Competitive Security Assessment

UniPassVerifierContract

Jun 26th, 2023

Secure3

secure3.io

Summary 4

Overview 5

Audit Scope 6

Code Assessment Findings 7

UPV-1: Lack of Point Validation in `point_mul` Function 10

UPV-2:Inconsistency in Formula Representation for z_substring Calculation 12

UPV-3:Information Leakage in `batch_evaluate_lagrange_poly_out_of_domain` Function with a Potential Attack Scenario 17

UPV-4:trust model is obscured in `UnipassVerifier` contract `verifyProof` function 19

UPV-5:Collision computation in `UnipassVerifier` contract `setupVKHash` function 20

UPV-6:Gas Optimization for Large Data Array. 24

UPV-7:Integer Overflow and Underflow risk in `UnipassVerifier.sol` contract `sha256PaddingLen` function 26

UPV-8:Integer Overflow and Underflow risk in `UnipassVerifier` contract `verifyProof` function 29

UPV-9:Logical error in `UnipassVerifier.sol` contract `setBit` function 36

UPV-10:abi.encodePacked() should not be used with dynamic types when passing the result to a hash function such as keccack256() 39

UPV-11:Cache the for loop count variable in memory 44

UPV-12:Code Style in `UnipassVerifier` contract `deserialize_proof` function 48

UPV-13:Don't initialize variables with default value 53

UPV-14:Insufficient Input Validation 61

UPV-15:Lack Security Proof 62

UPV-16:Logical risk in `UnipassVerifier.sol` contract `verifyV2048` function 63

UPV-17:Missing Error Messages and Event Emissions in `new_g1_checked` Function 66

UPV-18:Redundant variables 68

UPV-19: `++i` costs less gas than `i++`, especially when it's used in `for`-loops (`--i` / `i--` too) 69

UPV-20: `i < dens_len` condition is useless in `PlookupSingleCore` contract `batch_evaluate_lagrange_poly_out_of_domain` function 74

UPV-21:poly_nums values not used in `PlookupSingleCore` contract `batch_evaluate_lagrange_poly_out_of_domain` function 75

UPV-22:replace subsequent `PairingsBn254.new_fr(1)` with `PairingsBn254.copy(one)` in `PlookupSingleCore` contract `batch_evaluate_lagrange_poly_out_of_domain` function 76

UPV-23:tmp3 doesn't need to be initialized in `PlookupSingleCore` contract `batch_evaluate_lagrange_poly_out_of_domain` function 77

Disclaimer 78

Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

- Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
- Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
- Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.
- Verify the code base is compliant with the most up-to-date industry standards and security best practices.
- Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

Overview

Project Detail

Project Name	UniPassVerifierContract
Platform & Language	Solidity
Codebase	• https://github.com/UniPassID/UniPass-verifier-contract • audit commit - cf6e856df8aada682eccab1459e6028fd7c2bb09 • final commit - 6284228012f90c2d51a0cdff3f20f0186723c79d
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

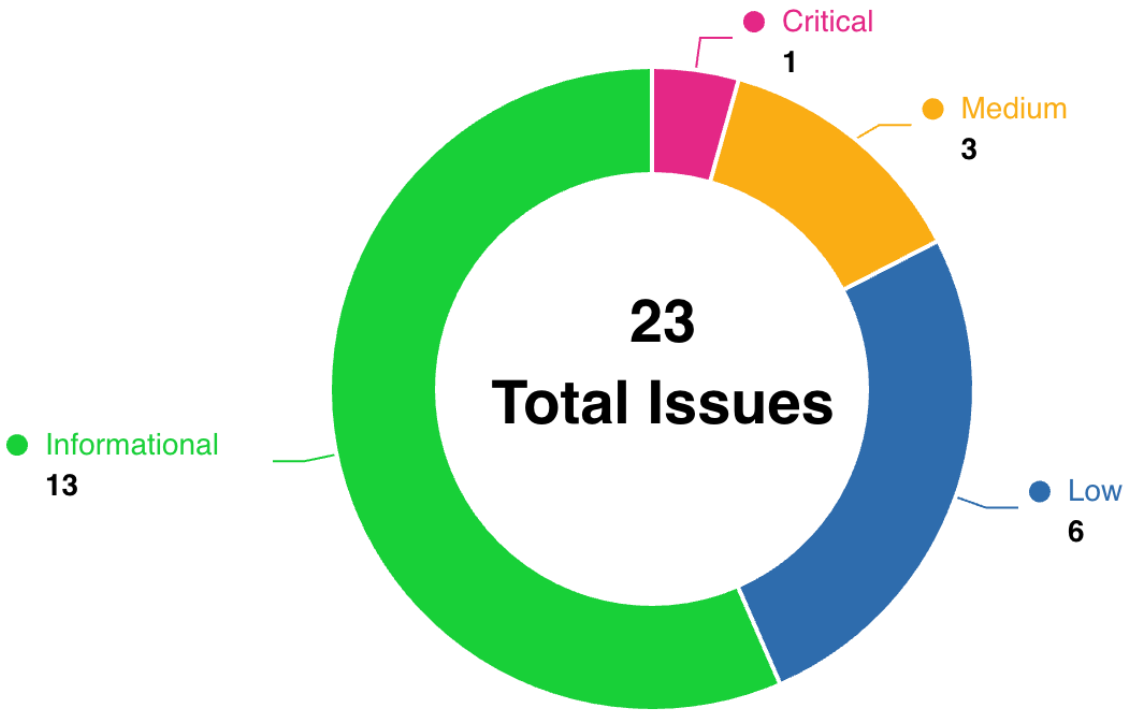
Code Vulnerability Review Summary

Vulnerability Level	Total	Reported	Acknowledged	Fixed	Mitigated	Declined
Critical	1	0	0	0	0	1
Medium	3	0	0	0	0	3
Low	6	0	0	0	0	6
Informational	13	0	0	7	1	5

Audit Scope

File	Commit Hash
contracts/PlookupSingleCore.sol	cf6e856df8aada682eccab1459e6028fd7c2bb09
contracts/UnipassVerifier.sol	cf6e856df8aada682eccab1459e6028fd7c2bb09
contracts/PlonkCoreLib.sol	cf6e856df8aada682eccab1459e6028fd7c2bb09

Code Assessment Findings



ID	Name	Category	Severity	Status	Contributor
UPV-1	Lack of Point Validation in `point_mul` Function	Logical	Critical	Declined	BradMoonUESTC
UPV-2	Inconsistency in Formula Representation for z_substring Calculation	Logical	Medium	Declined	BradMoonUESTC
UPV-3	Information Leakage in `batch_evaluate_lagrange_poly_out_of_domain` Function with a Potential Attack Scenario	Logical	Medium	Declined	BradMoonUESTC

UPV-4	trust model is obscured in `UnipassVerifier` contract `verifyProof` function	Logical	Medium	Declined	alansh
UPV-5	Collision computation in `UnipassVerifier` contract `setupVKHash` function	Increase Security	Low	Declined	newway55
UPV-6	Gas Optimization for Large Data Array.	Gas Optimization	Low	Declined	lfzkoala
UPV-7	Integer Overflow and Underflow risk in `UnipassVerifier.sol` contract `sha256PaddingLen` function	Integer Overflow and Underflow	Low	Declined	newway55
UPV-8	Integer Overflow and Underflow risk in `UnipassVerifier` contract `verifyProof` function	Integer Overflow and Underflow	Low	Declined	newway55
UPV-9	Logical error in `UnipassVerifier.sol` contract `setBit` function	Logical	Low	Declined	newway55
UPV-10	abi.encodePacked() should not be used with dynamic types when passing the result to a hash function such as keccack256()	Language Specific	Low	Declined	thereksfour
UPV-11	Cache the for loop count variable in memory	Gas Optimization	Informational	Declined	thereksfour
UPV-12	Code Style in `UnipassVerifier` contract `deserialize_proof` function	Code Style	Informational	Declined	newway55
UPV-13	Don't initialize variables with default value	Gas Optimization	Informational	Declined	thereksfour
UPV-14	Insufficient Input Validation	Code Style	Informational	Fixed	BradMoonUESTC
UPV-15	Lack Security Proof	Code Style	Informational	Mitigated	lfzkoala
UPV-16	Logical risk in `UnipassVerifier.sol` contract `verifyV2048` function	Logical risk	Informational	Declined	newway55
UPV-17	Missing Error Messages and Event Emissions in `new_g1_checked` Function	Code Style	Informational	Fixed	BradMoonUESTC

UPV-18	Redundant variables	Code Style	Informational	Declined	lfzkoala
UPV-19	`++i` costs less gas than `i++`, especially when it's used in `for`-loops (`--i`/`i--` too)	Gas Optimization	Informational	Fixed	thereksfour
UPV-20	`i < dens_len` condition is useless in `PlookupSingleCore` contract `batch_evaluate_lagrange_poly_out_of_domain` function	Gas Optimization	Informational	Fixed	alansh
UPV-21	poly_nums values not used in `PlookupSingleCore` contract `batch_evaluate_lagrange_poly_out_of_domain` function	Gas Optimization	Informational	Fixed	alansh
UPV-22	replace subsequent `PairingsBn254.new_fr(1)` with `PairingsBn254.copy(one)` in `PlookupSingleCore` contract `batch_evaluate_lagrange_poly_out_of_domain` function	Gas Optimization	Informational	Fixed	alansh
UPV-23	tmp3 doesn't need to be initialized in `PlookupSingleCore` contract `batch_evaluate_lagrange_poly_out_of_domain` function	Gas Optimization	Informational	Fixed	alansh

UPV-1: Lack of Point Validation in `point_mul` Function

Category	Severity	Status	Contributor
Logical	Critical	Declined	BradMoonUESTC

Code Reference

- code/UniPass-verifier-contract/contracts/PlonkCoreLib.sol#L263-L270

```
263:    /// Scalar multiplication for G1 point, return `s * p`
264:    function point_mul(G1Point memory p, Fr memory s)
265:        internal
266:        view
267:        returns (G1Point memory r)
268:    {
269:        point_mul_into_dest(p, s, r);
270:        return r;
```

Description

BradMoonUESTC : The `point_mul` function in the first contract does not perform a validity check on the input point, unlike the second contract. This can potentially lead to undefined behavior when the provided point is not on the elliptic curve, which might result in incorrect calculations or introduce potential security risks. Elliptic curve points are encoded as a Jacobian pair (X, Y) where the point at infinity is encoded as (0, 0)

For example, consider a scenario where an attacker provides an invalid point to the `point_mul` function. Without proper validation, the function may perform calculations on this invalid point, which can lead to incorrect results. These incorrect results could then be used in other parts of the contract or returned to the user, leading to unintended consequences and potential vulnerabilities.

Other reference：https://eips.ethereum.org/EIPS/eip-197

Recommendation

BradMoonUESTC : It is recommended to add a validity check for the input point in the `point_mul` function of the first contract to ensure that the point is a valid elliptic curve point. This can be achieved by replicating the check from the second contract, which sets the Y-coordinate to 0 if X is 0 and Y is 1:

```
if (p.X == 0 && p.Y == 1) {
    p.Y = 0;
}
```

Client Response

Declined, I don't quite understand the issues described here. First of all, `point_mul()` is an internal function and cannot be invoked by attackers. Secondly, the points are represented in affine form and `(0,0)` is used to denote infinity. All points submitted by attackers will be validated by `new_g1_checked()`, which checks if a point is on the curve. Also, one question: what does "Jacobian Pair" mean, as mentioned in the description? Regarding the recommendation part, can you clarify the purpose of setting `p.Y` to 0 if `p.X==0 && p.Y==1`?

UPV-2:Inconsistency in Formula Representation for z_substring Calculation

Category	Severity	Status	Contributor
Logical	Medium	Declined	BradMoonUESTC

Code Reference

- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L263-L375

```
263:     function verify_at_z(
264:         PairingsBn254.Fr memory zeta_n,
265:         PartialVerifierState memory state,
266:         Proof memory proof
267:     ) internal pure returns (bool) {
268:         /// lhs = t(z) * v(z)
269:         PairingsBn254.Fr memory lhs = PairingsBn254.copy(zeta_n);
270:         lhs.sub_assign(PairingsBn254.new_fr(1));
271:         require(lhs.value != 0); // we can not check a polynomial relationship if point `z` is i
n the domain
272:         lhs.mul_assign(proof.quotient_polynomial_at_z);
273:
274:         /** rhs = r(z) - pi(z) - (r_permu + r_lookup + r_substring)
275:          *
276:          *      alpha * (                                       |
277:          *          (w_0(z) + beta * sigma_0(z) + gamma)        |
278:          *          * (w_1(z) + beta * sigma_1(z) + gamma)      |  "r_permu"
279:          *          * (w_2(z) + beta * sigma_2(z) + gamma)      |
280:          *          * (w_3(z) + gamma) * z(zw)                  |
281:          *          + alpha * L1(z)                             |
282:          *      )                                               |
283:          *
284:          *      alpha^3 * (                                     |
285:          *          zlookup(zw) * gamma1                        |
286:          *              * (beta1 * s(zw) + (beta1+1)gamma1)     |  "r_lookup"
287:          *          + alpha * L1(z)                             |
288:          *      )                                               |
289:          *      alpha^5 * (                                     |
290:          *          - z_substring(zw)                           |  "r_substring"
291:          *      )                                               |
292:          */
293:
294:         /// rhs = r(z)
295:         PairingsBn254.Fr memory rhs = PairingsBn254.copy(
296:             proof.linearization_polynomial_at_z
297:         );
298:         // public inputs
299:         PairingsBn254.Fr memory inputs_term = PairingsBn254.new_fr(0);
300:         PairingsBn254.Fr memory tmp = PairingsBn254.new_fr(0);
301:         uint256 inputs_len = proof.input_values.length;
302:         for (uint256 i = 0; i < inputs_len; i++) {
303:             tmp.assign(state.cached_lagrange_evals[i]);
```

```
304:             tmp.mul_assign(PairingsBn254.new_fr(proof.input_values[i]));
305:             inputs_term.add_assign(tmp);
306:         }
307:
308:         /// rhs -= pi(x)
309:         rhs.sub_assign(inputs_term);
310:
311:         /// rhs -= "r_complement"
312:         PairingsBn254.Fr memory quotient_challenge = PairingsBn254.copy(
313:             state.alpha
314:         );
315:
316:         PairingsBn254.Fr memory r_permu = PairingsBn254.copy(
317:             proof.grand_product_at_z_omega
318:         );
319:         //"r_permu"
320:         for (uint256 i = 0; i < STATE_WIDTH - 1; i++) {
321:             tmp.assign(proof.permutation_polynomials_at_z[i]);
322:             tmp.mul_assign(state.beta);
323:             tmp.add_assign(state.gamma);
324:             tmp.add_assign(proof.wire_values_at_z[i]);
325:
326:             r_permu.mul_assign(tmp);
327:         }
328:         //(w_4(z) + gamma)
329:         tmp.assign(state.gamma);
330:         tmp.add_assign(proof.wire_values_at_z[STATE_WIDTH - 1]);
331:         r_permu.mul_assign(tmp);
332:
333:         //alpha * L1(z) (reuse var inputs_term)
334:         inputs_term.assign(state.cached_lagrange_evals[0]);
335:         inputs_term.mul_assign(state.alpha);
336:         r_permu.add_assign(inputs_term);
337:
338:         r_permu.mul_assign(state.alpha);
339:         rhs.sub_assign(r_permu);
340:
341:         //alpha^2
342:         quotient_challenge.mul_assign(state.alpha);
343:         //r_lookup (reuse var r_permu)
344:         r_permu.assign(proof.sorted_lookup_at_z_omega);
345:         r_permu.mul_assign(state.beta_1);
346:
```

```
347:         tmp.assign(state.beta_1);
348:         tmp.add_assign(PairingsBn254.new_fr(1));
349:         tmp.mul_assign(state.gamma_1);
350:         r_permu.add_assign(tmp);
351:
352:         r_permu.mul_assign(proof.grand_product_lookup_at_z_omega);
353:         r_permu.mul_assign(state.gamma_1);
354:
355:         // + alpha * L1(z)
356:         r_permu.add_assign(inputs_term);
357:
358:         //alpha^3
359:         quotient_challenge.mul_assign(state.alpha);
360:         // r_lookup * alpha^3
361:         r_permu.mul_assign(quotient_challenge);
362:
363:         rhs.sub_assign(r_permu);
364:
365:         //alpha^5
366:         quotient_challenge.mul_assign(state.alpha);
367:         quotient_challenge.mul_assign(state.alpha);
368:         // "r_substring" (reuse var)
369:         // alpha^5 * z_substring(zw)
370:         quotient_challenge.mul_assign(proof.z_substring_at_z_omega);
371:
372:         rhs.add_assign(quotient_challenge); //over
373:
374:         return lhs.value == rhs.value;
375:     }
```

Description

BradMoonUESTC : There is a discrepancy between the provided comment and the previously derived formula for the calculation of the z_substring term. The inconsistency lies in the coefficient used for this term.

In the comment, the expression is: `alpha^5 * (alpha^2 * L1[z] - 1) * [z_substring]`

However, the formula that was previously provided is: `(alpha^2 * L1[z] - 1) * v * alpha^5 * [z_substring]`

The primary difference between the two expressions is the coefficient. In the comment, the coefficient is `alpha^5`, while in the derived formula, it is `v * alpha^5.`

Recommendation

BradMoonUESTC : To ensure the correctness and consistency of the mathematical calculation, revise the formula to match the original comment. Update the coefficient in the formula for the z_substring term from `v * alpha^5` to `alpha^5`. The corrected formula should be:

`alpha^5 * (alpha^2 * L1[z] - 1) * [z_substring]`

By making this adjustment, the formula will align with the provided comment, reducing the potential for errors or misunderstandings in the implementation.

Client Response

Declined,In the comment, `v` as a scalar is outside of `[r]`. However, in the code, `v` is inlined to multiply every term of `[r]`.

UPV-3:Information Leakage in `batch_evaluate_lagrange_poly_out_of_domain` Function with a Potential Attack Scenario

Category	Severity	Status	Contributor
Logical	Medium	Declined	BradMoonUESTC

Code Reference

- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L173

```
173:            PairingsBn254.Fr memory return_zeta_pow_n
```

Description

BradMoonUESTC : The `batch_evaluate_lagrange_poly_out_of_domain` function returns an additional finite field element `return_zeta_pow_n`, which is a copy of `vanishing_at_z` calculated as `at.pow(domain_size)`. In cryptographic applications, the values `at` and `domain_size` may be sensitive as they could be related to encryption, decryption, or verification processes.

An attacker could potentially use the leaked `return_zeta_pow_n` value to gain insight into sensitive aspects of the system. For instance, they might attempt to deduce the values of at and `domain_size`, which could be critical parameters in the cryptographic system. If an attacker gains access to these values, they might exploit them to crack the cryptographic system, decrypt data, or forge signatures.

Attack Scenario: Suppose an attacker knows the function's signature and calls it with the following controlled input values:

poly_nums: [1, 2, 3] domain_size: 5 omega: 7 at: 3 The function returns the additional finite field element `return_zeta_pow_n`. The attacker receives this value and attempts to deduce the values of at and `domain_size`.

Initially, the attacker could try to calculate the discrete logarithm of `return_zeta_pow_n` to recover `domain_size`. Although the discrete logarithm problem is typically difficult in finite fields, attackers might find an efficient method for small domain sizes.

Assuming the attacker successfully recovers `domain_size`, they might use the `return_zeta_pow_n` value combined with the known `domain_size` to recover the value of `at`. If the attacker can determine at, they can potentially calculate the key k and compromise the entire encryption system.

Recommendation

BradMoonUESTC : To mitigate this potential vulnerability, it is advised to:

Remove the additional `return_zeta_pow_n` value from the function's return statement if it is not necessary for the intended application. If the `return_zeta_pow_n value` is required, ensure that it is handled securely and not leaked

or used improperly. Audit the rest of the code to make sure there are no other instances of sensitive information leakage. By addressing the information leakage in the `batch_evaluate_lagrange_poly_out_of_domain` function, the overall security of the cryptographic system can be improved.

Client Response

Declined,zeta_pow_n is originally public and should not lead to information leakage.

UPV-4:trust model is obscured in `UnipassVerifier` contract `verifyProof` function

Category	Severity	Status	Contributor
Logical	Medium	Declined	alansh

Code Reference

- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L464

```
464:            vk.g2_x = PairingsBn254.new_g2(tmpx, tmpy);
```

Description

alansh : The membership of G1 is checked in `new_g1_checked`, but not checked for G2 in `new_g2`, and `vk.omega` is also not checked.

Recommendation

alansh : Make it explicit that checking the validity of `vkdata` is the responsibility of the caller side, `verifyProof` should not do any checking. Keep in mind that `VerificationKey` is the commitment for the circuit, should be trusted once and reused subsequently, not to be passed from external each time, this may cause security issue in the future.

Client Response

Declined

UPV-5:Collision computation in `UnipassVerifier` contract `setupVKHash` function

Category	Severity	Status	Contributor
Increase Security	Low	Declined	newway55

Code Reference

- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L56-L84

```
56:     function setupVKHash(
57:         uint64 circuit_type,
58:         uint64 num_inputs,
59:         uint128 domain_size,
60:         uint256[] memory vkdata
61:     ) public adminOnly returns (bool) {
62:         if (circuit_type == 1024) {
63:             vk1024hash = sha256(
64:                 abi.encodePacked(num_inputs, domain_size, vkdata)
65:             );
66:             // console.log("domain_size 1024: %s", domain_size);
67:         } else if (circuit_type == 2048) {
68:             vk2048hash = sha256(
69:                 abi.encodePacked(num_inputs, domain_size, vkdata)
70:             );
71:             // console.log("domain_size 2048: %s", domain_size);
72:         } else if (circuit_type == 3) {
73:             vk2048trihash = sha256(
74:                 abi.encodePacked(num_inputs, domain_size, vkdata)
75:             );
76:         } else if (circuit_type == 4) {
77:             openIdhash = sha256(
78:                 abi.encodePacked(num_inputs, domain_size, vkdata)
79:             );
80:         } else {
81:             return false;
82:         }
83:         return true;
84:     }
```

Description

newway55 : The verification key need to be strong enough to not be tampered or modified in any way. Hashing its contents here is not enough secure, one way would be to add a layer of security to it.

Consider below POC contract

```
function testsetupVKHash() public {

        uint64 circuit_type = 1024;
        uint64 num_inputs = 3;
        uint128 domain_size = 256;
        uint256[] memory vkdata = new uint256[](4);

        // Call the function with the given parameters
        bool success = verifier.setupVKHash(circuit_type, num_inputs, domain_size, vkdata);

        // Check that the function returns true
        assertTrue(success, "setupVKHash should return true");

        // Check that the correct hash was set for circuit type 1024
        bytes32 expectedHash = keccak256(
            abi.encodePacked(
                keccak256(abi.encodePacked(block.timestamp, address(this))),
                circuit_type,
                num_inputs,
                domain_size,
                vkdata
            )
        );
        assertEq32(verifier.vk1024hash(), expectedHash, "Incorrect vk1024hash");

        // Check that the other hashes were not set
        assertEq(verifier.vk2048hash(), bytes32(0), "vk2048hash should be zero");
        assertEq(verifier.vk2048trihash(), bytes32(0), "vk2048trihash should be zero");
        assertEq(verifier.openIdhash(), bytes32(0), "openIdhash should be zero");
    }


}
```

Recommendation

newway55 : Improve the security of a hash function by adding an additional layer of randomness to the hash value :

- Use a salt value in the hash function to increase the complexity and make it more difficult for an attacker to compute a collision. Additionally, it may be useful to consider using a more secure hash function, such as SHA-3 or BLAKE2, which are designed to be resistant to collision attacks.

Consider below fix in the `setupVKHash` function

```
function setupVKHash(
    uint64 circuit_type,
    uint64 num_inputs,
    uint128 domain_size,
    uint256[] memory vkdata
) public adminOnly returns (bool) {
    bytes32 salt = keccak256(abi.encodePacked(block.timestamp, msg.sender));
    bytes32 hash = keccak256(
        abi.encodePacked(salt, circuit_type, num_inputs, domain_size, vkdata)
    );
    if (circuit_type == 1024) {
        vk1024hash = hash;
    } else if (circuit_type == 2048) {
        vk2048hash = hash;
    } else if (circuit_type == 3) {
        vk2048trihash = hash;
    } else if (circuit_type == 4) {
        openIdhash = hash;
    } else {
        return false;
    }
    return true;
}
```

Client Response

Declined,VKHash can only be modified by the administrator, and its format is restricted by the circuit.

UPV-6:Gas Optimization for Large Data Array.

Category	Severity	Status	Contributor
Gas Optimization	Low	Declined	lfzkoala

Code Reference

code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L56-L84

```
56:     function setupVKHash(
57:         uint64 circuit_type,
58:         uint64 num_inputs,
59:         uint128 domain_size,
60:         uint256[] memory vkdata
61:     ) public adminOnly returns (bool) {
62:         if (circuit_type == 1024) {
63:             vk1024hash = sha256(
64:                 abi.encodePacked(num_inputs, domain_size, vkdata)
65:             );
66:             // console.log("domain_size 1024: %s", domain_size);
67:         } else if (circuit_type == 2048) {
68:             vk2048hash = sha256(
69:                 abi.encodePacked(num_inputs, domain_size, vkdata)
70:             );
71:             // console.log("domain_size 2048: %s", domain_size);
72:         } else if (circuit_type == 3) {
73:             vk2048trihash = sha256(
74:                 abi.encodePacked(num_inputs, domain_size, vkdata)
75:             );
76:         } else if (circuit_type == 4) {
77:             openIdhash = sha256(
78:                 abi.encodePacked(num_inputs, domain_size, vkdata)
79:             );
80:         } else {
81:             return false;
82:         }
83:         return true;
84:     }
```

Description

lfzkoala : The `setupVKHash()` function takes in several parameters, including `num_inputs`, `domain_size`, and `vkdata`. These parameters are used to calculate a hash value using the `sha256()` function, which takes in a single parameter: a packed ABI-encoded byte array containing the concatenation of the input parameters.

One potential issue is that the sha256() function has a fixed gas cost of 63 gas per word (or 32 bytes), plus a base cost of 757 gas. This means that if the `vkdata` array is very large, the function could become expensive to execute in terms of gas costs. This could be mitigated by breaking up the `vkdata` array into smaller chunks and hashing each chunk separately, although this would require additional code to reassemble the hashes into a single hash value.

Recommendation

lfzkoala : I'd recommend UniPass considering whether you care about the similar gas cost issue. If so, you can optimize the code to optimize gas fee while running the contract

Client Response

Declined,Generally, VK will not be particularly large, and this part needs to be consistent with the circuit.

UPV-7:Integer Overflow and Underflow risk in `UnipassVerifier.sol` contract `sha256PaddingLen` function

Category	Severity	Status	Contributor
Integer Overflow and Underflow	Low	Declined	newway55

Code Reference

- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L132-L140

```
132:    function sha256PaddingLen(uint256 input_len) public pure returns (uint256) {
133:        uint256 input_remainder = (input_len * 8) % 512;
134:        uint256 padding_count = 0;
135:        if (input_remainder < 448) {
136:            padding_count = (448 - input_remainder) / 8;
137:        } else {
138:            padding_count = (448 + 512 - input_remainder) / 8;
139:        }
140:        return input_len + padding_count + 8;
```

Description

newway55 : The function may face issues with the input value of input_len causing integer overflow or underflow. If input_len is a very large value, the calculation of input_len + padding_count + 8 could exceed the maximum value that can be stored in a uint256 variable, resulting in an integer overflow. If input_remainder is a very small value, the calculation of padding_count can result in an integer underflow.

This function also have impact on verification key setup so it's important to make sure security checks.

Consider below POC contract

Log error :

```
[25458] PlonkTest::testSha256PaddingLen()      ├─ [1014] UnipassVerifier::sha256PaddingLen(100) [staticcall]     │   └─ ← 128     ├─ emit log_named_string(key: Error, val: Padding length calculation failed)     ├─ emit log(: Error: a == b not satisfied [uint])     ├─ emit log_named_uint(key:       Left, val: 128)     ├─ emit log_named_uint(key:      Right, val: 136)     ├─ [0] VM::store(VM: [0x7109709ECfa91a80626fF3989D68f67F5b1DD12D], 0x6661696c6564000000000000000000000000000000000000000000000000000000, 0x0000000000000000000000000000000000000000000000000000000000000001)      │   └─ ← ()     ├─ [382] UnipassVerifie-
```

`r::sha256PaddingLen(115792089237316195423570985008687907853269984665640564039457584007913129639927)  └─ ← "Arithmetic over/underflow"  └─ ← ()`

- Check the padding length calculation with a normal input length value of 1000.
- Check if the function can detect potential overflow by setting the input length to the maximum possible value for a uint256.

```
 function testSha256PaddingLen() public {
        uint256 input_len = 1000;
        uint256 result = verifier.sha256PaddingLen(input_len);
        uint256 expected = 1056;
        assertEq(result, expected, "Padding length calculation failed");

        input_len = type(uint256).max - 8;
        result = verifier.sha256PaddingLen(input_len);
        expected = input_len + 72; // 64 (padding) + 8 (message length)
        assertEq(result, expected, "Potential overflow not detected");
}

}
```

Recommendation

newway55 : - Include a range check to ensure that the value of padding_count is within the acceptable range

Consider below fix in the `sha256PaddingLen()` function

```
function sha256PaddingLen(uint256 input_len) public pure returns (uint256) {

        uint256 input_remainder = (input_len * 8) % 512;
        uint256 padding_count = 0;
        if (input_remainder < 448) {
            padding_count = (448 - input_remainder) / 8;
        } else {
            padding_count = (448 + 512 - input_remainder) / 8;
        }
        require(padding_count <= type(uint256).max - input_len - 8, "Potential overflow");

        return input_len + padding_count + 8;
    }
```

Client Response

Declined,Solidity 8.0 and above will come with overflow checking.

UPV-8:Integer Overflow and Underflow risk in `UnipassVerifier` contract `verifyProof` function

Category	Severity	Status	Contributor
Integer Overflow and Underflow	Low	Declined	newway55

Code Reference

- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L394-L483

```
394:     }
395:
396:     function verifyProof(
397:         bytes32 vkhash,
398:         uint128 domain_size,
399:         uint256[] memory vkdata,
400:         uint256[] memory public_inputs,
401:         uint256[] memory serialized_proof
402:     ) public view returns (bool) {
403:         VerificationKey memory vk;
404:         vk.domain_size = domain_size;
405:         vk.num_inputs = uint64(public_inputs.length);
406:         vk.omega = PairingsBn254.new_fr(vkdata[0]);
407:
408:         {
409:             uint256 j = 1;
410:             for (uint256 i = 0; i < STATE_WIDTH + 2 + 2; ) {
411:                 vk.selector_commitments[i] = PairingsBn254.new_g1_checked(
412:                     vkdata[j],
413:                     vkdata[j + 1]
414:                 );
415:                 j += 2;
416:                 unchecked {
417:                     ++i;
418:                 }
419:             }
420:
421:             for (uint256 i = 0; i < STATE_WIDTH; ) {
422:                 vk.permutation_commitments[i] = PairingsBn254.new_g1_checked(
423:                     vkdata[j],
424:                     vkdata[j + 1]
425:                 );
426:                 j += 2;
427:                 unchecked {
428:                     ++i;
429:                 }
430:             }
431:
432:             for (uint256 i = 0; i < STATE_WIDTH + 1; ) {
433:                 vk.tables_commitments[i] = PairingsBn254.new_g1_checked(
434:                     vkdata[j],
435:                     vkdata[j + 1]
```

```
436:                 );
437:                 j += 2;
438:                 unchecked {
439:                     ++i;
440:                 }
441:             }
442:
443:             // q_substring q_substring_r
444:             for (uint256 i = 0; i < 2; ) {
445:                 vk.selector_commitments[STATE_WIDTH + 2 + 2 + i] = PairingsBn254
446:                     .new_g1_checked(vkdata[j], vkdata[j + 1]);
447:                 j += 2;
448:                 unchecked {
449:                     ++i;
450:                 }
451:             }
452:
453:             uint256[2] memory tmpx;
454:             uint256[2] memory tmpy;
455:
456:             tmpx[1] = vkdata[j];
457:             j += 1;
458:             tmpx[0] = vkdata[j];
459:             j += 1;
460:             tmpy[1] = vkdata[j];
461:             j += 1;
462:             tmpy[0] = vkdata[j];
463:             j += 1;
464:             vk.g2_x = PairingsBn254.new_g2(tmpx, tmpy);
465:         }
466:         Proof memory proof = deserialize_proof(public_inputs, serialized_proof);
467:
468:         PartialVerifierState memory state;
469:
470:         (bool res, PairingsBn254.Fr memory return_zeta_pow_n) = verify_initial(
471:             state,
472:             proof,
473:             vk,
474:             vkhash
475:         );
476:         if (res == false) {
477:             return false;
478:         }
```

```
479:
480:         bool success = verify_commitments(return_zeta_pow_n, state, proof, vk);
481:
482:         return success;
483:     }
```

Description

newway55 : There is no "if" statement which checks that "i" is larger than STATE_WIDTH, even if there a check in the for initialization, but inside unchecked incrementation is being proceeded and no check is done. This could cause a security loop to exploit. vkdata is the read data of Verification Key. A wrong location of key-value pair will lead to a false commitment. This can be exploited by a hacker who knows when it will override.

Consider below POC contract

Logs : [15385] PlonkTest::testVerifyProofOverflow() ├─ [8815] UnipassVerifier::verifyProof(0xe8e478e45ed03cdfe886096ff999065b381db27e1b18be495523b62f76b37952, 340282366920938463463374607431768211455, [0, 0, 0, 0, 0, 0, 0, 0], [], []) [staticcall] │ └─ ← "Index out of bounds" └─ ← "Index out of bounds"

```
function testVerifyProofOverflow() public {
        // set domain_size to the max value of uint128 to test for potential overflow
        uint128 domain_size = type(uint128).max;
        uint256[] memory vkdata = new uint256[](8);
        uint256[] memory public_inputs;
        uint256[] memory serialized_proof;

        bytes32 vkhash = keccak256(abi.encodePacked("verification key hash"));

        // call the function and check that it returns false to indicate failure due to overflow
        bool success = verifier.verifyProof(vkhash, domain_size, vkdata, public_inputs, serialized_p
roof);
        assertEq(success, false, "Function should return false due to potential overflow");
}


}
```

Recommendation

newway55 : Ensure the loop does not run beyond the bounds of the array.

Consider below fix in the `verifyProof()` function

```
......

for (uint256 i = 0; i < STATE_WIDTH + 2 + 2; i++) {
    if (i >= STATE_WIDTH + 2 + 2) {
        break; // ensure the loop does not run beyond the bounds of the array
    }
    vk.selector_commitments[i] = PairingsBn254.new_g1_checked(
        vkdata[j],
        vkdata[j + 1]
    );
    j += 2;
}

for (uint256 i = 0; i < STATE_WIDTH; i++) {
    if (i >= STATE_WIDTH) {
        break; // ensure the loop does not run beyond the bounds of the array
    }
    vk.permutation_commitments[i] = PairingsBn254.new_g1_checked(
        vkdata[j],
        vkdata[j + 1]
    );
    j += 2;
}

for (uint256 i = 0; i < STATE_WIDTH + 1; i++) {
    if (i >= STATE_WIDTH + 1) {
        break; // ensure the loop does not run beyond the bounds of the array
    }
    vk.tables_commitments[i] = PairingsBn254.new_g1_checked(
        vkdata[j],
        vkdata[j + 1]
    );
    j += 2;
}

for (uint256 i = 0; i < 2; i++) {
    if (i >= 2) {
        break; // ensure the loop does not run beyond the bounds of the array
    }
    vk.selector_commitments[STATE_WIDTH + 2 + 2 + i] = PairingsBn254
        .new_g1_checked(vkdata[j], vkdata[j + 1]);
    j += 2;
```

```
    }

    ......
```

Client Response

Declined,Can't understand what problem with the code its description points out.

UPV-9:Logical error in `UnipassVerifier.sol` contract `setBit` function

Category	Severity	Status	Contributor
Logical	Low	Declined	newway55

Code Reference

- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L86-L88

```
86:     function setBit(bytes1 a, uint8 n) public pure returns (bytes1) {
87:         return a | bytes1(uint8(2 ** (7 - n)));
88:     }
```

Description

newway55 : setBit function is that it returns the original byte value if the specified bit is already set to 1. This means that if you call setBit with an input byte that already has the specified bit set to 1, the function will not set the bit to 1 again.

Consider below POC contract

`forge test -vvvv` returns always this error no matter what inputs : Error: a == b not satisfied [bytes32] Left: 0x2000000000000000000000000000000000000000000000000000000000000000 Right: 0x0400000000000000000000000000000000000000000000000000000000000000

```
{
function testSetBit() public {
    // Test setting the 3rd bit (index 2) of the byte 0b00000000
    bytes1 result = verifier.setBit(bytes1(0x00), 2);
    bytes1 expected = bytes1(0x04);
    assertEq(result, expected, "Setting bit failed");

    // Test setting the 5th bit (index 3) of the byte 0b01010101
    result = verifier.setBit(bytes1(0x55), 3);
    expected = bytes1(0x5d);
    assertEq(result, expected, "Setting bit failed");

    // Test setting the 1st bit (index 7) of the byte 0b10101010
    result = verifier.setBit(bytes1(0xaa), 7);
    expected = bytes1(0x82);
    assertEq(result, expected, "Setting bit failed");

    // Test setting the 4th bit (index 4) of the byte 0b11110000
    result = verifier.setBit(bytes1(0xf0), 4);
    expected = bytes1(0xf8);
    assertEq(result, expected, "Setting bit failed");

    // Test setting the 2nd bit (index 6) of the byte 0b00100000
    result = verifier.setBit(bytes1(0x20), 6);
    expected = bytes1(0x60);
    assertEq(result, expected, "Setting bit failed");
}


}
```

Recommendation

newway55 : Always set the specified bit to 1, regardless of its original value. This version of the function creates a mask with a 1 in the specified bit position, and then overrides the mask with the original byte value to set the specified bit to 1, regardless of its original value.

Consider below fix in the `setBit()` function

```
function setBit(bytes1 a, uint8 n) public pure returns (bytes1) {
    bytes1 mask = bytes1(uint8(2 ** (7 - n)));
    return a | mask;
}
```

Client Response

Declined,After testing, this suggestion is meaningless and needs to be supplemented with a more accurate description.

UPV-10:abi.encodePacked() should not be used with dynamic types when passing the result to a hash function such as keccack256()

Category	Severity	Status	Contributor
Language Specific	Low	Declined	thereksfour

Code Reference

- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L159-L169
- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L196-L206
- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L248-L258
- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L286-L293

```
159:         bytes32 hash_result = sha256(
160:             abi.encodePacked(
161:                 public_params.header_hash,
162:                 public_params.addr_hash,
163:                 bit_location_a,
164:                 bit_location_b,
165:                 public_params.pub_match_hash,
166:                 uint16(sha256PaddingLen(public_params.header_len) / 64),
167:                 uint16(sha256PaddingLen(public_params.from_len + 32) / 64)
168:             )
169:         );

196:         bytes32 hash_result = sha256(
197:             abi.encodePacked(
198:                 public_params.header_hash,
199:                 public_params.addr_hash,
200:                 bit_location_a,
201:                 bit_location_b,
202:                 public_params.pub_match_hash,
203:                 uint16(sha256PaddingLen(public_params.header_len) / 64),
204:                 uint16(sha256PaddingLen(public_params.from_len + 32) / 64)
205:             )
206:         );

248:         bytes memory sha256_input = abi.encodePacked(
249:             public_params.concat_hash,
250:             location_id_token_1,
251:             location_payload_base64,
252:             location_id_token_2,
253:             location_header_base64,
254:             location_payload_raw,
255:             location_email_addr
256:         );
257:
258:         bytes32 hash_result = sha256(sha256_input);

286:             bytes32 r = sha256(
287:                 abi.encodePacked(
288:                     public_params[i].header_hash,
289:                     public_params[i].addr_hash,
290:                     bit_location_a,
291:                     bit_location_b
```

```
292:                 )
293:             );
```

Description

thereksfour : abi.encodePacked can result in hash collisions when used with two dynamic arguments (string/bytes). Use abi.encode() instead which will pad items to 32 bytes, which will prevent hash collisions (e.g. abi.encodePacked(0x123,0x456) => 0x123456 => abi.encodePacked(0x1,0x23456), but abi.encode(0x123,0x456) => 0x0...1230...456). "Unless there is a compelling reason, abi.encode should be preferred". If there is only one argument to abi.encodePacked() it can often be cast to bytes() or bytes32() instead. If all arguments are strings and or bytes, bytes.concat() should be used instead

Instances(4):

code/UniPass-verifier-contract/contracts/UnipassVerifier.sol

```
159: bytes32 hash_result = sha256(
            abi.encodePacked(
                public_params.header_hash,
                public_params.addr_hash,
                bit_location_a,                        // bytes
                bit_location_b,                       // bytes
                public_params.pub_match_hash,
                uint16(sha256PaddingLen(public_params.header_len) / 64),
                uint16(sha256PaddingLen(public_params.from_len + 32) / 64)
            )
        );
...
196: bytes32 hash_result = sha256(
            abi.encodePacked(
                public_params.header_hash,
                public_params.addr_hash,
                bit_location_a,                        // bytes
                bit_location_b,                       // bytes
                public_params.pub_match_hash,
                uint16(sha256PaddingLen(public_params.header_len) / 64),
                uint16(sha256PaddingLen(public_params.from_len + 32) / 64)
            )
        );
...
248: bytes memory sha256_input = abi.encodePacked(
            public_params.concat_hash,
            location_id_token_1,               // bytes
            location_payload_base64,     // bytes
            location_id_token_2,              // bytes
            location_header_base64,      // bytes
            location_payload_raw,           // bytes
            location_email_addr               // bytes
        );
...
286:     bytes32 r = sha256(
                abi.encodePacked(
                    public_params[i].header_hash,
                    public_params[i].addr_hash,
                    bit_location_a,                // bytes
                    bit_location_b                 // bytes
                )
```

```
    );
```

There is also discussion of removing abi.encodePacked from future versions of Solidity (https://github.com/ethereum/solidity/issues/11593), so using abi.encode now will ensure compatibility in the future.

Recommendation

thereksfour : Use abi.encode instead.

Client Response

Declined,It needs to be consistent with the circuit design, and the type or length of all parameters inside is fixed in the contract and cannot be modified by the user

UPV-11:Cache the for loop count variable in memory

Category	Severity	Status	Contributor
Gas Optimization	Informational	Declined	thereksfour

Code Reference

- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L320
- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L410
- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L421
- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L432
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L438
- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L501
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L509
- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L536
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L554
- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L558
- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L568
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L698
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L734
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L741
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L764
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L802
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L809
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L924
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L943
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L981
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L984

```
320:        for (uint256 i = 0; i < STATE_WIDTH - 1; i++) {

410:            for (uint256 i = 0; i < STATE_WIDTH + 2 + 2; ) {

421:            for (uint256 i = 0; i < STATE_WIDTH; ) {

432:            for (uint256 i = 0; i < STATE_WIDTH + 1; ) {

438:        for (uint256 i = 1; i < STATE_WIDTH; i++) {

501:        for (uint256 i = 0; i < STATE_WIDTH; ) {

509:        for (uint256 i = 1; i < STATE_WIDTH - 1; i++) {

536:        for (uint256 i = 0; i < STATE_WIDTH; ) {

554:        for (uint256 i = 1; i < STATE_WIDTH; i++) {

558:        for (uint256 i = 0; i < STATE_WIDTH; ) {

568:        for (uint256 i = 0; i < STATE_WIDTH - 1; ) {

698:        for (uint256 i = 1; i < STATE_WIDTH; i++) {

734:        for (uint256 i = 3; i < STATE_WIDTH; i++) {

741:        for (uint256 i = 0; i < STATE_WIDTH - 1; i++) {

764:        for (uint256 i = 0; i < STATE_WIDTH; i++) {

802:        for (uint256 i = 0; i < STATE_WIDTH; i++) {

809:        for (uint256 i = 0; i < STATE_WIDTH - 1; i++) {

924:        for (uint256 i = 0; i < STATE_WIDTH; i++) {

943:        for (uint256 i = 0; i < STATE_WIDTH; i++) {

981:        for (uint256 i = 0; i < STATE_WIDTH; i++) {

984:        for (uint256 i = 0; i < STATE_WIDTH - 1; i++) {
```

Description

thereksfour : Some variables used to determine the count of for loops are storage types, and caching them as memory variables can reduce gas consumption.

```
File: UniPass-verifier-contract/contracts/UnipassVerifier.sol

410:             for (uint256 i = 0; i < STATE_WIDTH + 2 + 2; ) {

421:             for (uint256 i = 0; i < STATE_WIDTH; ) {

432:             for (uint256 i = 0; i < STATE_WIDTH + 1; ) {

501:         for (uint256 i = 0; i < STATE_WIDTH; ) {

536:         for (uint256 i = 0; i < STATE_WIDTH; ) {

558:         for (uint256 i = 0; i < STATE_WIDTH; ) {

568:         for (uint256 i = 0; i < STATE_WIDTH - 1; ) {
```

```
File: UniPass-verifier-contract/contracts/PlookupSingleCore.sol

320:         for (uint256 i = 0; i < STATE_WIDTH - 1; i++) {

438:         for (uint256 i = 1; i < STATE_WIDTH; i++) {

509:         for (uint256 i = 1; i < STATE_WIDTH - 1; i++) {

554:         for (uint256 i = 1; i < STATE_WIDTH; i++) {

698:         for (uint256 i = 1; i < STATE_WIDTH; i++) {

734:         for (uint256 i = 3; i < STATE_WIDTH; i++) {

741:         for (uint256 i = 0; i < STATE_WIDTH - 1; i++) {

764:         for (uint256 i = 0; i < STATE_WIDTH; i++) {

802:         for (uint256 i = 0; i < STATE_WIDTH; i++) {

809:         for (uint256 i = 0; i < STATE_WIDTH - 1; i++) {

924:         for (uint256 i = 0; i < STATE_WIDTH; i++) {

943:         for (uint256 i = 0; i < STATE_WIDTH; i++) {

981:         for (uint256 i = 0; i < STATE_WIDTH; i++) {

984:         for (uint256 i = 0; i < STATE_WIDTH - 1; i++) {
```

Recommendation

thereksfour : Cache the STATE_WIDTH used in the for loop in memory variable

Client Response

Declined,After testing, this suggestion does not reduce gas consumption.

UPV-12:Code Style in `UnipassVerifier` contract `deserialize_proof` function

Category	Severity	Status	Contributor
Code Style	Informational	Declined	newway55

Code Reference

- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L548-L556

```
548:         proof.quotient_polynomial_at_z = PairingsBn254.new_fr(
549:             serialized_proof[j]
550:         );
551:         j += 1;
552:         //r(z)
553:         proof.linearization_polynomial_at_z = PairingsBn254.new_fr(
554:             serialized_proof[j]
555:         );
556:         j += 1;
```

Description

newway55 : Plonk proof relies on field arithmetic and invalid field elements could lead to incorrect computation and verification of the proof.

Consider below POC contract

```
function testDeserializeProof() public {
    // Set up input data
    uint256[] memory public_inputs = new uint256[](3);
    public_inputs[0] = 1;
    public_inputs[1] = 2;
    public_inputs[2] = 3;

    uint256[] memory serialized_proof = new uint256[](47);
    for (uint256 i = 0; i < serialized_proof.length; i++) {
        serialized_proof[i] = i;
    }

    // Call the function to deserialize the proof
    Proof memory proof = deserialize_proof(public_inputs, serialized_proof);

    // Check that input_values are set correctly
    assertEq(proof.input_values[0], 1, "Incorrect input value");
    assertEq(proof.input_values[1], 2, "Incorrect input value");
    assertEq(proof.input_values[2], 3, "Incorrect input value");

    // Check that wire_commitments are set correctly
    for (uint256 i = 0; i < STATE_WIDTH; i++) {
        uint256 j = i * 2;
        assert(proof.wire_commitments[i].X == serialized_proof[j], "Incorrect wire commitment");
        assert(proof.wire_commitments[i].Y == serialized_proof[j + 1], "Incorrect wire commitment");
    }

    // Check that sorted_lookup_commitment is set correctly
    uint256 j = STATE_WIDTH * 2;
    assert(proof.sorted_lookup_commitment.X == serialized_proof[j], "Incorrect sorted lookup commitm
ent");
    assert(proof.sorted_lookup_commitment.Y == serialized_proof[j + 1], "Incorrect sorted lookup com
mitment");

    // Check that grand_product_commitment is set correctly
    j += 2;
    assert(proof.grand_product_commitment.X == serialized_proof[j], "Incorrect grand product commitm
ent");
    assert(proof.grand_product_commitment.Y == serialized_proof[j + 1], "Incorrect grand product com
mitment");
```

```
// Check that grand_product_lookup_commitment is set correctly
    j += 2;
    assert(proof.grand_product_lookup_commitment.X == serialized_proof[j], "Incorrect grand product
lookup commitment");
    assert(proof.grand_product_lookup_commitment.Y == serialized_proof[j + 1], "Incorrect grand prod
uct lookup commitment");

    // Check that z_substring_commitment is set correctly
    j += 2;
    assert(proof.z_substring_commitment.X == serialized_proof[j], "Incorrect z substring commitmen
t");
    assert(proof.z_substring_commitment.Y == serialized_proof[j + 1], "Incorrect z substring commitm
ent");

    // Check that quotient_poly_commitments are set correctly
    for (uint256 i = 0; i < STATE_WIDTH; i++) {
        j += 2;
        assert(proof.quotient_poly_commitments[i].X == serialized_proof[j], "Incorrect quotient poly
nomial commitment");
        assert(proof.quotient_poly_commitments[i].Y == serialized_proof[j + 1], "Incorrect quotient
polynomial commitment");
    }

    // Check that quotient_polynomial_at_z is set correctly
    j += 1;
    assert(proof.quotient_polynomial_at_z == PairingsBn254.new_fr(serialized_proof[j]), "Incorrect q
uotient polynomial at z");

    // Check that linearization_polynomial_at_z is set correctly
    j += 1;
    assert(proof.linearization_polynomial_at_z == PairingsBn254.new_fr(serialized_proof[j]), "Incorr
ect linearization polynomial at z");
}
function testDeserializeProof() public {
    uint64 circuit_type = 1024;
    uint64 num_inputs = 3;
    uint128 domain_size = 256;
    uint256[] memory vkdata = new uint256[](4);


    verifier.setupVKHash(circuit_type, num_inputs, domain_size, vkdata);

    // define sample public inputs and serialized proof
```

```
uint256[] memory public_inputs = new uint256[](3);
    public_inputs[0] = 111;
    public_inputs[1] = 222;
    public_inputs[2] = 333;

    uint256[] memory serialized_proof = new uint256[](64);

    // call the deserialize_proof() function with the given inputs

    proved = verifier.deserialize_proof(public_inputs, serialized_proof);

    // check that the returned struct contains valid values
    assert(proof.input_values.length == 3, "Incorrect input_values length");
    assert(proof.wire_commitments.length == STATE_WIDTH, "Incorrect wire_commitments length");
    assert(proof.quotient_poly_commitments.length == STATE_WIDTH, "Incorrect quotient_poly_commitmen
ts length");
    assert(proof.wire_values_at_z.length == STATE_WIDTH, "Incorrect wire_values_at_z length");
    assert(proof.permutation_polynomials_at_z.length == STATE_WIDTH - 1, "Incorrect permutation_poly
nomials_at_z length");
}

}

}
```

Recommendation

newway55 : Add verification of the remaining values using the PairingsBn254.new_fr() function. This ensures that the values are valid field elements and are consistent with the input public_inputs array :

- require() statement to check that the field element is valid. If the field element is not valid, the require() statement will cause the function to revert with an error message.

Consider below fix in the `deserialize_proof()` function

```
....


proof.quotient_polynomial_at_z = PairingsBn254.new_fr(
    serialized_proof[j]
);
require(
    proof.quotient_polynomial_at_z.is_valid(),
    "Invalid quotient polynomial at z"
);
j += 1;

....
```

Client Response

Declined,Fr is a finite field, there is no overflow problem.

UPV-13:Don't initialize variables with default value

Category	Severity	Status	Contributor
Gas Optimization	Informational	Declined	thereksfour

Code Reference

- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L99
- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L102
- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L111
- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L119
- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L123
- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L134
- code/UniPass-verifier-contract/contracts/ZkTest.sol#L146
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L198
- code/UniPass-verifier-contract/contracts/PlonkCoreLib.sol#L215
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L243
- code/UniPass-verifier-contract/contracts/PlonkCoreLib.sol#L255
- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L276
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L302
- code/UniPass-verifier-contract/contracts/PlonkCoreLib.sol#L308
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L320
- code/UniPass-verifier-contract/contracts/PlonkCoreLib.sol#L356
- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L410
- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L421
- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L432
- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L444
- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L492
- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L500
- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L501
- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L536
- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L558
- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L568
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L741
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L764
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L802
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L809
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L919
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L924
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L943

- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L949
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L958
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L981
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L984

```
99:        for (uint256 i = 0; i < b_len / 8; i++) {

102:        for (uint256 i = 0; i < b_len % 8; i++) {

111:        for (uint256 i = 0; i < tmp_index; i++) {

119:        for (uint256 i = 0; i < tmp_index / 8; i++) {

123:        for (uint256 i = 0; i < tmp_index % 8; i++) {

134:        uint256 padding_count = 0;

146:        for (uint256 i = 0; i < 3; i++) {

198:        for (uint256 i = 0; i < poly_nums_len; i++) {

215:            bool success = false;

243:        for (uint256 i = 0; i < nums_len; i++) {

255:            bool success = false;

276:        for (uint256 i = 0; i < 3; i++) {

302:        for (uint256 i = 0; i < inputs_len; i++) {

308:        for (uint256 i = 0; i < elements; i++) {

320:        for (uint256 i = 0; i < STATE_WIDTH - 1; i++) {

356:    uint32 constant private DST_0 = 0;

410:            for (uint256 i = 0; i < STATE_WIDTH + 2 + 2; ) {

421:            for (uint256 i = 0; i < STATE_WIDTH; ) {

432:            for (uint256 i = 0; i < STATE_WIDTH + 1; ) {

444:            for (uint256 i = 0; i < 2; ) {

492:        for (uint256 i = 0; i < inputs_len; ) {
```

```
500:        uint256 j = 0;
501:        for (uint256 i = 0; i < STATE_WIDTH; ) {
536:        for (uint256 i = 0; i < STATE_WIDTH; ) {
558:        for (uint256 i = 0; i < STATE_WIDTH; ) {
568:        for (uint256 i = 0; i < STATE_WIDTH - 1; ) {
741:        for (uint256 i = 0; i < STATE_WIDTH - 1; i++) {
764:        for (uint256 i = 0; i < STATE_WIDTH; i++) {
802:        for (uint256 i = 0; i < STATE_WIDTH; i++) {
809:        for (uint256 i = 0; i < STATE_WIDTH - 1; i++) {
919:        for (uint256 i = 0; i < vk.num_inputs; i++) {
924:        for (uint256 i = 0; i < STATE_WIDTH; i++) {
943:        for (uint256 i = 0; i < STATE_WIDTH; i++) {
949:        uint256 tmp = 0;
958:        for (uint256 i = 0; i < lagrange_poly_len; i++) {
981:        for (uint256 i = 0; i < STATE_WIDTH; i++) {
984:        for (uint256 i = 0; i < STATE_WIDTH - 1; i++) {
```

Description

thereksfour : If a variable is not initialized, it is assumed to have the default value. Explicitly initializing a variable with its default value costs unnecessary gas.

For more info, see Mudit Gupta's Blog at point "No need to initialize variables with default values".

Therefore, following variable declarations could be refactored:

Instances (37):

```
File: UniPass-verifier-contract/contracts/PlonkCoreLib.sol

215:             bool success = false;

255:             bool success = false;

308:         for (uint256 i = 0; i < elements; i++) {

356:     uint32 constant private DST_0 = 0;
```

File: UniPass-verifier-contract/contracts/PlookupSingleCore.sol

```
198:         for (uint256 i = 0; i < poly_nums_len; i++) {

243:         for (uint256 i = 0; i < nums_len; i++) {

302:         for (uint256 i = 0; i < inputs_len; i++) {

320:         for (uint256 i = 0; i < STATE_WIDTH - 1; i++) {

741:         for (uint256 i = 0; i < STATE_WIDTH - 1; i++) {

764:         for (uint256 i = 0; i < STATE_WIDTH; i++) {

802:         for (uint256 i = 0; i < STATE_WIDTH; i++) {

809:         for (uint256 i = 0; i < STATE_WIDTH - 1; i++) {

919:         for (uint256 i = 0; i < vk.num_inputs; i++) {

924:         for (uint256 i = 0; i < STATE_WIDTH; i++) {

943:         for (uint256 i = 0; i < STATE_WIDTH; i++) {

949:         uint256 tmp = 0;

958:         for (uint256 i = 0; i < lagrange_poly_len; i++) {

981:         for (uint256 i = 0; i < STATE_WIDTH; i++) {

984:         for (uint256 i = 0; i < STATE_WIDTH - 1; i++) {
```

```
File: UniPass-verifier-contract/contracts/UnipassVerifier.sol

99:         for (uint256 i = 0; i < b_len / 8; i++) {

102:         for (uint256 i = 0; i < b_len % 8; i++) {

111:         for (uint256 i = 0; i < tmp_index; i++) {

119:         for (uint256 i = 0; i < tmp_index / 8; i++) {

123:         for (uint256 i = 0; i < tmp_index % 8; i++) {

134:         uint256 padding_count = 0;

276:         for (uint256 i = 0; i < 3; i++) {

410:             for (uint256 i = 0; i < STATE_WIDTH + 2 + 2; ) {

421:             for (uint256 i = 0; i < STATE_WIDTH; ) {

432:             for (uint256 i = 0; i < STATE_WIDTH + 1; ) {

444:             for (uint256 i = 0; i < 2; ) {

492:         for (uint256 i = 0; i < inputs_len; ) {

500:         uint256 j = 0;

501:         for (uint256 i = 0; i < STATE_WIDTH; ) {

536:         for (uint256 i = 0; i < STATE_WIDTH; ) {

558:         for (uint256 i = 0; i < STATE_WIDTH; ) {

568:         for (uint256 i = 0; i < STATE_WIDTH - 1; ) {


File: UniPass-verifier-contract/contracts/ZkTest.sol

146:         for (uint256 i = 0; i < 3; i++) {
```

Recommendation

thereksfour : Don't initialize variables with default value

Client Response

Declined,After testing, this suggestion does not reduce gas consumption.

UPV-14:Insufficient Input Validation

Category	Severity	Status	Contributor
Code Style	Informational	Fixed	BradMoonUESTC

Code Reference

- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L90-L96

```
90:     function bitLocation(
91:         uint32 b_left_index,
92:         uint32 b_len,
93:         uint32 maxaLen,
94:         uint32 maxbLen
95:     ) public pure returns (bytes memory, bytes memory) {
96:         bytes memory bit_location_a = new bytes(maxaLen / 8);
```

Description

BradMoonUESTC : The bitLocation function accepts parameters b_left_index, b_len, maxaLen, and maxbLen, but it does not perform any input validation checks. Without proper validation, the function may exhibit unexpected behavior or errors.

Recommendation

BradMoonUESTC : Add appropriate require statements to validate the input parameters. Ensure that the values of b_left_index, b_len, maxaLen, and maxbLen are within valid ranges to prevent any unforeseen issues. For example:

```
require(b_left_index < maxaLen * 8, "b_left_index is out of range");
require(b_len <= maxbLen * 8, "b_len is too large");
require(maxaLen > 0 && maxbLen > 0, "maxaLen and maxbLen must be greater than zero");
```

Adding these checks will increase the robustness and security of the function by ensuring that the input parameters are within the expected ranges.

Client Response

Fixed,maxaLen and maxbLen are hard-coded in the contract, so there is no need to check, other parameters can be checked.

UPV-15:Lack Security Proof

Category	Severity	Status	Contributor
Code Style	Informational	Mitigated	lfzkoala

Code Reference

code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L8

```
8:contract Plonk4SingleVerifierWithAccessToDNext {
```

Description

lfzkoala : UniPass implements an variant circuit with different width and custom gates. This is different than the original Plonk protocol. Overall the implementation of the variant protocol looks correct to me but the document doesn't give a rigorous security proof for this variant.

Recommendation

lfzkoala : I'd highly recommend UniPass write down a rigorous security and cryptography proof for this variant and circuit design. This could be a critical level issue if the proof doesn't work.

Client Response

Mitigated,All Plonk libraries, such as Halo2, have absorbed some of the latest developments in the industry, but the comprehensive security proofs for these changes are difficult to summarize, and may be added later.

UPV-16:Logical risk in `UnipassVerifier.sol` contract `verifyV2048` function

Category	Severity	Status	Contributor
Logical risk	Informational	Declined	newway55

Code Reference

- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L356-L374

```
356:    function verifyV2048(
357:        uint128 domain_size,
358:        uint256[] memory vkdata,
359:        uint256[] memory public_inputs,
360:        uint256[] memory serialized_proof
361:    ) public view returns (bool) {
362:        bytes32 vkhash = sha256(
363:            abi.encodePacked(uint64(public_inputs.length), domain_size, vkdata)
364:        );
365:        require(vk2048hash == vkhash, "E: wrong vkey");
366:        return
367:            verifyProof(
368:                vkhash,
369:                domain_size,
370:                vkdata,
371:                public_inputs,
372:                serialized_proof
373:            );
374:    }
```

Description

newway55 : If the verification key hash is stored in the first element of the vkdata array as a uint256, then you can use the following :

- attacker is able to modify the value of vk2048hash to a value that does not match the hash of the verification key, and the original code require(vk2048hash == vkhash, "E: wrong vkey") is used, the attacker could bypass the verification check and execute some malicious code.

Consider below POC contract

```
function testVerifyV2048() public {
       function testVerifyV2048() public {
       uint128 domain_size = 123456789;
       uint256[] memory vkdata = new uint256[](2);
       vkdata[0] = uint256(keccak256("verification key"));
       vkdata[1] = uint256(keccak256("some other data"));
       uint256[] memory public_inputs = new uint256[](1);
       public_inputs[0] = 1234567890;
       uint256[] memory serialized_proof = new uint256[](4);
       serialized_proof[0] = 1;
       serialized_proof[1] = 2;
       serialized_proof[2] = 3;
       serialized_proof[3] = 4;

       // Test with correct input
       bool result = verifier.verifyV2048(domain_size, vkdata, public_inputs, serialized_proof);
       assertEq(result, true, "Verification failed");

       // Test with incorrect input
       vkdata[0] = uint256(keccak256("incorrect verification key"));
       result = verifier.verifyV2048(domain_size, vkdata, public_inputs, serialized_proof);
       assertEq(result, false, "Verification should fail with incorrect verification key");
   }
```

Recommendation

newway55 : Fix the check : `require(bytes32(vkdata[0]) == vkhash, "E: wrong vkey");`

Consider below fix in the `verifyV2048()` function

```
function verifyV2048(
    uint128 domain_size,
    uint256[] memory vkdata,
    uint256[] memory public_inputs,
    uint256[] memory serialized_proof
) public view returns (bool) {
    bytes32 vkhash = sha256(
        abi.encodePacked(uint64(public_inputs.length), domain_size, vkdata)
    );
    require(bytes32(vkdata[0]) == vkhash, "E: wrong vkey");
    return verifyProof(vkhash, domain_size, vkdata, public_inputs, serialized_proof);
}
```

Client Response

Declined,Can't understand the meaning of the suggestion, and vk2048hash can only be changed by the administrator.

UPV-17:Missing Error Messages and Event Emissions in `new_g1_checked` Function

Category	Severity	Status	Contributor
Code Style	Informational	Fixed	BradMoonUESTC

Code Reference

- code/UniPass-verifier-contract/contracts/PlonkCoreLib.sol#L102-L123

```
102:     /// 2. affine points: `x` and `y` in field Fq, and satisfy the curve equation $y^2 = x^3 + b
$
103:     function new_g1_checked(uint256 x, uint256 y)
104:         internal
105:         pure
106:         returns (G1Point memory)
107:     {
108:         if (x == 0 && y == 0) {
109:             // point of infinity is (0,0)
110:             return G1Point(x, y);
111:         }
112:
113:         // check encoding
114:         require(x < q_mod);
115:         require(y < q_mod);
116:         // check on curve
117:         uint256 lhs = mulmod(y, y, q_mod); // y^2
118:         uint256 rhs = mulmod(x, x, q_mod); // x^2
119:         rhs = mulmod(rhs, x, q_mod); // x^3
120:         rhs = addmod(rhs, bn254_b_coeff, q_mod); // x^3 + b
121:         require(lhs == rhs);
122:
123:         return G1Point(x, y);
```

Description

BradMoonUESTC : The first contract is missing both error messages and event emissions in the new_g1_checked function. In this function, the contract should emit events to inform external listeners about the occurrences of certain

events, such as the creation of new G1 points. Moreover, the function should provide descriptive error messages for require statements to facilitate better debugging and provide more context in case of an error.

Recommendation

BradMoonUESTC : For example, the first contract's new_g1_checked function could emit an event like NewG1PointCreated(uint256 x, uint256 y) to notify external listeners that a new G1 point has been created with the specified coordinates.

Additionally, the require statements should include error messages, such as:

```
require(x < q_mod, "x coordinate is not within the valid range");
require(y < q_mod, "y coordinate is not within the valid range");
require(lhs == rhs, "point is not on the curve");
```

Add appropriate event emissions and error messages to the require statements in the new_g1_checked function in the first contract to ensure transparency, facilitate debugging, and provide more context in case of an error.

Client Response

Fixed

UPV-18:Redundant variables

Category	Severity	Status	Contributor
Code Style	Informational	Declined	lfzkoala

Code Reference

code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L197

```
197:        uint256 poly_nums_len = poly_nums.length;
```

Description

lfzkoala : There are many places that assigning `dens.length` to a newly created variable `dens_len` and use `dens_len` in the for loop, but it's not necessary to create new variables, you can immediately use `dens.length` in the for loop. This also works for `poly_nums.length` or `nums.length`.

For example

```
uint256 poly_nums_len = poly_nums.length;
        for (uint256 i = 0; i < poly_nums_len; i++) {
            //  = omega.pow(poly_nums[i]); // power of omega
            nums[i].assign(vanishing_at_z);
            nums[i].mul_assign(tmp_1);

            dens[i].assign(at); // (X - omega^i) * N
            dens[i].sub_assign(tmp_1);
            dens[i].mul_assign(tmp_2); // mul by domain size

            tmp_1.mul_assign(omega);
        }
```

Recommendation

lfzkoala : Immediately use `xxx.length` in the for loop instead of creating new variables before the loop.

Client Response

Declined,Create a new variable for dens_len to save a little gas.

UPV-19: `++i` costs less gas than `i++`, especially when it's used in `for`-loops (`--i`/`i--` too)

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	thereksfour

Code Reference

- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L99
- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L102
- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L111
- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L119
- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L123
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L198
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L222
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L243
- code/UniPass-verifier-contract/contracts/UnipassVerifier.sol#L276
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L302
- code/UniPass-verifier-contract/contracts/PlonkCoreLib.sol#L308
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L320
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L438
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L509
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L554
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L698
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L734
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L741
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L764
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L802
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L809
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L919
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L924
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L943
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L958
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L981
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L984

```
99:         for (uint256 i = 0; i < b_len / 8; i++) {
102:         for (uint256 i = 0; i < b_len % 8; i++) {
111:         for (uint256 i = 0; i < tmp_index; i++) {
119:         for (uint256 i = 0; i < tmp_index / 8; i++) {
123:         for (uint256 i = 0; i < tmp_index % 8; i++) {
198:         for (uint256 i = 0; i < poly_nums_len; i++) {
222:         for (uint256 i = 1; i < dens_len; i++) {
243:         for (uint256 i = 0; i < nums_len; i++) {
276:         for (uint256 i = 0; i < 3; i++) {
302:         for (uint256 i = 0; i < inputs_len; i++) {
308:         for (uint256 i = 0; i < elements; i++) {
320:         for (uint256 i = 0; i < STATE_WIDTH - 1; i++) {
438:         for (uint256 i = 1; i < STATE_WIDTH; i++) {
509:         for (uint256 i = 1; i < STATE_WIDTH - 1; i++) {
554:         for (uint256 i = 1; i < STATE_WIDTH; i++) {
698:         for (uint256 i = 1; i < STATE_WIDTH; i++) {
734:         for (uint256 i = 3; i < STATE_WIDTH; i++) {
741:         for (uint256 i = 0; i < STATE_WIDTH - 1; i++) {
764:         for (uint256 i = 0; i < STATE_WIDTH; i++) {
802:         for (uint256 i = 0; i < STATE_WIDTH; i++) {
809:         for (uint256 i = 0; i < STATE_WIDTH - 1; i++) {
```

```
919:         for (uint256 i = 0; i < vk.num_inputs; i++) {

924:         for (uint256 i = 0; i < STATE_WIDTH; i++) {

943:         for (uint256 i = 0; i < STATE_WIDTH; i++) {

958:         for (uint256 i = 0; i < lagrange_poly_len; i++) {

981:         for (uint256 i = 0; i < STATE_WIDTH; i++) {

984:         for (uint256 i = 0; i < STATE_WIDTH - 1; i++) {
```

Description

thereksfour : `++i` costs less gas than `i++`, especially when it's used in `for`-loops (`--i`/`i--` too) *Saves 5 gas per loop*

Instances (27):

```
File: UniPass-verifier-contract/contracts/PlonkCoreLib.sol

308:          for (uint256 i = 0; i < elements; i++) {
```

File: UniPass-verifier-contract/contracts/PlookupSingleCore.sol

```
198:         for (uint256 i = 0; i < poly_nums_len; i++) {

222:         for (uint256 i = 1; i < dens_len; i++) {

243:         for (uint256 i = 0; i < nums_len; i++) {

302:         for (uint256 i = 0; i < inputs_len; i++) {

320:         for (uint256 i = 0; i < STATE_WIDTH - 1; i++) {

438:         for (uint256 i = 1; i < STATE_WIDTH; i++) {

509:         for (uint256 i = 1; i < STATE_WIDTH - 1; i++) {

554:         for (uint256 i = 1; i < STATE_WIDTH; i++) {

698:         for (uint256 i = 1; i < STATE_WIDTH; i++) {

734:         for (uint256 i = 3; i < STATE_WIDTH; i++) {

741:         for (uint256 i = 0; i < STATE_WIDTH - 1; i++) {

764:         for (uint256 i = 0; i < STATE_WIDTH; i++) {

802:         for (uint256 i = 0; i < STATE_WIDTH; i++) {

809:         for (uint256 i = 0; i < STATE_WIDTH - 1; i++) {

919:         for (uint256 i = 0; i < vk.num_inputs; i++) {

924:         for (uint256 i = 0; i < STATE_WIDTH; i++) {

943:         for (uint256 i = 0; i < STATE_WIDTH; i++) {

958:         for (uint256 i = 0; i < lagrange_poly_len; i++) {

981:         for (uint256 i = 0; i < STATE_WIDTH; i++) {

984:         for (uint256 i = 0; i < STATE_WIDTH - 1; i++) {
```

```
File: UniPass-verifier-contract/contracts/UnipassVerifier.sol

99:         for (uint256 i = 0; i < b_len / 8; i++) {

102:         for (uint256 i = 0; i < b_len % 8; i++) {

111:         for (uint256 i = 0; i < tmp_index; i++) {

119:         for (uint256 i = 0; i < tmp_index / 8; i++) {

123:         for (uint256 i = 0; i < tmp_index % 8; i++) {

276:         for (uint256 i = 0; i < 3; i++) {
```

Recommendation

thereksfour : use `++i` instead of `i++`

Client Response

Fixed

UPV-20: `i < dens_len` condition is useless in `PlookupSingleCore` contract `batch_evaluate_lagrange_poly_out_of_domain` function

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	alansh

Code Reference

code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L233-L239

```
233:         for (uint256 i = dens_len - 1; i < dens_len; i--) {
234:             tmp3.assign(dens[i]);
235:             dens[i].assign(tmp_2); // all inversed
236:             dens[i].mul_assign(partial_products[i]); // clear lowest terms
237:             tmp_2.mul_assign(tmp3);
238:             if (i == 0) break;
239:         }
```

Description

alansh : This condition is always satisfied.

Recommendation

alansh : Remove this condition to save some gas, or replace this condition with `i>=0` and remove `if (i == 0) break;` in loop body.

Client Response

Fixed

UPV-21:poly_nums values not used in `PlookupSingleCore` contract `batch_evaluate_lagrange_poly_out_of_domain` function

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	alansh

Code Reference

- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L163

```
163:        uint256[] memory poly_nums, //just [0,n)
```

Description

alansh : This function only make use of `poly_nums.length` but not the actual elements.

Recommendation

alansh : Directly pass `poly_nums_len` instead, this will also save some gas.

Client Response

Fixed

UPV-22:replace subsequent `PairingsBn254.new_fr(1)` with `PairingsBn254.copy(one)` in `PlookupSingleCore` contract `batch_evaluate_lagrange_poly_out_of_domain` function

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	alansh

Code Reference

- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L177
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L221

```
177:        PairingsBn254.Fr memory tmp_1 = PairingsBn254.new_fr(1);

221:        partial_products[0].assign(PairingsBn254.new_fr(1));
```

Description

alansh : Using `PairingsBn254.copy(one)` will save a bit gas since it does has the `require` check inside.

Recommendation

alansh : Replace `PairingsBn254.new_fr(1)` with `PairingsBn254.copy(one)`, positions are recorded below.

Client Response

Fixed

UPV-23:tmp3 doesn't need to be initialized in `PlookupSingleCore` contract `batch_evaluate_lagrange_poly_out_of_domain` function

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	alansh

Code Reference

- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L232
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L300
- code/UniPass-verifier-contract/contracts/PlookupSingleCore.sol#L426

```
232:        PairingsBn254.Fr memory tmp3 = PairingsBn254.new_fr(0);

300:        PairingsBn254.Fr memory tmp = PairingsBn254.new_fr(0);

426:        PairingsBn254.Fr memory tmp_fr = PairingsBn254.new_fr(0);
```

Description

alansh : This statement is useless:

```
PairingsBn254.Fr memory tmp3 = PairingsBn254.new_fr(0);
```

Since `tmp3` is always `assigned` in the loop. The same for L300/L426.

Recommendation

alansh : Remove the assignment outside of the loop, just declaration is enough. The same for L300.

Client Response

Fixed

Disclaimer

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 Invoices, or the scope of services, and terms and conditions provided to you (“Customer” or the “Company”) in connection with the Invoice. This report provided in connection with the services set forth in the Invoices shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Invoice. 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 Secure3’s prior written consent in each instance.

This report is not an “endorsement” or “disapproval” of any particular project or team. This report is not an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Secure3 to perform a security assessment. This report does not provide any warranty or guarantee of free of bug of codes analyzed, nor do they provide any indication of the technologies, business model or legal compliancy.

This report should not be used in any way to make decisions around investment or involvement with any particular project. Instead, it represents an extensive assessing process intending to help our customers increase the quality of their code and high-level consistency of implementation and business model, while reducing the risk presented by cryptographic tokens and blockchain technology.

Secure3’s position on the final decisions over blockchain technologies and corresponding associated transactions is that each company and individual are responsible for their own due diligence and continuous security.

The assessment services provided by Secure3 is subject to dependencies and under continuing development. 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.