Report

v. 1.0

Customer

Exactly

# Smart Contract Audit

# Protocol Update

31st October 2025

Report prepared by

ABDK Consulting

# Contents

# 1 Changelog

| # | Date | Author | Description |
|---|---|---|---|
| 0.1 | 31.10.25 | A. Zveryanskaya | Initial Draft |
| 0.2 | 31.10.25 | A. Zveryanskaya | Minor revision |
| 1.0 | 31.10.25 | A. Zveryanskaya | Release |

# 2 Summary



This document presents the results of a smart contract audit performed by ABDK Consulting for the `Exactly`, conducted by Mikhail Vladimirov and Dmitry Khovratovich during 17th October to 31st October. The audit goal was to assess correctness and safety of access control, error propagation, and edge-case handling across the System, with particular focus on verified components, adapters, and lifecycle operations as reflected in the issues list and subsequent fixes, including `Firewall` logic adjustments.

Our conclusion is that the System demonstrates solid maturity and good code quality, using `Solidity` 0.8.x, `OpenZeppelin` upgradeable patterns (`Initializable`, `AccessControlUpgradeable`), and established libraries such as `solmate` and `solady`. The team promptly addressed identified issues with targeted refactors and tests. The access control model in `Firewall.sol` was improved to correctly handle allower role revocation, and tests were extended to cover disallow semantics.

General recommendations are to continue tightening access control invariants in `Firewall` and verified modules, prefer modifiers for repeated allowedness checks, standardize error forwarding for clearer root-cause analysis, add explicit range validations where user-facing parameters admit wide domains, and consolidate top-level declarations into contracts or dedicated files for navigability. Final verdict: the System is well engineered with responsive maintenance, and the implemented changes materially strengthen authorization guarantees and operability across verified components. Included CVF references from the CSV and cited the reviewed commit. Emphasized access control, error handling, and key improvements to Firewall and related tests.

**It is important to note that a security audit is not a guarantee of absolute security but rather a snapshot of the system's security posture at a specific point in time. While we strive to uncover all potential issues, the evolving nature of blockchain technology and DeFi means new vulnerabilities can emerge.**

# 3 System overview

This section provides a high-level overview of the Exactly System as assessed by our team.

The audit was based on the following diffs:

- Original Code
- Code with Fixes

Files:

| | | | |
|---|---|---|---|
| **/** | | | |
| | Auditor.sol | Market.sol | MarketBase.sol |
| | MarketExtension.sol | RewardsController.sol | |
| **verified/** | | | |
| | VerifiedMarket.sol | VerifiedAuditor.sol | Firewall.sol |
| **periphery/** | | | |
| | FlashLoanAdapter.sol | DebtRoller.sol | |

The System enables permissioned market operations where accounts must be explicitly allowed before interacting, with an auditor component that can lock or unlock activity, a market component that enforces allowlist checks, and auxiliary adapters for flash liquidity and debt management. The access-control boundary is centralized in Firewall, which governs who may permit or revoke access. The System is implemented in Solidity 0.8.x, with some components constrained to 0.8.20 due to named mappings in FlashLoanAdapter. The codebase uses OpenZeppelin upgradeable patterns, specifically Initializable and AccessControlUpgradeable, and relies on math and transfer utilities from solmate and solady. Testing and execution flows are consistent with a Foundry-based workflow as indicated by test files and cheatcodes. Token standards and vault abstractions are integrated through IERC20 and IERC4626 interfaces. The System's repository includes targeted modifications that focus on firewall authorization behavior and expanded test coverage link.

At the core is Firewall, which maintains an allowlist of accounts and enforces an ALLOWER_ROLE for authorizing changes. The allowlist registry associates an account with an originator and an allowed flag, and it emits errors when attempting duplicate enables or unauthorized disables. The recent change introduces role-aware disallow semantics that verify whether the previous allower still holds the allower role before

denying a non-originator, which allows deactivation to proceed if the original allower has lost privileges. Tests cover allowing, disallowing, cross-allower attempts, events, and behavior when the original allower's role is revoked, demonstrating a comprehensive authorization surface.

`VerifiedAuditor` coordinates account status and interacts with `Firewall`. It defines domain-specific events and errors, including invalid initializer and operation cases and an explicit not-allowed condition. The locking routine centralizes administrative actions related to restricting activity. The issues list discusses separating account locking from asset-related procedures to keep lock operations bounded in cost and focused on authorization state. The component emits a firewall-configuration event, which shows explicit linkage between the auditor and the authorization layer, and it underpins the permission model that upstream markets depend on.

`VerifiedMarket` builds on the base `Market` and systematically enforces allowedness checks on callers and relevant counterparties. Calls that transfer, borrow, repay, or withdraw are guarded by an internal allowedness requirement, ensuring only accounts present on the firewall allowlist can execute market operations. The base `Market` improves error forwarding through an `ExtensionFailed` pathway that avoids masking errors returned by extensions, aiding diagnosis and making behavior consistent when extensions fail. Declared domain errors in the market domain capture invariant violations such as zero-amount operations and protocol liquidity constraints.

The `FlashLoanAdapter` integrates the System with an external vault API to source flash liquidity. It maintains a registry of wrapped tokens and their relationship to underlying assets, enabling conversions when direct liquidity is sparse. The adapter includes input validation and custom error signaling for unauthorized vaults and insufficient liquidity. It follows array-based interfaces compatible with an external vault, and provides an event for changes to wrapped-token configuration. The approach is tailored to environments where liquidity often resides in wrapped or boosted instruments, enabling the System to fulfill flash operations using designated wrappers.

`DebtRoller` coordinates short-lived borrowing to restructure positions using a flash lender abstraction. It stores a short-hash of call data to ensure response integrity across callbacks and computes proportional amounts through fixed-point arithmetic utilities from `solady`. The component receives arrays of tokens, principals, and fees via a callback interface and enforces a simple invalid-operation error on misuse, aligning with an explicit revert-on-violation style. Parameterization uses a percentage value with clear treatment for values at or above a full scale, ensuring predictable behavior during partial or full roll operations.

Component relationships are explicit. `VerifiedAuditor` links to `Firewall` to control allowlist state for accounts. `VerifiedMarket` depends on this allowlist through allowedness checks to gate operations by caller and target addresses. `Market` extensions communicate failures through standardized error forwarding, which simplifies upstream handling. `FlashLoanAdapter` connects to an external vault interface and a wrapped-token registry so the System can utilize wrapped liquidity where direct reserves are insufficient. `DebtRoller` consumes flash liquidity to adjust positions atomically, integrating with the same external callback conventions.

External connections use interfaces for tokens and vaults. The adapter references a

vault API consistent with `Balancer v3` semantics through an internal interface, while supporting `IERC20` and `IERC4626` wrappers to convert between asset and share units. Client notes indicate that wrapped tokens are used to bridge to boosted liquidity in the vault ecosystem. Across modules, access control is consistently managed through `AccessControlUpgradeable` roles and allowlist checks near user-facing entry points. The latest firewall change aligns disallow paths with current role ownership, preventing stale authorizations and reinforcing the System's permission perimeter.

# 4 Methodology

The methodology is not a strict formal procedure, but rather a selection of methods and tactics combined differently and tuned for each particular project, depending on the project structure and technologies used, as well as on client expectations from the audit.

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows best code practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.
- **Entity Usage Analysis**. Usages of various entities defined in the code are analysed. This includes both: internal usages from other parts of the code as well as potential external usages. We check that entities are defined in proper places as well as their visibility scopes and access levels are relevant. At this phase, we understand overall system architecture and how different parts of the code are related to each other.
- **Access Control Analysis**. For those entities, that could be accessed externally, access control measures are analysed. We check that access control is relevant and done properly. At this phase, we understand user roles and permissions, as well as what assets the system ought to protect.
- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. We check if code actually does what it is supposed to do, if that algorithms are optimal and correct, and if proper data types are used. We also make sure that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.

We classify issues by the following severity levels:

- **Critical issue** directly affects the smart contract functionality and may cause a significant loss.
- **Major issue** is either a solid performance problem or a sign of misuse: a slight code modification or environment change may lead to loss of funds or data. Sometimes it is an abuse of unclear code behaviour which should be double checked.
- **Moderate issue** is not an immediate problem, but rather suboptimal performance in edge cases, an obviously bad code practice, or a situation where the code is correct only in certain business flows.
- **Recommendations** cover code style, best practices and general improvements.

# 5 Our findings

We've provided the client with some recommendations.

| | Info | Fixed |
|---|---|---|
| Moderate | 6 | 2 |
| Recommendations | 23 | 6 |

# 6 Moderate Issues

## CVF-1. INFO

- **Category** Suboptimal
- **Source** VerifiedAuditor.sol

**Description** This function does too many things and could exceed the block gas limit, which would make it impossible to lock certain accounts.

**Recommendation** Separate account locking from seizing account's assets. Locking should just mark account as locked, effectively preventing further operations with this account. Seizing account's assets should be performed after locking in separate transactions, one or several markets per transaction.

**Client Comment** *Even though the Seize event is emitted, there's no actual transfer of assets here. The account's access to the assets is removed by disallowing the account to interact with the protocol (in the Firewall contract), and here we're only de-assigning them and removing from totalAssets, so the market doesn't account for these assets.*

```
35 +function lock(address account) external onlyAllowed(msg.sender) {
```

## CVF-3 FIXED

- **Category** Unclear behavior
- **Source** Firewall.sol

**Description** In case "allowed" is false, the "allower" field should be cleared.

**Client Comment** *Ok, we updated the way this was handled to store only the allower when it's allowed.*

```
35 +allowlist[account] = Allowed({ allower: msg.sender, allowed:
   ↪ allowed });
```

## CVF-4 INFO

- **Category** Procedural
- **Source** FlashLoanAdapter.sol

**Description** Here arrays passed as argument are modified, which is a bad practice, even though these modification cannot be seen by the caller.

**Recommendation** Use local variables instead.

**Client Comment** *We prefer this way to reduce gas consumption.*

```
29 +tokens[0] = IERC20(address(wToken));
```

```
32 +amounts[0] = shares;
```

## CVF-5 INFO

- **Category** Procedural
- **Source** DebtRoller.sol

**Recommendation** This variable should be declared as “transient”.

**Client Comment** *We already have this implementation in another contract working fine. We also tried this approach using solady LibTransient and the gas consumption was worse.*

```
29 +bytes32 private callHash;
```

## CVF-6 INFO

- **Category** Unclear behavior
- **Source** DebtRoller.sol

**Description** There is no range check for this argument.

**Recommendation** Implement an appropriate check.

**Client Comment** *For better UX, there’s no range for this argument because if the value is higher than 1e18 then we roll 1e18 (100%).*

```
50 +uint256 percentage
```

## CVF-7 INFO

- **Category** Unclear behavior
- **Source** DebtRoller.sol

**Description** Any percentage value higher than 1e18 is silently treated as 1e18. This could make error investigations harder.

**Recommendation** Revert on incorrect percentage values.

**Client Comment** *For better UX, there’re no incorrect percentage values, when it’s above 100% we use 100%.*

```
90 +uint256 positionAssets = r.percentage < 1e18 ? r.percentage.mulWad(
   ↪ principal + fee) : principal + fee;
```

## CVF-8 INFO

- **Category** Procedural
- **Source** Auditor.sol

**Description** This argument is not used.

**Recommendation** Remove it.

**Client Comment** *The method is virtual and when overriding, we use it. it's used on VerifiedAuditor, where this function is extended.*

```
242 +address
```

## CVF-9 FIXED

- **Category** Unclear behavior
- **Source** Market.sol

**Description** This makes it impossible to distinguish the "ExtensionFailed" error throws here and the same error thrown from within the extension.

**Recommendation** Either always forward the error message from extension as is (even when the message is empty) or always wrap it into an error.

**Client Comment** *Ok, we changed it.*

```
952 +if (data.length == 0) revert ExtensionFailed();
```

```
954 +assembly ("memory-safe") {
955 +  revert(add(32, data), mload(data))
956 +}
```

# 7 Recommendations

## CVF-2 INFO

- **Category** Flaw
- **Source** FlashLoanAdapter.sol

**Description** If the wToken was already set for this asset, Assets deposited there could be lost.

**Recommendation** Either move the assets from the old wToken to new new one or explicitly forbid setting wToken for an assets that already has wToken.

**Client Comment** *The contract doesn't hold any assets, there's no deposit. The idea of this mapping is that if there's a flashLoan request of an asset that there's no liquidity, we can check if there's a wrapped token for that asset and use it to fulfill the request. The story behind this contract is that most of the Balancer v3 liquidity is aave boosted, so we need to wrap/unwrap. Overwriting could be an intended update if we want to use another wrapped token for the same asset.*

```
68 +wTokens[asset] = token;
```

## CVF-10 FIXED

- **Category** Procedural
- **Source** Firewall.sol

**Description** Consider specifying as "^0.8.0" unless there is something special regarding this particular version.

**Recommendation** See also: VerifiedMarket.sol, VerifiedAuditor.sol, FlashLoanAdapter.sol, DebtRoller.sol.

**Client Comment** *We changed except for the flash loan adapter, where we're using named mappings and we need 0.8.20 version.*

```
2 +pragma solidity ^0.8.17;
```

## CVF-12 FIXED

- **Category** Procedural
- **Source** Firewall.sol

**Description** We didn't review this file.

**Client Comment** *Ok - it was unused so we dropped it.*

```
6 +import { FixedPointMathLib } from "solmate/src/utils/
    ↪ FixedPointMathLib.sol";
```

## CVF-13 INFO

- **Category** Suboptimal
- **Source** Firewall.sol

**Description** Declaring top level structures and errors in a file named after a contract makes it harder navigating through code.

**Recommendation** Move the declarations into the contract or move them in a separate file.

**Client Comment** *Style choice, we prefer to stay consistent with the base code.*

```
50 +struct Allowed {
```

```
55 +error AlreadyAllowed(address account);
56 +error NotAllower(address account, address allower);
```

## CVF-14 FIXED

- **Category** Procedural
- **Source** VerifiedMarket.sol

**Recommendation** It is a good practice to put a comment into an empty block to explain why the block is empty.

**Client Comment** *Ok - we added missing natspec.*

```
10 +constructor(ERC20 asset_, VerifiedAuditor auditor_) Market(asset_,
   ↪ auditor_) {}
```

```
143 +function handleRewards(bool, address) internal override {} //
   ↪ solhint-disable-line no-empty-blocks
```

```
145 +function setRewardsController(RewardsController) public override {}
   ↪  // solhint-disable-line no-empty-blocks
```

## CVF-15 INFO

- **Category** Procedural
- **Source** VerifiedMarket.sol

**Recommendation** This should be turned into a modifier.

**Client Comment** *Style choice, it reduces the bytecode size.*

```
13  +_requireAllowed(receiver);
20  +_requireAllowed(receiver);
31  +_requireAllowed(receiver);
36  +_checkIsAuditor();
75  +_checkIsAuditor();
101 +_requireAllowed(borrower);
107 +_requireAllowed(borrower);
118 +_requireAllowed(borrower);
123 +_requireAllowed(to);
128 +_requireAllowed(to);
139 +_requireAllowed(owner);
14  +_requireAllowed(msg.sender);
19  +_requireAllowed(msg.sender);
30  +_requireAllowed(msg.sender);
100 +_requireAllowed(msg.sender);
106 +_requireAllowed(msg.sender);
117 +_requireAllowed(msg.sender);
```

## CVF-16 INFO

- **Category** Procedural
- **Source** VerifiedMarket.sol

**Description** Declaring top level events in a file named after a contract makes it harder navigating through code.

**Recommendation** Move the declarations into the contract or move them into a separate file.

**Client Comment** *Style choice, we prefer to stay consistent with the base code.*

```
156 +event Locked(address indexed account, uint256 assets);
157 +event Unlocked(address indexed account, uint256 assets);
```

## CVF-17 INFO

- **Category** Procedural
- **Source** VerifiedAuditor.sol

**Description** We didn't review these files.

**Client Comment** *Ok.*

```
5   +import { FixedPointMathLib } from "solmate/src/utils/
        ↪ FixedPointMathLib.sol";
```

## CVF-18 FIXED

- **Category** Procedural
- **Source** VerifiedAuditor.sol

**Recommendation** It is a good practice to put a comment into an empty block to explain why the block is empty.

**Client Comment** *Added missing natspec.*

```
16  +constructor(uint256 priceDecimals_) Auditor(priceDecimals_) {}
```

## CVF-19 INFO

- **Category** Procedural
- **Source** VerifiedAuditor.sol

**Description** Declaring top-level events and errors in a file named after a contract makes it harder navigating through code.

**Recommendation** Move the declarations into the contract or move them into a separate file.

**Client Comment** *Style choice, we prefer to stay consistent with the base code.*

```
115 +event FirewallSet(Firewall indexed firewall);
```

```
117 +error InvalidInitializer();
118 +error InvalidOperation();
119 +error NotAllowed(address account);
```

## CVF-20 INFO

- **Category** Procedural
- **Source** VerifiedAuditor.sol

**Recommendation** These errors could be made more useful by adding certain parameters into them.

**Client Comment** *Style choice, there's only one reason for them to happen.*

```
117 +error InvalidInitializer();
118 +error InvalidOperation();
```

## CVF-21 INFO

- **Category** Suboptimal
- **Source** FlashLoanAdapter.sol

**Recommendation** It would be more efficient to merge these mappings into a single mapping whose keys are wTokens and values are structs encapsulating the values of the original mappings.

**Client Comment** *They're not mapped by the same keys. one is a mapping to know what's the wrapped version of an asset. and the other one is a flag to know if the received assets is wrapped.*

```
11 +mapping(IERC20 wToken => bool isWToken) public isWToken;
12 +mapping(IERC20 asset => IERC4626 wToken) public wTokens;
```

## CVF-22 INFO

- **Category** Suboptimal
- **Source** FlashLoanAdapter.sol

**Recommendation** It would be more efficient to pass a single array of structures with two fields, rather than two parallel arrays. This would also make the length check unnecessary.

**Client Comment** *We want to stay compatible with balancer v2 interface.*

```
19 +function flashLoan(address recipient, IERC20[] memory tokens,
   ↪ uint256[] memory amounts, bytes memory data) external {
```

## CVF-23 FIXED

- **Category** Suboptimal
- **Source** FlashLoanAdapter.sol

**Recommendation** Here "require" should be used instead of "assert" as the conditions could be violated because of external reasons.

**Client Comment** *We changed it to use custom errors.*

```
20 +assert(tokens.length == 1);
21 +assert(amounts.length == 1);
```

## CVF-24 INFO

- **Category** Procedural
- **Source** FlashLoanAdapter.sol

**Recommendation** This interface should be moved into a separate file named "IBalancerVaultV3.sol".

**Client Comment** *Style choice, we prefer to stay consistent with the base code.*

```
74 +interface IBalancerVaultV3 {
```

## CVF-25 INFO

- **Category** Procedural
- **Source** FlashLoanAdapter.sol

**Recommendation** This interface should be moved into a separate file named "IFlashLoanRecipient.sol".

**Client Comment** *Style choice, we prefer to stay consistent with the base code.*

```
80 +interface IFlashLoanRecipient {
```

## CVF-26 INFO

- **Category** Procedural
- **Source** FlashLoanAdapter.sol

**Description** Declaring top-level errors and event in a file named after a contract makes it harder navigating through code.

**Recommendation** Move the declarations into the contract of move then into a separate file.

**Client Comment** *Style choice, we prefer to stay consistent with the base code.*

```
89 +error InsufficientLiquidity();
90 +error UnauthorizedVault();
```

```
92 +event WTokenSet(IERC20 indexed asset, IERC4626 indexed wToken,
   ↪ address indexed account);
```

## CVF-27 INFO

- **Category** Procedural
- **Source** FlashLoanAdapter.sol

**Recommendation** These errors could be made more useful by adding certain parameters into them.

**Client Comment** *Style choice, there's only one reason for them to happen.*

```
89 +error InsufficientLiquidity();
90 +error UnauthorizedVault();
```

## CVF-28 INFO

- **Category** Procedural
- **Source** DebtRoller.sol

**Description** We didn't review these files.

**Client Comment** *Ok.*

```
7 +import { FixedPointMathLib } from "solady/src/utils/
  ↪ FixedPointMathLib.sol";
8 +import { SafeTransferLib } from "solady/src/utils/SafeTransferLib.
  ↪ sol";
```

## CVF-29 INFO

- **Category** Procedural
- **Source** DebtRoller.sol

**Recommendation** This interface should be moved into a separate file named "IFlashLoanRecipient.sol".

**Client Comment** *Style choice, we prefer to stay consistent with the base code.*

```
12 +interface IFlashLoanRecipient {
```

## CVF-30 INFO

- **Category** Suboptimal
- **Source** DebtRoller.sol

**Recommendation** It would be more efficient to pass a single array of structs with three fields, rather than three parallel arrays. This would also make the length check unnecessary.

**Client Comment** *We want to stay compatible with balancer v2 interface.*

```
14 +IERC20[] memory tokens,
15 +uint256[] memory amounts,
16 +uint256[] memory fees,
```

## CVF-31 INFO

- **Category** Bad naming
- **Source** DebtRoller.sol

**Description** This function looks like a pure utility, while actually it has an unexpected side effect.

**Recommendation** Rename into something like "storeHash".

**Client Comment** *We like this readability and we're keeping the same name and structure we used in other contracts for the same purpose.*

```
78 +function _hash(bytes memory data) internal returns (bytes memory) {
79 +  callHash = keccak256(data);
```

## CVF-32 FIXED

- **Category** Suboptimal
- **Source** DebtRoller.sol

**Recommendation** Here "require" should be used instead of "assert", as the condition could be violated due to incorrect usage of the contract.

**Client Comment** *We changed it to use custom errors.*

```
85 +assert(msg.sender == address(flashLoaner) && memCallHash ==
     ↪ keccak256(data));
```

## CVF-33 INFO

- **Category** Procedural
- **Source** DebtRoller.sol

**Description** Declaring a top-level structure in a file named after a contract makes it harder navigating through code.

**Recommendation** Move the structure declaration into the contract or move it into a separate file.

**Client Comment** *Style choice, we prefer to stay consistent with the base code.*

```
105 +struct RollFixedData {
```

## CVF-34 INFO

- **Category** Procedural
- **Source** DebtRoller.sol

**Recommendation** This interface should be moved into a separate file named "IFlashLoaner.sol".

**Client Comment** *Style choice, we prefer to stay consistent with the base code.*

```
115 +interface IFlashLoaner {
```

## CVF-35 INFO

- **Category** Procedural
- **Source** DebtRoller.sol

**Description** Declaring a top-level error in a file named after a contract makes it harder navigating through code.

**Recommendation** Move the error declaration into the contract or move it into a separate file.

**Client Comment** *Style choice, we prefer to stay consistent with the base code.*

```
119 +error InvalidOperation();
```

## CVF-36 INFO

- **Category** Suboptimal
- **Source** DebtRoller.sol

**Recommendation** This error could be made move useful by adding certain parameters into it.

**Client Comment** *Style choice, we prefer to stay consistent with the base code.*

```
119 +error InvalidOperation();
```

## CVF-37 INFO

- **Category** Procedural
- **Source** Market.sol

**Description** Declaring top-level errors in a file named after a contract make is harder navigating through code.

**Recommendation** Move the declarations into the contract or move them into a separate file.

**Client Comment** *Style choice, we prefer to stay consistent with the base code.*

```
1454 1089  error Disagreement();
     1090 +error ExtensionFailed();
1455 1091  error InsufficientProtocolLiquidity();
1456 1092  error MarketFrozen();
     1093 +error MaxSupplyExceeded();
1457 1094  error NotAuditor();
1458 1095  error NotPausingRole();
1459 1096  error SelfLiquidation();
1460 1097  error ZeroBorrow();
1461 1098  error ZeroDeposit();
1462 1099  error ZeroRepay();
1463 1100  error ZeroWithdraw();
```

## CVF-38 INFO

- **Category** Suboptimal
- **Source** Market.sol

**Recommendation** These errors could be made more useful by adding certain parameters into them.

**Client Comment** *Style choice, we prefer to stay consistent with the base code.*

```
1454 1089  error Disagreement();
     1090 +error ExtensionFailed();
1455 1091  error InsufficientProtocolLiquidity();
1456 1092  error MarketFrozen();
     1093 +error MaxSupplyExceeded();
1457 1094  error NotAuditor();
1458 1095  error NotPausingRole();
1459 1096  error SelfLiquidation();
```

ABDK
Consulting

## About us

Established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function.

The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 300 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

## Contact


**Email**

dmitry@abdkconsulting.com

**Website**

abdk.consulting

**Twitter**

twitter.com/ABDKconsulting

**LinkedIn**

linkedin.com/company/abdk-consulting