256 hash.

Since `B2B.out` is a quadratic expression, it is best to use `<==` to assign and constrain the output.

Proof of Concept

The issue is the same as in finding **Underconstrained Circuit `BitsToBytes` Allows Arbitrary SHA256 Hash**. We refer to the proof of concept given there.

Recommendation

It is recommended to use `<==` to constrain the output.

Alleviation

[Portkey Team, Sep. 7, 2024]: The team heeded the advice and resolved the issue in commit c2ff65f77a750feac0125a9e7b926d00c54b5d89 by constraining the output.

UTI-01 | UNDERCONSTRAINED CIRCUIT `BitsToBytes` ALLOWS ARBITRARY SHA256 HASH

Category	Severity	Location	Status
Logical Issue	Critical	circuits/helpers/utils.circom: 232	Resolved

Description

The circuit `BitsToBytes` is meant to turn an array of bits into an array of bytes. However, there are no constraints involved, allowing the output to have elements greater than 8 bits, or even elements that have no connection with the original input array.

```
template BitsToBytes(bits){
  signal input in[bits];
  signal output out[bits/8];
  for (var i=0; i<bits/8; i++) {
    var bytevalue = 0;
    for (var j=0; j<8; j++) {
      bytevalue |= in[i * 8 + j] ? (1 << (7-j)) : 0;
    }
    out[i] <-- bytevalue;
  }
```

This means that regardless of the input array, the output array can hold any values. This circuit is used in the main circuits `ZkLoginSha256` and `IdHashMapping`, which do not have further checks on the output.

This is dangerous as the output is meant to be a specific SHA256 hash, but an attacker would be able to choose an arbitrary hash to use.

Proof of Concept

Using zkREPL, a proof is created for the following circuit:

```
template BitsToBytes(bits){
  signal input in[bits];
  signal output out[bits/8];
  for (var i=0; i<bits/8; i++) {
    var bytevalue = 0;
    for (var j=0; j<8; j++) {
      bytevalue |= in[i * 8 + j] ? (1 << (7-j)) : 0;
    }
    out[i] <-- 0;
  }
}

component main = BitsToBytes(8);

/* INPUT = {
    "in": ["1", "1", "1", "1", "1", "1", "1", "1"]
} */
```

Note that the output for this circuit is just 0, whereas the original circuit would have produced 255. We then give this proof with output 0 (the .zkey file) to the verifier of the original circuit that has `out[i] <-- bytevalue`, and this proof was accepted.

This shows that an attacker can set `out` to be any arbitrary value as there are no constraints on `out`.

Recommendation

It is recommended to constrain the output properly. A possible solution is to use the `Bits2Num(8)` circuit in circomlib to turn each 8 bits of `in` to a byte.

Alleviation

[Portkey Team, Sep. 7, 2024]: The team heeded the advice and resolved the issue in commit b8587568047b046d3c449e4c93c6cbab9be939e1 by constraining the output.

GLOBAL-02 | USE OF TRUSTED SETUP

Category	Severity	Location	Status
Centralization	**Major**		**Mitigated**

Description

A trusted setup is used to construct a common reference string (CRS), known as toxic waste, which is required to obtain proving and verification parameters. If this toxic waste is ever revealed, then fake proofs can be created for false statements.

As this project concerns a login system, these fake proofs can allow one to log in to any account.

Recommendation

The risk describes the current project design and potentially makes iterations to improve the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We recommend carefully managing all accounts involved in the generation of the CRS.

It is recommended to be as transparent as possible about the parties involved in generating the CRS, such as if and which multi-party computation (MPC) was used as well as the current number of participants.

Alleviation

[Portkey Team, Sep. 7, 2024]: We are organising one round of trusted setup and will do one more if there are further changes done during auditing. The one included in the CI job of the repo is for testing purposes only.

[Portkey Team, Dec. 5, 2024]: We have performed a public trusted setup with 30 participants. See https://medium.com/portkey-aa-wallet-did/portkey-zklogin-groth16-trusted-setup-contributor-guide-e1fe0e6729fd and https://medium.com/portkey-aa-wallet-did/portkey-zklogin-groth16-trusted-setup-ceremony-call-for-participants-7be098de88f4 . And the contributions were performed using public npm tool @portkey/ceremony. All the intermediate artifacts are publicly available via s3://portkey-zklogin-ph2-ceremony/circuits/zklogin/ .

SH2-02 | LACK OF CHECK ON PADDED MESSAGE

Category	Severity	Location	Status
Logical Issue	Medium	circuits/helpers/sha256.circom: 92	Resolved

Description

The `Sha256PadBytes` circuit is meant to take a message and produce a padded message for the SHA256 hasher. This circuit is used as part of the `IdHashSha256` circuit, but there are no checks on the output of `Sha256PadBytes` , primarily the length of the padded message and the padded message itself.

```
85      padded_len <-- (in_bytes + 9) + padding_len;
86      assert(padded_len % 64 == 0);
87
88      component len2bytes = Packed2BytesBigEndian(8);
89      len2bytes.in <== in_bytes * 8;
90
91      for (var i = 0; i < max_bytes; i++) {
92          padded_text[i] <-- i < in_bytes ? in[i] : (i == in_bytes ? (1 << 7) : ((i < padded_len && i >= padded_len - 8) ? len2bytes.out[(i % 64 - 56)]: 0)); // Add the 1 on the end and text length
93      }
```

This allows the possibility of a witness with an invalid padding to be accepted.

It should be noted that there is a circuit `SHA2PadVerifier` in `sha256-new.circom` that checks for the correctness of padding that can be used.

Recommendation

It is recommended to constrain all signals so that only valid witness values are accepted. A possible solution is to

1. Use the `SHA2PadVerifier` circuit in `sha256-new.circom` to check that the padding is correct
2. Constrain that the first `in_bytes` elements of `padded_text` match `in` to ensure the message portion is correct

Alleviation

[Portkey Team, Sep. 10, 2024]: The team heeded the advice and resolved the issue in commits 4b26a8eeed38339df74ab7dc134934380a911c77 and 2d95d9483a01f9038a815ec167d49270c5f7b33a by checking that the padding is correct and that the message portion of the padded message is correct.

SH2-03 | UNDERCONSTRAINED CIRCUIT `Sha256PadAndHash`

Category	Severity	Location	Status
Logical Issue	Medium	circuits/helpers/sha256.circom: 29	Resolved

Description

The circuit `Sha256PadAndHash` is meant to pad and hash a message under the SHA256 spec. It first pads a message by using the `Sha256PadBytes` circuit.

```
28    component sha256Pad = Sha256PadBytes(max_padded_len);
29    sha256Pad.in <-- paddedBytes;
30    sha256Pad.in_bytes <== in_len;
```

However, the input message for the `Sha256PadBytes` message is only assigned with the value `paddedBytes` and is not constrained to be equal to this value `paddedBytes`. This allows a malicious user to generate a witness where `paddedBytes` and `sha256Pad.in` to be different values, meaning a message different from `paddedBytes` will be padded and hashed.

Recommendation

It is recommended to constrain `sha256Pad.in`. One solution is to use the `strings.circom::SliceFromStart` circuit to create an array whose start is `in[0:in_len]` and the rest of the elements are 0.

Alleviation

[Portkey Team, Sep. 7, 2024]: The team heeded the advice and resolved the issue in commit 1e1c6c21b2f16fc1656773f4611467caefdce81e by constraining the input.

GLOBAL-01 | UNUSED CIRCUITS

Category	Severity	Location	Status
Code Optimization	Minor		Resolved

Description

There are several unused circuits that can be removed from the codebase. Some of these circuits, such as those in `string.circom` or the circuit `Base64Lookup` in `base64.circom`, may pose risks as several signals are not constrained. This allows the possibility of improper use of these circuits that can cause vulnerabilities.

Since these circuits are not used in any main circuits, it would be best to remove them or move them into a test folder, if needed for testing.

Recommendation

It is recommended to remove unused code.

Alleviation

[Portkey Team, Sep. 7, 2024]: The team heeded the advice and resolved the issue in commit a86ee05a46a5dc0b706a45487c1e9485e65af218 by removing unused circuits.

LPW-02 | INCONSISTENCY IN CLAIM LENGTHS

Category	Severity	Location	Status
Inconsistency	● Informational	circuits/zkLogin.circom: 13; circuits/zkLoginSha256.circom: 13	● Resolved

Description

The length for the `sub` claim does not include the length for a colon and comma (or brace), while the length for a `nonce` claim does. The auditing team would like to check if this is intended or not.

Recommendation

If this is not intentional, then it is recommended to have similar formatting for claim lengths.

Alleviation

[Portkey Team, Sep. 10, 2024]: It's intentional that `maxSubClaimLen` doesn't include 2 additional places while we allow more room for nonce claim. Our reasoning is we allow 255 places for sub whose full capacity is unliked to be occupied hence we don't need to be so strict on the formatting. However for nonce, the full capacity is always used. Hence we want to leave more room for the nonce claim to cater to formatting scenarios.

LPW-03 | UNIMPLEMENTED JWT CLAIMS

Category	Severity	Location	Status
Design Issue	● Informational	circuits/zkLogin.circom: 26; circuits/zkLoginSha256.circom: 26	● Resolved

Description

Currently, only two claims are extracted from the JWT: sub and nonce. Other important claims (that may or may not exist in the JWT) that are not checked include:

1. iat: time the JWT was issued
2. exp: time the JWT expires
3. iss: issuer of the JWT
4. aud: party meant to process the JWT

The auditing team would like to check that this design is intentional as some checks may be helpful for security. For example, checking `iat` or `exp` can help prevent replay attacks.

Recommendation

Depending on how the proofs will be used, it may be necessary to implement additional checks.

Alleviation

[Portkey Team, Sep. 10, 2024]: For our case, all other claims are not needed.

1. iat: It's not needed as our nonce is derived from a payload which already contains a timestamp
2. exp: We won't make use of this expiry time, as the consumer side sets a more stricter expiry time
3. iss: We don't need it as we already know the issuer whose JWK is configured
4. aud: It's not needed as we are an open protocol whereby everyone is allowed to interact with the OIDC provider to get the token and the security is guaranteed by the protocol via the use of nonce

OPTIMIZATIONS | PORTKEY - ZKLOGIN IMPLEMENTATION

ID	Title	Category	Severity	Status
HML-01	Unused Signal	Code Optimization	Optimization	Resolved
LPW-01	Unused Variables	Code Optimization	Optimization	Resolved

HML-01 | UNUSED SIGNAL

Category	Severity	Location	Status
Code Optimization	Optimization	circuits/idHashMapping.circom: 10	Resolved

Description

The signal `saltLen` in the circuit `IdHashMapping` is unused. Since 16 is always used as the salt length, this signal is not needed.

Recommendation

It is recommended to remove unnecessary signals.

Alleviation

[Portkey Team, Sep. 10, 2024]: The team heeded the advice and resolved the issue in commit 695166edeb9ba613fc6ce28f3f7ec3ff066456aa by removing the unused signal.

LPW-01 | UNUSED VARIABLES

Category	Severity	Location	Status
Code Optimization	Optimization	circuits/zkLogin.circom: 15~18; circuits/zkLoginSha256.circom: 15~18	Resolved

Description

Lengths for an `Exp` claim are unused.

Recommendation

It is recommended to remove unnecessary variables for code clarity.

Alleviation

[Portkey Team, Sep. 10, 2024]: The team heeded the advice and resolved the issue in commit 0a544a7b1fc3cfa8266e463e7706a150439b0644 by removing the unused variables.

APPENDIX | PORTKEY - ZKLOGIN IMPLEMENTATION

Finding Categories

Categories	Description
Inconsistency	Inconsistency findings refer to different parts of code that are not consistent or code that does not behave according to its specification.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.
Design Issue	Design Issue findings indicate general issues at the design level beyond program logic that are not covered by other finding categories.

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.

 Copyright © CertiK