CERTIK

Audit Proposal

PROPRIETARY AND CONFIDENTIAL

Submitted by: CertiK

Delivered on: 09/19/2023

Project Proposal: Security Audit

Prepared for: XinFin

EXECUTIVE SUMMARY

XinFin is seeking a third party to evaluate the codebase security, and CertiK will review and verify the project specifications and source code with a detailed focus on weaknesses, potential vulnerabilities, and overall security. CertiK will also address its findings with solutions that may mitigate future attacks or loopholes.

Project Background & Mission

CertiK's mission of the audit is to apply different types of approaches and detections, ranging from manual, static, and dynamic analysis, as well as formal verification, to ensure that XinFin is checked against known attacks and potential vulnerabilities. A highlight of failures and security issues include:

- Inconsistency between specification and implementation
- Flawed design, logic
- Reentrance, code injection, and Denial of Service attacks
- Limit exceeded on bytecode and gas usage
- Miner attacks on timestamp and ordering

CertiK leverages a team of seasoned engineers and security auditors to apply testing methodologies and verifications on XinFin's project, in turn creating a more secure and robust software system.

CertiK has served more than 3,000 clients with high quality auditing and consulting services, ranging from stablecoins such as Binance's BGBP and Paxos Gold to decentralized oracles such as Band Protocol and Tellor. CertiK customizes its engineering tool kits, while applying cutting-edge research on smart contracts, for each client on its project to offer a high quality a delivery. As XinFin utilizes technologies from blockchain and smart contracts, CertiK team will continue to support the project as a service provider and collaborator.

DLT METHODOLOGY

At CertiK, we implement a transparent process and make our reviews a collaborative effort. The goals of our security audits are to prove the soundness of protocol design, enhance the source code quality, and output sufficient remediations to the system. CertiK team utilizes the following methodologies in the security audit process.

Expert Manual Review

By leveraging experts in security-oriented code review, we greatly reduce the chance that potential bugs and vulnerabilities go undetected. This diligent review offers customized detections and suggestions about high-level system design and protocol logics. Our team also identifies areas where more defensive programming could be applied to reduce the risk of mistakes and loopholes. In addition to focusing on the source code in scope, we also examine code dependencies when they are relevant to the investigated area.

Automated Static Analysis

CertiK security engineers and researchers have put tremendous efforts into developing automated static analysis toolkits that increase the chances of detecting flaws and critical risks, including risks not easily detected by human efforts. Our approaches range from Syntax Analysis, Semantics Analysis, Vulnerability Base Analysis (60,000+ findings database), Rule Base Analysis (1,000+ rules), and Formal Verification, which mathematically proves the correctness of core components. We customize the threat models and attack scenarios for each project, then apply analytical examinations and investigations accordingly.

Interactive Dynamic Analysis

CertiK's security engineers and researchers have dedicated substantial efforts to crafting interactive dynamic analysis toolkits. These toolkits significantly enhance the likelihood of identifying vulnerabilities and critical risks, even those that might elude human inspection and static analysis methods. Our methodology encompasses a spectrum of techniques, including conventional unit and integration testing, advanced property-based fuzz testing, and interactive examination. Each project benefits from a tailored threat model and attack scenario, ensuring that our analytical scrutiny is precise and effective in uncovering potential issues.

In-depth Documented Findings

The whole process of auditing is thorough, transparent, interactive, and accompanied with milestone-driven preliminary comments/reports. Communication channels are opened between the teams of engineers for fast collaboration, and upon client request, this communication may be logged as well. All findings of the security analysis are fully documented with in-depth reasoning and supporting materials like testing outputs or screenshots for demonstrative purposes.

DLT METHODOLOGY - continued

Expressive Correctness Specification

CertiK verification engineers and researchers will, based on the documentation and code, create the formal specification of the artifacts to verify. The specification covers the security, safety, and correctness properties of the code implementation w.r.t. the documented or publicly stated invariants, assumptions, and guarantees. The specification can be written in Separation Logic or in the Calculus of Inductive Constructions (a higher-order formal logic), which gives expressive power that has been demonstrated to be adequate for handling complex mathematical, financial, protocol, and systems code.

Trustworthy Machine-checked Verification

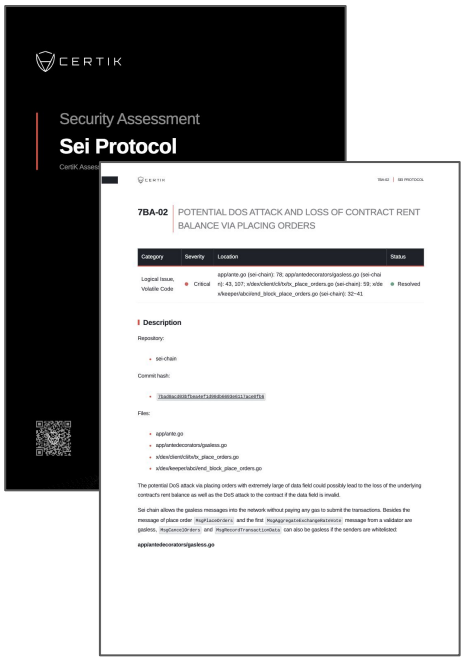
CertiK's verification engineers harness the power of our cutting-edge deductive formal verification framework, Scivik, along with automation tools, to ensure the safety, correctness, and security of code implementations in line with specifications. Our verification process is entirely rigorous and machine-checked, effectively eradicating potential human errors in proofs. Moreover, our CertiK team possesses an arsenal of broadly applicable specifications. We not only apply these specifications as a foundation but also offer supplementary services to tailor and refine specifications for each project. This ensures that our formal verification aligns perfectly with the distinct threat models, attack scenarios, and desired properties of each engagement.

Remediations and Recommendations

The primary objective is to offer the client with actionable items and upgrade suggestions from our analysis and discovery. CertiK engineers, seasoned with general software engineering and security experience, will try their best to outline or mitigate the vulnerabilities that may affect the system as a whole. At the completion and delivery of the final report, all critical and medium findings and recommendations should be resolved.

Layer-1 Case Studies

Case 1: Sei Protocol Audit



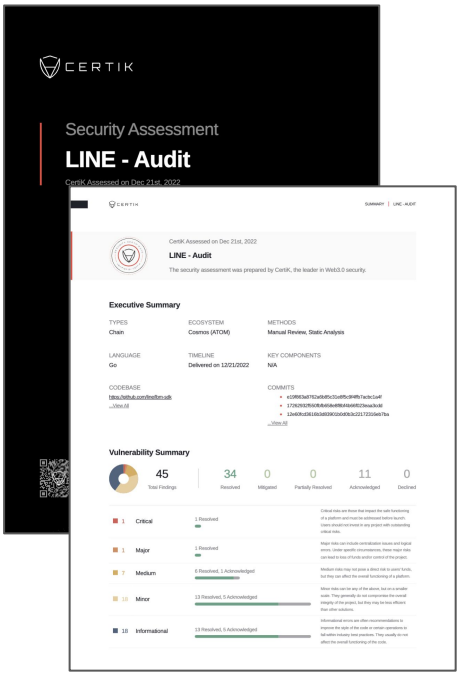
CertiK performed a full security audits before Sei Mainnet launched, CertiK identified (and confirmed by Sei) a Critical Issue where transaction validation was missing for a specific data field from Sei's unique gasless transaction design.

- Allows an attacker to Launch a DoS attack on the Sei protocol.
- Result in a loss of funds from linked contracts managing this transaction

Impacts:
With CertiK's guidance, Sei prevented potential attacks and secured multiple dex contracts from fund losses. Sei has token CertiK advices to refine its gasless feature design and will now charge extra for high data usage.

Conclusion:
Always validate user inputs to deter harmful actions. Also, ensure custom transaction handlers are tested and aligned with existing standards.

Audit Report:
https://skynet.certik.com/projects/sei-protocol



Case 2: Finschia (formaly Line Blockchain) - LBN-SDK Audit

CertiK performed a full security audits before Sei Mainnet launched, CertiK identified (and confirmed by Sei) a Critical Issue where transaction validation was missing for a specific data field from Sei's unique gasless transaction design.

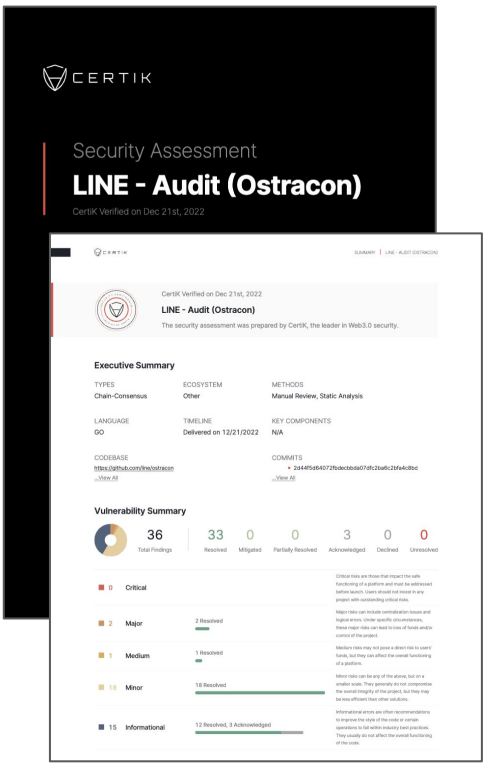
- Allows an attacker to Launch a DoS attack on the Sei protocol.
- Result in a loss of funds from linked contracts managing this transaction

Impacts:
With CertiK's guidance, Sei prevented potential attacks and secured multiple dex contracts from fund losses.
Sei has token CertiK advices to refine its gasless feature design and will now charge extra for high data usage.

Conclusion:
Always validate user inputs to deter harmful actions. Also, ensure custom transaction handlers are tested and aligned with existing standards.

Audit Report:
https://skynet.certik.com/projects/finschia

Layer-1 with Consensus Module Case Studies



Case 1: Finschia (formally Line Blockchain) Ostracon Audit

During the audits Ostracon Consensus Audit, CertiK identified (and confirmed by LineCorp) that missing gas restriction on the transactions

- Allows an attacker to exhaust the mempool to perform the DoS attack;

Additionally, CertiK identified (and confirmed by LineCorp) that there is a design flow in the voter set election algorithm

- Allows the attackers to control the validators in the voter set and eventually take over the consensus.

Impacts:
The client decided to work with CertiK to add the proper gas constraint to the transactions and remove the voter set election algorithm.

Conclusion:
The transaction gas should always be properly charged to avoid DoS attack and any modification to the Tendermint consensus algorithm needs to be thoroughly tested.

Audit Report:
https://skynet.certik.com/projects/finschia

Case 2: Massa Protocol - Consensus Audit

CertiK performed a new designed Consensus Audit for Massa Protocol, CertiK identified (and confirmed by Massa Lab) a Critical logical flaw in the voter set election algorithm.

- This loophole provides a window for a minority group of validators to control the voter set, enabling them to take control of the network consensus and manipulate the entire system.

Impacts:
The implications of such a vulnerability are grave. An attacker gaining control of the voter set disrupts the blockchain consensus, which can result in stalled transactions or even double spends. Such a security breach would have a profound effect on the network's integrity, operations, and reliability, consequently reducing its overall value proposition.

Conclusion:
The identified consensus vulnerability poses a severe threats, If exploited, it could enable a successful attack, leading to a complete takeover of the blockchain. The client addressed and resolved before the mainnet launch.

Audit Report:
https://skynet.certik.com/projects/massa-labs

ZKP SECURITY ASSESSMENT CASE STUDIES

This case study examines the critical security analyses conducted on various zero knowledge protocols and implementations, it illustrate common pitfalls and lessons learned for securely constructing zero knowledge primitives.

Description

It is possible to validate fake proofs with arbitrary public values. Since these proofs are used to update block headers, it may be possible to render the bridge inoperable by updating the headers to invalid values.

The main reason why fake proofs are accepted is because commitments associated to the wire polynomials $\mathbf{a}, \mathbf{b}, \mathbf{c}$ are not used in the challenges $\gamma, \beta, \alpha, \zeta$. This allows these polynomials to be decided after calculating ζ and chosen in a way to bypass the verification requirements.

Scenario

The following steps allow one to create a fake proof that will be accepted:

1. Choose a random public input w_1
2. Choose $\mathbf{z}, \mathbf{t}$ to be the zero polynomial.
   - This causes proof elements $[z]_1, [t_{lo}]_1, [t_{mid}]_1, [t_{hi}]_1, [W_\zeta]_1, \bar{z}_\omega$ to all be 0
3. Compute $\gamma, \beta, \alpha, \zeta$.
4. Choose random polynomials $\mathbf{a}, \mathbf{b}$. We use the zero polynomial for simplicity.
5. The requirement in `verify_quotient_poly_eval_at_zeta()` is simplified to the following:
   - $\bar{t} = \frac{\bar{r} + w_1 L_1(\zeta) - \bar{t}_1(\zeta)\mathbf{Z}_H(\zeta)}{\mathbf{Z}_H(\zeta)}$
   - This allows us to calculate $\bar{r} := \mathbf{r}(\zeta)$ as $\bar{t} = 0$
6. Compute $\mathbf{c}(\zeta)$. This is done through the following equation:
   - $\mathbf{r}(\zeta) = \bar{c}q_c(\zeta) + q_t(\zeta)$, which is acquired from the definition of $\mathbf{r}(X)$.
7. Choose a polynomial $\mathbf{c}(X)$ such that $\mathbf{c}(\zeta)$ is what we calculated above. For simplicity, we take $\mathbf{c}$ to be a constant polynomial, so $\mathbf{c}(x) = \mathbf{c}(\zeta)$ although we do not know what x is.
   - Since the value $[c]_1$ is never committed, we are able to make this choice.
8. Compute $\mathbf{r}(X) = \bar{c}q_c(X) + q_t(X)$.
9. Compute $\bar{s}_{\sigma_1}, \bar{s}_{\sigma_2}$.
10. Compute the challenge v.

Case 1: Critical Security Gap Unveiled: Counterfeit Proofs in PLONK-based ZK Bridge Project

A security audit of a cross-chain bridge project using the PLONK zero-knowledge proof revealed a critical vulnerability. This flaw allows counterfeit proof creation, bypassing verifier scrutiny. Originating from a misstep in implementing the Fiat-Schamir transformation, a key part of proof verification, the issue lies in the omission of a Fiat-Schamir hash computation. Consequently, attackers could manipulate public input values, leading to seemingly legitimate but false verifications—a potent reminder of the necessity for stringent security protocols.

Case 2: A ZK Rollup Project

CertiK conducted an audit for a cryptographic library that is used with Prover to create “circuits” for rollup blocks on a layer 2 solution. These circuits are then used to make SNARK proofs. These proofs are then sent to the smart contracts on the first layer.

The system uses cryptographic proofs made using zkSNARK, especially the **Groth16 proving system**. These proofs check the correctness of each transaction within a block and make sure that any changes in the block state are accurately shown.

The design of the block circuit is made specifically to make zkSNARK proofs using public inputs and block witnesses. It’s crucial that the constraints of the circuit design match the witness generation process to ensure they are equivalent. This match is key to make sure the zkSNARK proof is complete and sound. During the audit, we focused mostly on whether the block execution program and the block circuit were equivalent, and if the constraints were performed correctly. Below is a summary of situations where this match could break:

- **Incorrect Constraint:** If the constraints is incorrectly added, then the zkSNARK proof generation fails as the block witness will not make the circuit satisfiable.
- **Missing Constraint:** In the cases that a constraint is missing, then extra data could be injected into the witness such that the generated proof does not change.
- **Under-constraint:** If the circuit is underconstrained, the block witness could possibly be modified in such a way that the proof could still be generated. In other words, the proof could correspond to multiple instances of witness.
- **Lack of Overflow Check:** The computation inside circuits are performed over a finite field, which requires careful design to ensure that the passed witness does not exceed the modulus of the finite field. Meanwhile, the intermediate witness should also not be within the modulus to align with the computation inside the circuits.
- **Optimization:** A zkSNARK proof generation is extremely computationally intensive, a well designed circuit should be reduced to avoid too much redundancy but achieve the same goal.

Security Research Case Studies

Case 8: SUI' Blockchain HamsterWheel Attack

An innovative form of attack leverages a critical vulnerability to result in a complete network shutdown of the Sui blockchain

On 06/19/2023, CertiK identified (and confirmed by Sui) that Sui network is vulnerable to the "**HamsterWheel attack**"

- Allows an attacker to bring down the whole SUI network and a simple reboot won't recover the network
- Allows the attacker to block any new transactions submitted to the SUI network.
- SUI offered a half million dollar cash award bounty to CertiK for the finding.

Impacts:
CertiK helped SUI to fix the problem right after protocol mainnet launch.

Conclusion:
Layer one security feature (such as type safety) strongly relies on implementation correctness. Such implementation needs careful review by security professionals.

Reference Articles:
The HamsterWheel: An In-Depth Exploration of a Novel Attack Vector on the Sui Blockchain

REVIEW OBJECTIVES (Smart contract)

In order to analyze and detect potential vulnerabilities that could affect the project, CertiK engineers will conduct technical due diligence to assess and verify the modules' security correctness of design and implementation. The following list of checkpoints is included as reference:

Vulnerability	Checkpoints
Arithmetic	• Integer underflow/overflow • Floating Points and Precision
Access & Privilege Control	• Administrative functionality for control and emergency handling • Restriction access for sensitive functions and data • Rate limit for critical operations, permission to contract state changes, and delay operations for malicious/sensitive actions
Denial of Service	• Unexpected Revert • Gas spent exceeds its limit on Contract via unbounded operations or block stuffing
Miner Manipulation	• Block Number Dependence • Timestamp Dependence • Transaction Ordering Or Front-Running
External Referencing	• Correct usage of the pull over push favor for external calls • Correct usage of checks-effects-interactions pattern to minimize the state changes after external contract or call referencing • Avoid state changes after external calls • Error handling and logging
Race Conditions	• Reentrancy - unexpected state changes when call.value()() occurs • Cross-function racing - attacks that using different functions while share the same state
Low-level Call	• Code Injection by delegatecall • Unsuited adoption on assembly code
Visibility	• Specify the correct visibility of variables and functions
Incorrect Interface	• Ensure the defined function signatures match with the contract interface and implementation

SCOPE OF ENGAGEMENT

CertiK will assess the following files based on client's provided source code repository.

The audit process will utilize a mixture of expert manual review, static analysis, and dynamic analysis techniques.

The engagement was scoped to provide a security assessment of the XDC 2.0 Release.
Repository: https://github.com/XinFinOrg/XDPoSChain
Commit Hash: 78d25c6e620160de303dff52b7a85503cb5a3222

- All the code developed between V1.1.0 and V1.4.8 under the XDPoSChain repo.
- Ensuring the BFT security properties (safety and liveness) are met.
- Checking code quality, robustness, and coverage against potential network failures or security-threatening attacks.

Upon examining the XDPoSChain codebase, the audit can be segmented into chain and smart contract in golang and solidity language scope:

- XDCx related logic
  - XDCx
  - XDCxDAO
  - XDCxlending
- Consensus
- Core
- Native Smart Contract

SCOPE OF ENGAGEMENT

XDCx (5863)

The XDCx folder contains all the core building blocks and logic that are specific to XDCx on XDPoSChain. It provides the currency and transactional capabilities on top of the underlying XDPoSChain platform.

Some key responsibilities of the XDCx folder:

- Implements the XDCx state transition logic, including executing transactions, updating account balances, gas costs, etc.
- Manages the XDCx state trie, which stores account balances and contract storage.
- Implements the XDCx consensus rules and block validation logic.
- Provides APIs for sending XDCx transactions and querying XDCx state.
- Manages the XDCx transaction pool, pending transactions, and transaction ordering.
- Implements XDCx specific RPC APIs for wallet functionality, account management, etc.
- Contains XDCx specific configurations, parameters, fees, etc.

```
.
├── XDCx.go
├── api.go
├── order_processor.go
├── token.go
└── tradingstate
    ├── XDCx_trie.go
    ├── common.go
    ├── database.go
    ├── dump.go
    ├── encoding.go
    ├── epochpriceitem.go
    ├── journal.go
    ├── managed_state.go
    ├── orderitem.go
    ├── relayer_state.go
    ├── settle_balance.go
    ├── state_lendingbook.go
    ├── state_liquidationprice.go
    ├── state_orderItem.go
    ├── state_orderList.go
    ├── state_orderbook.go
    ├── statedb.go
    └── trade.go
```

SCOPE OF ENGAGEMENT

XDCxDAO (991)

The **XDCxDAO** abstracts away the underlying database implementation details and provides a consistent API for the rest of the XDCx code to work with the database. It handles performance optimization and translates between the database representations.

Some key responsibilities of the XDCxDAO include:

- Define interfaces for interacting with the XDCx database, including basic CRUD operations. The main interface is XDCXDAO.
- Provide concrete implementations of these interfaces for different database backends:
    - BatchDatabase - uses LevelDB for storage, provides put/get/delete methods
    - MongoDatabase - uses MongoDB, provides object storage methods
- Handle caching and batching to improve performance.
- Encapsulate the differences between the database backends behind a common interface.
- Manage connections and provide a common way to access the XDCx data layer across the codebase.

```
.
├── interfaces.go
├── leveldb.go
└── mongodb.go
```

SCOPE OF ENGAGEMENT

XDCxlending (5863)

The **XDCxlending** contains core lending logic and state for XDC, enabling lending functionality like creating orders, matching trades, tracking balances, and processing actions like repayments.

Some key responsibilities of the XDCxlending include:

- Manage lending order books, identified by the lending token and term. The GetLendingOrderBookHash function generates the order book hash.
- Track lending trades, which represent an individual lending transaction between a borrower and lender. The lending state database stores and retrieves these trades.
- Verify token balances when processing lending operations like top-ups, repayments, etc. The VerifyBalance function does these checks before allowing lending actions.
- Calculate repayment amounts based on lending terms like interest rate and liquidation time.
- Handle limit and market lending orders, with logic differing for the investing and borrowing sides.

```
.
├── XDCxlending.go
├── api.go
├── lendingstate
│   ├── XDCx_trie.go
│   ├── common.go
│   ├── database.go
│   ├── dump.go
│   ├── journal.go
│   ├── lendingcontract.go
│   ├── lendingitem.go
│   ├── managed_state.go
│   ├── relayer.go
│   ├── settle_balance.go
│   ├── state_itemList.go
│   ├── state_lendingbook.go
│   ├── state_lendingitem.go
│   ├── state_lendingtrade.go
│   ├── state_liquidationtime.go
│   ├── statedb.go
│   └── trade.go
└── order_processor.go
```

SCOPE OF ENGAGEMENT

Consensus (4309)

The **consensus** folder implements the core consensus mechanisms that allow nodes in the Ethereum network to securely collaborate and maintain a decentralized blockchain. The choice of consensus algorithm is foundational to the security, scalability and decentralization properties of a blockchain.

Some key responsibilities of the consensus include:

- XDPoS - Implementation of XinFin's XDPoS (XinFin Delegated Proof of Stake) consensus algorithm.
    - It is split into two main engines:
        - engine_v1 contains the implementation for XDPoS v1.
        - engine_v2 contains the implementation for XDPoS v2. This added improvements like dynamic validator set, slashing, and faster block times.
- Clique - Implementation of the Clique proof-of-authority consensus algorithm.
- Ethash - Implementation of the Ethash proof-of-work consensus algorithm.
- Containing utilities and common modules used by multiple consensus implementations like the misc package.Allowing the Ethereum node software to pluggable swap consensus algorithms based on config parameters. The consensus.go file handles this.

```
.
├── XDPoS
│   ├── XDPoS.go
│   ├── api.go
│   ├── engines
│   │   ├── engine_v1/*
│   │   └── engine_v2/*
│   └── utils/*
├── clique
│   ├── api.go
│   ├── clique.go
│   └── snapshot.go
├── consensus.go
├── errors.go
├── ethash
│   ├── algorithm.go
│   ├── consensus.go
│   ├── ethash.go
│   └── sealer.go
├── misc
│   ├── dao.go
│   └── forks.go
```

SCOPE OF ENGAGEMENT

Core (20322 12 days + privacy 14 days)

- Implementing the consensus rules and logic like validating blocks and transactions.
- Defining the data structures and types used in XDxcPOSChain like blocks, transactions, receipts, accounts etc.
- Managing the state tree and state transitions. The state folder contains the state database and all associated logic.
- Implementing the Ethereum fork Virtual Machine (EVM) which handles execution of smart contracts.
- Networking logic for node discovery and communication using the devp2p stack.
- APIs and utilities used by other Ethereum modules and applications.

- **Privacy**: a new zero-knowledge proof protocol for proving that a secret committed value lies in a given range. (paper, LoC 2000, 14 days)
    - https://github.com/XinFinOrg/XDPoSChain/blob/master/core/vm/privacy/bulletproof.go
    - During review, we'll examine these hypotheses:
        - **Cryptographic threats** like advances in discrete log, quantum computing, vulnerabilities like frozen heart in Fiat-Shamir transform, or inconcise in implementations.
        - **Implementation issues** like incorrect parameters, timing attacks, side channel attacks, and numeric stability problems.
        - **Ring Signature:**
            - Check signature verification and anonymity guarantees
            - Ensure security against revealing actual signer

```
├── asm
│   ├── asm.go
│   ├── compiler.go
│   └── lexer.go
├── block_validator.go
├── blockchain.go
├── blocks.go
├── bloombits
│   ├── doc.go
│   ├── generator.go
│   ├── matcher.go
│   └── scheduler.go
├── chain_indexer.go
├── chain_makers.go
├── database_util.go
├── error.go
├── events.go
├── evm.go
├── gaspool.go
├── gen_genesis.go
├── gen_genesis_account.go
├── genesis.go
├── genesis_alloc.go
├── headerchain.go
├── lending_pool.go
├── lending_tx_journal.go
├── lending_tx_list.go
├── mkalloc.go
├── order_pool.go
├── order_tx_journal.go
├── order_tx_list.go
├── rawdb/x
├── state
│   ├── database.go
│   ├── dump.go
│   ├── iterator.go
│   ├── journal.go
│   ├── managed_state.go
│   ├── state_object.go
│   ├── state_reader.go
│   ├── statedb.go
│   ├── statedb_utils.go
│   ├── sync.go
│   └── trc21_reader.go
├── state_processor.go
├── state_transition.go
├── token_validator.go
├── tx_journal.go
├── tx_list.go
├── tx_pool.go
├── types
│   ├── block.go
│   ├── bloom9.go
│   ├── consensus_v2.go
│   ├── derive_sha.go
│   ├── forensics.go
│   ├── gen_header_json.go
│   ├── gen_log_json.go
│   ├── gen_receipt_json.go
│   ├── gen_tx_json.go
│   ├── lending_signing.go
│   ├── lending_transaction.go
│   ├── log.go
│   ├── order_signing.go
│   ├── order_transaction.go
│   ├── receipt.go
│   ├── transaction.go
│   └── transaction_signing.go
├── types.go
└── vm
    ├── XDCx_price.go
    ├── analysis.go
    ├── common.go
    ├── contract.go
    ├── contracts.go
    ├── doc.go
    ├── eips.go
    ├── errors.go
    ├── evm.go
    ├── gas.go
    ├── gas_table.go
    ├── gen_structlog.go
    ├── instructions.go
    ├── int_pool_verifier.go
    ├── int_pool_verifier_empty.go
    ├── interface.go
    ├── interpreter.go
    ├── intpool.go
    ├── jump_table.go
    ├── logger.go
    ├── logger_json.go
    ├── memory.go
    ├── memory_table.go
    ├── opcodes.go
    ├── privacy
    │   ├── bulletproof.go
    │   └── ringct.go
    ├── runtime
    │   ├── doc.go
    │   ├── env.go
    │   ├── fuzz.go
    │   └── runtime.go
    ├── stack.go
    └── stack_table.go
```

SCOPE OF ENGAGEMENT

Solidity: Native Smart Contracts (2199)

The contracts folder contains all the Solidity smart contract source code for the XinFin XDPoSChain project. It is organized into subdirectories for each main contract system or application.

The key responsibilities of the contracts folder are:

- Implement core protocol functionality like accounts, transactions, consensus, etc.
- Implement decentralized applications like token contracts, name/identity registration, random number generation, etc.
- Provide libraries and utilities that are reused across multiple contracts.
- Define interfaces, inheritance structures and overall architecture for XinFin's smart contract ecosystem.

```
.
├── XDCx
│   ├── contract
│   │   ├── LendingRegistration.sol
│   │   ├── Registration.sol
│   │   ├── SafeMath.sol
│   │   ├── TRC21.sol
│   │   ├── TRC21Issuer.sol
│   │   ├── XDCXListing.sol
│   │   └── XDCXPrice.sol
├── blocksigner
│   └── contract
│       ├── BlockSigner.sol
│       └── libs
│           └── SafeMath.sol
├── chequebook
│   └── contract
│       ├── chequebook.sol
│       ├── mortal.sol
│       └── owned.sol
├── ens
│   └── contract
│       ├── AbstractENS.sol
│       ├── ENS.sol
│       ├── FIFSRegistrar.sol
│       └── PublicResolver.sol
├── multisigwallet
│   └── contract
│       └── MultiSigWallet.sol
├── randomize
│   └── contract
│       ├── XDCRandomize.sol
│       └── libs
│           └── SafeMath.sol
├── solidity_0.6
│   ├── abstract.sol
│   ├── array_resizing.sol
│   ├── fallback.sol
│   ├── overriding.sol
│   ├── receive.sol
│   ├── state_variable_shadowing_disabled.sol
│   └── trycatch.sol
├── tests
│   └── contract
│       ├── Inherited.sol
│       └── libs
│           └── SafeMath.sol
├── trc21issuer
│   ├── contract
│   │   ├── TRC21.sol
│   │   ├── TRC21Issuer.sol
│   │   └── libs
│   │       └── SafeMath.sol
│   └── simulation
│       ├── deploy
│       └── test
└── validator
    └── contract
        ├── XDCValidator.sol
        └── libs
            └── SafeMath.sol
```

SCHEDULING

After a thorough review from our engineering team of all provided materials, the price quote of this audit request is as follows. The price quote is based on the scope of work, date of completion, and resource allocation required.

1. Golang: XDCx related logic (XDCx, XDCxDAO, XDCxlending)	
Days Allocated	17 business days
Start Date (upon receipt of payment)	September 20th, 2023
Prelim. Report Delivery Date	October 12th, 2023
Resource Allocation	2 Golang Blockchain Security Experts
Final Price	**$68,000**

2. Golang: Consensus	
Days Allocated	15 business days
Start Date (upon receipt of payment)	September 20th, 2023
Prelim. Report Delivery Date	October 10th, 2023
Resource Allocation	2 Golang Blockchain Security Experts with Consensus Experience
Final Price	**$60,000**

SCHEDULING

After a thorough review from our engineering team of all provided materials, the price quote of this audit request is as follows. The price quote is based on the scope of work, date of completion, and resource allocation required.

3. Golang: Core + other modules	
Days Allocated	20 business days
Start Date (upon receipt of payment)	October 10th, 2023
Prelim. Report Delivery Date	November 6th, 2023
Resource Allocation	2 Golang Blockchain Security Experts
Final Price	**$80,000**

4. Golang: Core/privacy modules (ZK)	
Days Allocated	14 business days
Start Date (upon receipt of payment)	November 7th, 202
Prelim. Report Delivery Date	November 24th, 2023
Resource Allocation	2 Golang Blockchain Security Experts with ZK experience • 2 Math PHD
Final Price	**$56,000**

SCHEDULING

After a thorough review from our engineering team of all provided materials, the price quote of this audit request is as follows. The price quote is based on the scope of work, date of completion, and resource allocation required.

5. Solidity: Native Smart Contracts	
Days Allocated	6 business days
Start Date (upon receipt of payment)	October 10th, 2023
Prelim. Report Delivery Date	October 17th, 2023
Resource Allocation	2 Solidity Security Experts
Final Price	**$24,000**

6. Security Package	
Deliverables	• Skynet Smart Contract Monitoring • KYC • Contract Verification • Bug Bounty Set up
Final Price	**$13,000**

Package Options Offering

Overview

Understanding that each customer has unique needs and constraints, we have developed a **flexible Package Options Offering** for XDPoSChain security assessment services. This approach allows clients to **select only** the critical components that need an in-depth security review. This way, you have the freedom to tailor your security assessment package based on what you can afford and what you deem most critical.

Our Package Options Offering gives you the flexibility to receive a security assessment that aligns with your financial and temporal constraints, without compromising the quality of the service delivered. This customer-centric approach ensures you getting exactly what you need based on your constraints.

Item	Modules	Time	Price
1.	Golang: XDCx related logic (XDCx, XDCxDAO, XDCxlending)	17 days	$68,000.00
2.	Golang: Consensus	15 days	$60,000.00
3. (**)	Golang: Core + Other Modules	20 days	$80,000.00
	Golang: Core/privacy	14 days	$56,000.00
4	Solidity: Native Smart Contracts	6 days	$24,000.00
5	Security Package		$13,000.00
(**): We suggest assessing the Core module in conjunction with both the Core + Other modules and the Core/Privacy module.			

WORKFLOW AND DELIVERABLES

As the leading security service provider in the industry, the CertiK team workflow prioritizes a high quality service and an excellent customer experience.

Initial & Kick-Off Meetings with CertiK

- Bilaterally agree upon preferred communication channels, auditing goals, and action items regarding: Infrastructure, product design, ecosystems, economy model, and broader cybersecurity plans

Information Gathering, Project Research, Audit Planning, and Executions

- With all documents relevant to the client's project, perform an in-depth review, formalizing structures and plans by decomposing into smaller, auditable pieces
- Conduct the assessments based upon different approaches and methodologies, including manual, static, and dynamic analyses that are feasible for the corresponding project

Preliminary Reports

- Deliver preliminary (or weekly) reports to highlight findings/vulnerabilities/recommendations that could help the client utilize the results and address patches or updates quickly (Remediation Window)

Re-Audits

- Re-audit the design and the code changes implemented after the assessment period, holding engineering meetings, if necessary, for further discussions

Further Iterations

- Repeat the steps above to improve the code quality until acceptable security confidence is reached

Final Report

- Deliver a comprehensive report that includes all the details and practices of the audit project, which may serve as a certificate for cryptocurrency exchanges or a technical document for the client's engineering team to utilize as a reference

WORKFLOW AND DELIVERABLES

CertiK actively engages with clients by presenting outputs that help clients detect potential vulnerabilities and improve the overall engineering quality. To highlight the auditing deliverables:

CertiK actively engages with clients by presenting outputs that help clients detect potential vulnerabilities and improve the overall engineering quality. To highlight the auditing deliverables:

Communication Channels

- Real-time discussions and interactions via platforms like Slack, Telegram, and email

Dedicated Engineering Hours

- Standby around the clock to collaborate with the client's team about security-focused tech stacks, as well as general engineering practices that may help improve the overall quality of the project

Audit Reports

- Comprehensive list of vulnerabilities with detailed explanations to identify and resolve the issues
- Supporting materials for full transparency and understanding, including scripts or artifacts that CertiK leveraged to reproduce bugs and outputs

WHY CERTIK?

CertiK leads blockchain and smart contract security by pioneering the use of cutting-edge technologies, including static/dynamic analysis and Formal Verification. Auditing is among one of the premium solutions that CertiK offers to its clients in its mission to verify and ensure the correctness and security of software. CertiK also contributes to the technical communities and ecosystems by providing guidance, research, and advisory about blockchain and smart contract best practices. To date, CertiK has served more than 700 clients and secured over $30B in digital assets across all major protocols.

CertiK was founded by Computer Science professors from Yale University and Columbia University, with its technologies derived from years of research in academia. CertiK is backed by notable investors including Coatue, AptosLabs Ventures, Binance, Lightspeed Venture Partners, Shunwei Capital, and IDG. Additionally, CertiK has received grants from IBM, the Qtum Foundation, and the Ethereum Foundation to support its research of improving security across the blockchain industry.

Globee Awards
Global Awards Silver Winner

CBINSIGHTS Blockchain 50 2022
CBINSIGHTS
The Top Blockchain Companies

WORLD ECONOMIC FORUM
Global Innovator

Trusted by the leading clients

CertiK | **Securing** the **Web3** World

Learn more at www.certik.com