

CERTIDAPP SMART CONTRACT AUDIT REPORT



CertiDapp Smart Contract Audit

The audit made by Aveesh Shetty

Overview

Certidapp is a certificate on the blockchain. Certidapp is composed of organizations that can sign the certificate and user who can register the certificate.

Disclaimer

The audit makes no statement or warranties about the utility of the code safety of the code, suitability of the business model regulatory regime for the business model, or any other statement about the fitness of the contracts to purpose, or their bug-free status. The audit documentation is for discussion purposes only.



Introduction :

This Audit Report highlights the overall security of the CertiDapp Smart Contract. With this report, we have tried to ensure the reliability of their smart contract by complete the assessment of their system's architecture and the smart contract code-base.

Auditing Approach and Methodologies applied :

I performed thorough testing of the project starting with analyzing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third party smart contracts and libraries.

Our team then performed a formal line by line inspection of the Smart Contract in order to find any potential issues like race conditions, transaction-ordering dependence, timestamp dependence, and denial of service attacks.

In the Unit testing Phase, we coded/conducted Custom unit tests written for each function in the contract to verify that each function works as expected. In Automated Testing, We tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was tested in collaboration of our multiple team members and this included -

1. Testing the functionality of the Smart Contract to determine proper logic has been followed throughout.
2. Analyzing the complexity of the code by thorough, manual review of the code, line-by-line.
3. Deploying the code on testnet using multiple clients to run live tests
4. Analyzing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
5. Checking whether all the libraries used in the code are on the latest version.
6. Analyzing the security of the on-chain data.

Certificate of Attendance


is awarded to

Saurav bhardwaj

For Acheiving **100%** Attendance in STTP on Blockchain from **16/01/2020** to **17/01/2020** at **Bharti Vidyapeeth Kharghar**.

The above certificate information is signed by following 1 signer which is cryptographically verified by the certificate smart contract.



Signer 1: Blocklogy Edutech 
(0x8e98...811A)

Signature: 0x8a03539c...700ce2aa1b

Certificate Hash: 0x5a189bf150601ca2afb6e06dcddc8c0886f360fc00eb401eee2f94241fbf5f87

Created at transaction
0xb8a0...b03b. [View on EtherScan](#)



Audit Details

- Project Name: CERTIDAPP
- website/Etherscan Code:
<https://kmpards.github.io/certidapp/>

<https://kovan.etherscan.io/address/ox3ea996e3A2f2A8b235065a7Fa5d55e5f626f0003#code>

- Languages: Solidity(Smart contract), Javascript(Unit Testing)

Summary of CertiDapp Smart

Contract :

I conducted a security audit of a smart contract of Certidapp. CertiDapp tecontract accepts certificates signed by multiple authorized signers

Smart contract contains basic functionalities of certificates signed by multiple authorized signers and some advance functionalities.

- present manager to change the manager wallet address
- manager to for to update KYC / verification status of Certifying Authorities
- Certifying Authorities to change their wallet (in case of theft).
- Submit a signed certificate to smart contract for adding it to storage.
- check whether the certificate is freshly being registered
- Contract to seperate signers from certificate data.

-Change the manager

-Check whether an address exists in packed addresses bytes

Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient and working according to its specifications. The audit activities can be grouped in the following three categories:

Security: Identifying security related issues within each contract and within the system of contracts.

Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.

Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include:

- Correctness
- Readability
- Sections of code with high complexity
- Quantity and quality of test coverage

Security Level references :

Every issue in this report was assigned a severity level from the following:

High severity issues will bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

High severity issues:-

Pure-functions should not read/change state

File: CertiDApp.sol

Lines: 179–197

Severity: 1

```
for(uint256 i = 0; i < _packedSigners.length; i += 20) {
    assembly {
        _tempSigner := mload(add(_packedSigners, add(0x14, i)))
    }
    if(_tempSigner == _signer) return false;
}

return true;
}
```

```

    /// @notice Used to get a number's utf8 representation
    /// @param i Integer
    /// @return utf8 representation of i
    function _getBytesStr(uint i) private pure returns (bytes
memory) {
        if (i == 0) {
            return "0";
        }
        uint j = i;
        uint len;
        while (j != 0) {
            len++;
            j /= 10;
        }
        bytes memory bstr = new bytes(len);
        uint k = len - 1;
        while (i != 0) {
            bstr[k--] = byte(uint8(48 + i % 10));
            i /= 10;
        }
        return bstr;
    }
}

```

File: RLP.sol

Lines: 156–174

Severity: 1

```

assembly {
    result := mload(memPtr)

    // shfit to the correct location if neccesary
    if lt(len, 32) {

```

File: RLP.sol

Lines: 107–119

Severity: 1

```
function isList(RLPItem memory item) internal pure returns
(bool) {
    if(item.len == 0) return false;

    uint8 byte0;
    uint memPtr = item.memPtr;
    assembly {
        byte0 := byte(0, mload(memPtr))
    }

    if(byte0 < LIST_SHORT_START) return false;

    return true;
}
```

File: RLP.sol

Lines: 268–282

Severity: 1

```
function _payloadOffset(uint memPtr) private pure returns (uint)
{
    uint byte0;
    assembly {
        byte0 := byte(0, mload(memPtr))
    }
}
```

File: RLP.sol

Lines: 51–58

Severity: 1

```
function toRlpItem(bytes memory item) internal pure returns
(RLPItem memory) {
    uint memPtr;
    assembly {
        memPtr := add(item, 0x20)
    }
}
```

```
    return RLPItem(item.length, memPtr);
}
```

File: RLP.sol

Lines: 177–188

Severity: 1

```
function toUIntStrict(RLPItem memory item) internal pure returns
(uint) {
    // one byte prefix
    require(item.len == 33);

    uint result;
    uint memPtr = item.memPtr + 1;
    assembly {
        result := mload(memPtr)
    }

    return result;
}
```

File: RLP.sol

Lines: 138–147

Severity: 1

```
function toBoolean(RLPItem memory item) internal pure returns
(bool) {
    require(item.len == 1);
    uint result;
    uint memPtr = item.memPtr;
    assembly {
        result := byte(0, mload(memPtr))
    }

    return result == 0 ? false : true;
}
```

File: RLP.sol

Lines: 190–204

Severity: 1

```
uint destPtr;
    assembly {
        destPtr := add(0x20, result)
    }
```

File: RLP.sol

Lines: 289–309

Severity: 1

```
// copy as many word sizes as possible
    for (; len >= WORD_SIZE; len -= WORD_SIZE) {
        assembly {
            mstore(dest, mload(src))
        }
    }
```

File: RLP.sol

Lines: 226–265

Severity: 1

```
function _itemLength(uint memPtr) private pure returns (uint) {
    uint itemLen;
    uint byte0;
    assembly {
        byte0 := byte(0, mload(memPtr))
    }
}
```

File: RLP.sol

Lines: 124–135

Severity: 1

```
function toRlpBytes(RLPItem memory item) internal pure returns
(bytes memory) {
    bytes memory result = new bytes(item.len);
    if (result.length == 0) return result;

    uint ptr;
    assembly {
        ptr := add(0x20, result)
    }

    copy(item.memPtr, ptr, item.len);
    return result;
}
```

View-function should not change the state

File: CertiDApp.sol

Lines: 119–166

```
function parseSignedCertificate(
    bytes memory _signedCertificate,
    bool _allowedSignersOnly
) public view returns (
    Certificate memory _certificateObj,
    bytes32 _certificateHash
) {
    RLP.RLPItem[] memory _certificateRLP =
_signedCertificate.toRlpItem().toList();

    _certificateObj.data = _certificateRLP[0].toRlpBytes();

    _certificateHash = keccak256(abi.encodePacked(
        PERSONAL_PREFIX,
        _getBytesStr(_certificateObj.data.length),
        _certificateObj.data
    ));

    /// @notice loop through every signature and use elliptic
```

```

curves cryptography to recover the
    /// address of the wallet used for signing the
certificate.
    for(uint256 i = 1; i < _certificateRLP.length; i += 1) {
        bytes memory _signature = _certificateRLP[i].toBytes();

        bytes32 _r;
        bytes32 _s;
        uint8 _v;

        assembly {
            let _pointer := add(_signature, 0x20)
            _r := mload(_pointer)
            _s := mload(add(_pointer, 0x20))
            _v := byte(0, mload(add(_pointer, 0x40)))
            if lt(_v, 27) { _v := add(_v, 27) }
        }

        require(_v == 27 || _v == 28, 'invalid recovery value');

        address _signer = ecrecover(_certificateHash, _v, _r, _s);

        require(_checkUniqueSigner(_signer,
_certificateObj.signers), 'each signer should be unique');

        if(_allowedSignersOnly) {
            require(certifyingAuthorities[_signer].status ==
AuthorityStatus.Authorised, 'certifier not authorised');
        }

        /// @dev packing every signer address into a single bytes
value
        _certificateObj.signers =
abi.encodePacked(_certificateObj.signers, _signer);
    }
}

```

Do not declare functions that change the state as a view.

The following statements are considered modifying the state:

Writing to state variables;

Emitting events;

Creating other contracts;

Using self-destruct;

Sending Ether via calls;

Calling any function not marked view or pure;

Using low-level calls;

Using inline assembly that contains certain opcodes.

5.)Prefer external to public visibility level

Prefer external to public visibility level

A function with public visibility modifier that is not called internally. Changing the visibility level to external increases code readability. Moreover, in many cases, functions with external visibility modifiers spend less gas compared to functions with public visibility modifiers.

File: CertiDApp.sol

Lines: 69–112

Severity: 1

```
function registerCertificate(  
    bytes memory _signedCertificate
```

```

) public returns (
    bytes32
) {
    (Certificate memory _certificateObj, bytes32
_certificateHash) = parseSignedCertificate(_signedCertificate,
true);

    /// @notice Signers in this transaction
    bytes memory _newSigners = _certificateObj.signers;

    /// @notice If certificate already registered then signers
    can be updated.
    /// Initializing _updatedSigners with existing signers on
    blockchain if any.
    /// More signers would be appended to this in next 'for'
    loop.
    bytes memory _updatedSigners =
certificates[_certificateHash].signers;

    /// @notice Check with every the new signer if it is not
    already included in storage.
    /// This is helpful when a same certificate is submitted
    again with more signatures,
    /// the contract will consider only new signers in that
    case.
    for(uint256 i = 0; i < _newSigners.length; i += 20) {
        address _signer;
        assembly {
            _signer := mload(add(_newSigners, add(0x14, i)))
        }
        if(!_checkUniqueSigner(_signer,
certificates[_certificateHash].signers)) {
            _updatedSigners = abi.encodePacked(_updatedSigners,
_signer);
            emit Certified(
                _certificateHash,
                _signer
            );
        }
    }

    /// @notice check whether the certificate is freshly being
    registered.
    /// For new certificates, directly proceed with adding it.
    /// For existing certificates only update the signers if

```

```
there are any new.
    if(certificates[_certificateHash].signers.length > 0) {
        require(_updatedSigners.length >
certificates[_certificateHash].signers.length, 'need new
signers');
        certificates[_certificateHash].signers = _updatedSigners;
    } else {
        certificates[_certificateHash] = _certificateObj;
    }

    return _certificateHash;
}
```

File: CertiDApp.sol

Lines: 46–64

Severity: 1

```
function migrateCertifyingAuthority(
    address _newAuthorityAddress
) public onlyAuthorisedCertifier {
    require(
        certifyingAuthorities[_newAuthorityAddress].status ==
AuthorityStatus.NotAuthorised
        , 'cannot migrate to an already authorised address'
    );

    certifyingAuthorities[msg.sender].status =
AuthorityStatus.Migrated;
    emit AuthorityStatusUpdated(msg.sender,
AuthorityStatus.Migrated);

    certifyingAuthorities[_newAuthorityAddress] =
CertifyingAuthority({
        data: certifyingAuthorities[msg.sender].data,
        status: AuthorityStatus.Authorised
    });
    emit AuthorityStatusUpdated(_newAuthorityAddress,
AuthorityStatus.Authorised);

    emit AuthorityMigrated(msg.sender, _newAuthorityAddress);
}
```


File: CertiDApp.sol

Lines: 28–40

Severity: 1

```
function updateCertifyingAuthority(
    address _authorityAddress,
    bytes memory _data,
    AuthorityStatus _status
) public onlyManager {
    if(_data.length > 0) {
        certifyingAuthorities[_authorityAddress].data = _data;
    }

    certifyingAuthorities[_authorityAddress].status = _status;

    emit AuthorityStatusUpdated(_authorityAddress, _status);
}
```

File: CertiDApp.sol

Lines: 21–23

Severity: 1

```
function changeManager(address _newManagerAddress) public
onlyManager {
    _changeManager(_newManagerAddress);
}
```

Implicit visibility level

The default function visibility level in contracts is public, in interfaces — external, state variable default visibility level is internal. In contracts, the fallback function can be external or public. In interfaces, all the functions should be declared as

external. Explicitly define function visibility to prevent confusion.

Avoid ambiguity: explicitly declare visibility levels.

File: Proxy.sol

Lines: 52–53

Severity: 1

```
for(uint256 i = 0; i < _newSigners.length; i += 20) {
    address _signer;
    assembly {
        _signer := mload(add(_newSigners, add(0x14, i)))
    }
    if(!_checkUniqueSigner(_signer,
certificates[_certificateHash].signers)) {
        _updatedSigners = abi.encodePacked(_updatedSigners,
_signer);
        emit Certified(
            _certificateHash,
            _signer
        );
    }
}
```

File: Proxy.sol

Lines: 47–47

Severity: 1

```
for(uint256 i = 0; i < _packedSigners.length; i += 20) {
    assembly {
        _tempSigner := mload(add(_packedSigners, add(0x14, i)))
    }
    if(_tempSigner == _signer) return false;
}
```

File: Proxy.sol

Lines: 53–54

Severity: 1

```
assembly {
    let _pointer := add(_signature, 0x20)
    _r := mload(_pointer)
    _s := mload(add(_pointer, 0x20))
    _v := byte(0, mload(add(_pointer, 0x40)))
    if lt(_v, 27) { _v := add(_v, 27) }
}
```

File: Proxy.sol

Lines: 54–57

Severity: 1

```
assembly {
    result := mload(memPtr)

    // shfit to the correct location if neccesary
    if lt(len, 32) {
        result := div(result, exp(256, sub(32, len)))
    }
}
```

File: Proxy.sol

Lines: 53–53

Severity: 1

```
function toBytes(RLPItem memory item) internal pure returns
(bytes memory) {
    require(item.len > 0);

    uint offset = _payloadOffset(item.memPtr);
```

```
uint len = item.len - offset; // data length
bytes memory result = new bytes(len);

uint destPtr;
assembly {
    destPtr := add(0x20, result)
}

copy(item.memPtr + offset, destPtr, len);
return result;
}
```

File: Proxy.sol

Lines: 52–52

Severity: 1

```
assembly {
    byte0 := byte(0, mload(memPtr))
}
```

File: Proxy.sol

Lines: 50–51

Severity: 1

```
assembly {
    let srcpart := and(mload(src), not(mask)) // zero out src
    let destpart := and(mload(dest), mask) // retrieve the
bytes
    mstore(dest, or(destpart, srcpart))
}
```

File: Proxy.sol

Lines: 57–57

Severity: 1

```
require(_impl != address(0));

assembly {
    let ptr := mload(0x40)
    calldatacopy(ptr, 0, calldatasize())
    let result := delegatecall(gas(), _impl, ptr,
calldatasize(), 0, 0)
    let size := returndatasize()
    returndatacopy(ptr, 0, size)

    switch result
    case 0 { revert(ptr, size) }
```

File: Proxy.sol

Lines: 57–57

Severity: 1

```
case 0 { revert(ptr, size) }
```

File: Proxy.sol

Lines: 51–52

Severity: 1

```
require(_impl != address(0));

assembly {
    let ptr := mload(0x40)
    calldatacopy(ptr, 0, calldatasize())
    let result := delegatecall(gas(), _impl, ptr,
calldatasize(), 0, 0)
    let size := returndatasize()
    returndatacopy(ptr, 0, size)

    switch result
    case 0 { revert(ptr, size) }
```

File: Proxy.sol

Lines: 50–50

Severity: 1

```
require(_impl != address(0));

    assembly {
        let ptr := mload(0x40)
        calldatacopy(ptr, 0, calldatasize())
        let result := delegatecall(gas(), _impl, ptr,
calldatasize(), 0, 0)
        let size := returndatasize()
        returndatacopy(ptr, 0, size)

        switch result
        case 0 { revert(ptr, size) }
```

File: Proxy.sol

Lines: 52–52

Severity: 1

```
assembly {
    let ptr := mload(0x40)
    calldatacopy(ptr, 0, calldatasize())
    let result := delegatecall(gas(), _impl, ptr,
calldatasize(), 0, 0)
    let size := returndatasize()
    returndatacopy(ptr, 0, size)

    switch result
    case 0 { revert(ptr, size) }
```

File: Proxy.sol

Lines: 51–51

Severity: 1

```
require(_impl != address(0));

assembly {
    let ptr := mload(0x40)
    calldatacopy(ptr, 0, calldatasize())
    let result := delegatecall(gas(), _impl, ptr,
calldatasize(), 0, 0)
    let size := returndatasize()
    returndatacopy(ptr, 0, size)

    switch result
    case 0 { revert(ptr, size) }
```

File: Proxy.sol

Lines: 47–50

Severity: 1

```
fallback () external {
    address _impl = implementation;
    require(_impl != address(0));

    assembly {
        let ptr := mload(0x40)
        calldatacopy(ptr, 0, calldatasize())
        let result := delegatecall(gas(), _impl, ptr,
calldatasize(), 0, 0)
        let size := returndatasize()
        returndatacopy(ptr, 0, size)

        switch result
        case 0 { revert(ptr, size) }
```

File: Proxy.sol

Lines: 46–46

Severity: 1

```
require(_impl != address(0));

    assembly {
        let ptr := mload(0x40)
        calldatacopy(ptr, 0, calldatasize())
        let result := delegatecall(gas(), _impl, ptr,
calldatasize(), 0, 0)
        let size := returndatasize()
        returndatacopy(ptr, 0, size)

        switch result
        case 0 { revert(ptr, size) }
```

File: Proxy.sol

Lines: 47-47

Severity: 1

```
require(_impl != address(0));

    assembly {
        let ptr := mload(0x40)
        calldatacopy(ptr, 0, calldatasize())
        let result := delegatecall(gas(), _impl, ptr,
calldatasize(), 0, 0)
        let size := returndatasize()
        returndatacopy(ptr, 0, size)

        switch result
        case 0 { revert(ptr, size) }
```

File: Proxy.sol

Lines: 52-52

Severity: 1

```
let ptr := mload(0x40)
    calldatacopy(ptr, 0, calldatasize())
    let result := delegatecall(gas(), _impl, ptr,
```



```
calldatasize(), 0, 0)
    let size := returndatasize()
    returndatacopy(ptr, 0, size)

    switch result
    case 0 { revert(ptr, size) }
```

File: Proxy.sol

Lines: 45–45

Severity: 1

```
require(_impl != address(0));

    assembly {
        let ptr := mload(0x40)
        calldatacopy(ptr, 0, calldatasize())
        let result := delegatecall(gas(), _impl, ptr,
calldatasize(), 0, 0)
        let size := returndatasize()
        returndatacopy(ptr, 0, size)

        switch result
        case 0 { revert(ptr, size) }
```

File: StorageStructure.sol

Lines: 20–20

Severity: 1

```
mapping(bytes32 => bytes32) extraData;
```

File: RLP.sol

Lines: 9–9

Severity: 1

```
uint8 constant STRING_SHORT_START = 0x80;
  uint8 constant STRING_LONG_START = 0xb8;
  uint8 constant LIST_SHORT_START = 0xc0;
  uint8 constant LIST_LONG_START = 0xf8;
  uint8 constant WORD_SIZE = 32;
```

File: RLP.sol

Lines: 10–10

Severity: 1

```
uint8 constant STRING_SHORT_START = 0x80;
  uint8 constant STRING_LONG_START = 0xb8;
  uint8 constant LIST_SHORT_START = 0xc0;
  uint8 constant LIST_LONG_START = 0xf8;
  uint8 constant WORD_SIZE = 32;
```

File: RLP.sol

Lines: 8–8

Severity: 1

```
pragma solidity ^0.6.2;
```

```
library RLP {
  uint8 constant STRING_SHORT_START = 0x80;
  uint8 constant STRING_LONG_START = 0xb8;
  uint8 constant LIST_SHORT_START = 0xc0;
  uint8 constant LIST_LONG_START = 0xf8;
  uint8 constant WORD_SIZE = 32;
```

File: RLP.sol

Lines: 7–7

Severity: 1

```
pragma solidity ^0.6.2;
```

```
library RLP {  
    uint8 constant STRING_SHORT_START = 0x80;  
    uint8 constant STRING_LONG_START  = 0xb8;  
    uint8 constant LIST_SHORT_START   = 0xc0;  
    uint8 constant LIST_LONG_START    = 0xf8;  
    uint8 constant WORD_SIZE = 32;  
}
```

File: RLP.sol

Lines: 11–11

Severity: 1

```
pragma solidity ^0.6.2;
```

```
library RLP {  
    uint8 constant STRING_SHORT_START = 0x80;  
    uint8 constant STRING_LONG_START  = 0xb8;  
    uint8 constant LIST_SHORT_START   = 0xc0;  
    uint8 constant LIST_LONG_START    = 0xf8;  
    uint8 constant WORD_SIZE = 32;  
}
```

Medium Severity Issues:–

File: CertiDApp.sol

Lines: 87–99

Severity: 1

```
for(uint256 i = 0; i < _newSigners.length; i += 20) {  
    address _signer;  
    assembly {  
        _signer := mload(add(_newSigners, add(0x14, i)))  
    }  
    if(_checkUniqueSigner(_signer,
```

```

certificates[_certificateHash].signers)) {
    _updatedSigners = abi.encodePacked(_updatedSigners,
    _signer);
    emit Certified(
        _certificateHash,
        _signer
    );
}
}

```

File: CertiDApp.sol

Lines: 189–194

```

for(uint256 i = 0; i < _packedSigners.length; i += 20) {
    assembly {
        _tempSigner := mload(add(_packedSigners, add(0x14, i)))
    }
    if(_tempSigner == _signer) return false;
}for(uint256 i = 0; i < _packedSigners.length; i += 20) {
    assembly {
        _tempSigner := mload(add(_packedSigners, add(0x14, i)))
    }
    if(_tempSigner == _signer) return false;
}

```

Lines: 217–217

Severity: 1

Costly loop

```

while (currPtr < endPtr) {

function _itemLength(uint memPtr) private pure returns (uint) {
    uint itemLen;
    uint byte0;
    assembly {
        byte0 := byte(0, mload(memPtr))
    }
}

```

Ethereum is a very resource-constrained environment. Prices per computational step are orders of magnitude higher than with centralized providers. Moreover, Ethereum miners impose a limit on the total number of gas consumed in a block. If `array.length` is large enough, the function exceeds the block gas limit, and transactions calling it will never be confirmed:

```
for (uint256 i = 0; i < array.length ; i++) {  
  
    costlyFunc();  
  
}
```

This becomes a security issue if an external actor influences `array.length`. E.g., if an array enumerates all registered addresses, an adversary can register many addresses, causing the problem described above.

Vulnerability type by SmartDec classification: Infinite loops.

Do not declare functions that read from or modify the state as pure.

The following statements are considered modifying the state:

Writing to state variables;

Emitting events;

Creating other contracts;

Using self-destruct;

Sending Ether via calls;

Calling any function not marked view or pure;

Using low-level calls;

Using inline assembly that contains certain opcodes.

The following statements are considered reading from the state:

Reading from state variables;

Accessing `this.balance` or `<address>.balance`;

Accessing any of the members of the block, tx, msg (with the exception of `msg.sig` and `msg.data`);

Calling any function not marked pure;

Using inline assembly that contains certain opcodes

Low Severity Issues:–

Use of assembly

Inline assembly is a way to access the Ethereum Virtual Machine at a low level. This discards several important safety features of Solidity.

Avoid using assembly if possible.

File: CertiDApp.sol

Lines: 89–91

Severity: 1

File: CertiDApp.sol

```
for(uint256 i = 0; i < _newSigners.length; i += 20) {
    address _signer;
    assembly {
        _signer := mload(add(_newSigners, add(0x14, i)))
    }
    if(!_checkUniqueSigner(_signer,
certificates[_certificateHash].signers)) {
        _updatedSigners = abi.encodePacked(_updatedSigners,
_signer);
        emit Certified(
            _certificateHash,
            _signer
        );
    }
}
```

Lines: 190–192

Severity: 1

File: CertiDApp.sol

```
for(uint256 i = 0; i < _packedSigners.length; i += 20) {
    assembly {
        _tempSigner := mload(add(_packedSigners, add(0x14, i)))
    }
    if(_tempSigner == _signer) return false;
}
```

Lines: 145–151

Severity: 1

File: RLP.sol

```
assembly {
    let _pointer := add(_signature, 0x20)
    _r := mload(_pointer)
    _s := mload(add(_pointer, 0x20))
    _v := byte(0, mload(add(_pointer, 0x40)))
    if lt(_v, 27) { _v := add(_v, 27) }
}
```

Lines: 240–247

Severity: 1

File: RLP.sol

```
assembly {
    let byteLen := sub(byte0, 0xb7) // # of bytes the actual
length is
    memPtr := add(memPtr, 1) // skip over the first byte

    /* 32 byte word size */
    let dataLen := div(mload(memPtr), exp(256, sub(32,
byteLen))) // right shifting to get the len
    itemLen := add(dataLen, add(byteLen, 1))
}
```

Lines: 198–200

Severity: 1

File: RLP.sol

```
assembly {
    destPtr := add(0x20, result)
}
```


Lines: 229–231

Severity: 1

File: RLP.sol

```
assembly {  
    byte0 := byte(0, mload(memPtr))  
}
```

Lines: 304–308

Severity: 1

File: RLP.sol

```
assembly {  
    let srcpart := and(mload(src), not(mask)) // zero out src  
    let destpart := and(mload(dest), mask) // retrieve the  
bytes  
    mstore(dest, or(destpart, srcpart))  
}
```

Lines: 53–55

Severity: 1

File: RLP.sol

```
assembly {  
    memPtr := add(item, 0x20)  
}
```

Lines: 294–296

Severity: 1

File: RLP.sol

```
assembly {
    let srcpart := and(mload(src), not(mask)) // zero out src
    let destpart := and(mload(dest), mask) // retrieve the
bytes
    mstore(dest, or(destpart, srcpart))
}
```

Lines: 164–171

Severity: 1

File: RLP.sol

```
assembly {
    result := mload(memPtr)

    // shfit to the correct location if neccesary
    if lt(len, 32) {
        result := div(result, exp(256, sub(32, len)))
    }
}
```

Lines: 129–131

Severity: 1

File: RLP.sol

```
assembly {
    ptr := add(0x20, result)
}
```

Lines: 255–261

Severity: 1

File: RLP.sol

```
assembly {
    let byteLen := sub(byte0, 0xf7)
    memPtr := add(memPtr, 1)

    let dataLen := div(mload(memPtr), exp(256, sub(32,
byteLen))) // right shifting to the correct length
    itemLen := add(dataLen, add(byteLen, 1))
}
```

Lines: 142–144

Severity: 1

File: RLP.sol

```
assembly {
    result := byte(0, mload(memPtr))
}
```

Lines: 183–185

Severity: 1

File: RLP.sol

```
assembly {
    result := mload(memPtr)
}
```

Lines: 112–114

Severity: 1

File: RLP.sol

```
assembly {
    byte0 := byte(0, mload(memPtr))
}
```

Lines: 270–272

Severity: 1

Compiler version not fixed

```
assembly {  
    byte0 := byte(0, mload(memPtr))  
}
```

File: RLP.sol

Lines: 4–4

Severity: 1

Solidity source files indicate the versions of the compiler they can be compiled with.

```
pragma solidity ^0.6.2;
```

```
library RLP {  
    uint8 constant STRING_SHORT_START = 0x80;  
    uint8 constant STRING_LONG_START  = 0xb8;  
    uint8 constant LIST_SHORT_START   = 0xc0;  
    uint8 constant LIST_LONG_START    = 0xf8;  
    uint8 constant WORD_SIZE = 32;
```

`pragma solidity ^0.4.17; // bad: compiles w 0.4.17 and above`

`pragma solidity 0.4.24; // good : compiles w 0.4.24 only`

It is recommended to follow the latter example, as future compiler versions may handle certain language constructions in a way the developer did not foresee.

Comments:

The use case of smart contracts is very well designed and Implemented. Overall, the code is clearly written and demonstrates the effective use of abstraction, separation of concerns, and modularity. **CERTIDAPP** development team demonstrated high technical capabilities, both in the design of the architecture and in the implementation.