CERTIK

USDD - Ethereum Update

Security Assessment

CertiK Assessed on Sept 2nd, 2025

CertiK Assessed on Sept 2nd, 2025

USDD - Ethereum Update

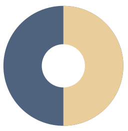
The security assessment was prepared by CertiK.

Executive Summary

TYPES	ECOSYSTEM	METHODS
StableCoin	Ethereum (ETH)	Manual Review, Static Analysis

LANGUAGE	TIMELINE
Solidity	Preliminary comments published on 09/02/2025 Final report published on 09/02/2025

Vulnerability Summary



8	5	0	3	0
Total Findings	Resolved	Partially Resolved	Acknowledged	Declined

🟧	0	Centralization		Centralization findings highlight privileged roles & functions and their capabilities, or instances where the project takes custody of users' assets.
🟥	0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
🟧	0	Major		Major risks may include logical errors that, under specific circumstances, could result in fund losses or loss of project control.
🟨	0	Medium		Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
🟨	4	Minor	1 Resolved, 3 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
🟦	4	Informational	4 Resolved	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | USDD - ETHEREUM UPDATE

CODEBASE | USDD - ETHEREUM UPDATE

Repository

- https://github.com/usdd-network/usddv2-contracts
- https://github.com/usdd-network/psm

Commit

- e0087ff4dc34f759441e526dc79b14e6678344b8
- 507d7d0ab4fcf1e18fcd6ce377c9d3aa7aad8444
- 6166a18fab3abcb1f8395707f613923d0278f534
- 0a6756eb55af7d2e134ceee2484ade9a9f4fa2a7

Audit Scope

The file in scope is listed in the appendix.

APPROACH & METHODS | USDD - ETHEREUM UPDATE

This audit was conducted for USDD to evaluate the security and correctness of the smart contracts associated with the USDD - Ethereum Update project. The assessment included a comprehensive review of the in-scope smart contracts. The audit was performed using a combination of Manual Review and Static Analysis.

The review process emphasized the following areas:

- Architecture review and threat modeling to understand systemic risks and identify design-level flaws.
- Identification of vulnerabilities through both common and edge-case attack vectors.
- Manual verification of contract logic to ensure alignment with intended design and business requirements.
- Dynamic testing to validate runtime behavior and assess execution risks.
- Assessment of code quality and maintainability, including adherence to current best practices and industry standards.

The audit resulted in findings categorized across multiple severity levels, from informational to critical. To enhance the project's security and long-term robustness, we recommend addressing the identified issues and considering the following general improvements:

- Improve code readability and maintainability by adopting a clean architectural pattern and modular design.
- Strengthen testing coverage, including unit and integration tests for key functionalities and edge cases.
- Maintain meaningful inline comments and documentations.
- Implement clear and transparent documentation for privileged roles and sensitive protocol operations.
- Regularly review and simulate contract behavior against newly emerging attack vectors.

REVIEW NOTES | USDD - ETHEREUM UPDATE

Overview

USDD is a fork of the MakerDAO (Sky Ecosysten) Protocol on TRON, a CDP-based stablecoin. The team plans to deploy a new USDD system on Ethereum. This audit focuses on several file changes between:

- USDD repo: adb799135095620b2909abe449af174102bf38e2 → e0087ff4dc34f759441e526dc79b14e6678344b8
- PSM repo: c368445876334664d6e58b7e186c78162271ff93 → 507d7d0ab4fcf1e18fcd6ce377c9d3aa7aad8444

Audit Scope:

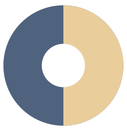
- usddv2
  - `src/dsr/SavingsUsdd.sol` : Tokenized DSR (sUSDD) (ERC4626 token) wrapper around USDD. Implements ERC-20 shares with EIP-712 permit and ERC-1271 support.
  - `src/dsr/pot.sol` : The `Pot` contract is where a USDD holder would lock up Internal USDD to accrue earned USDD at the USDD Savings Rate.
  - `src/dss/flop.sol` : Auction contract changes to support Ethereum deployments without JST token. Adds `GemLike.decimals()` , `file("gem", address)` to set the `gem` post-deploy, “gem-not-set” guards, event for address file, and `convertFrom18` when transferring out `lots` to non-18-decimals gem.
  - `src/esm/ESM.sol` : Makes `gem` mutable via file("gem") (was immutable). Adds checks in `join/burn` to require gem set.
  - `src/esm/end.sol` : Integrates `pot` into shutdown. Adds `file("pot", address)` and calls `pot.cage()` during `End.cage()` , ensuring DSR is disabled on shutdown.
  - `src/proxy/proxy.sol` : The updates enhance the `DSProxy` for Solidity `0.6.12` . The `Constructor` now invokes `setCache` with enhanced input validation; `execute(address,bytes)` returns bytes and appropriately propagates revert data; require statements have more precise messages; and handling of `returndata` in assembly is safer, among other improvements.
  - `src/manager/DssProxyActions.sol` : Transitions TRX/WTRX flows to ETH/WETH. Renames and rewires functions (`lockETH` , `freeETH` , `exitETH` , `lockETHAndDraw` , `wipeAndFreeETH` , etc.), removes unnecessary `convertTo18` for ETH, fixes signed/unsigned casts (e.g., `-toInt(art)`), and ensures ETH joins/withdrawals use WETH correctly.
- psm
  - `src/join-5-auth.sol` : USDC adapter (authorized join) for non-18 decimals tokens.
  - `src/join-7-auth.sol` : USDT adapter (authorized join) for non-18 decimals, upgradable token and fee-on-transfer token.

External Dependencies

USDD on Ethereum will depend on a full DSS stack (vat, vow, jug, dog, spot, esm, end, pot), robust on-chain oracles (medianizers/feeds) for timely, manipulation-resistant prices, and reliable keeper infrastructure to run auctions

(`flop` / `flap`), trigger `pot.drip()` and liquidation flows. We assume the contracts out of the audit scope above are implemented properly, the off-chain keepers are reliable, and governance is well managed.

FINDINGS | USDD - ETHEREUM UPDATE



8	0	0	0	0	4	4
Total Findings	Critical	Centralization	Major	Medium	Minor	Informational

This report has been prepared for USDD to identify potential vulnerabilities and security issues within the reviewed codebase. During the course of the audit, a total of 8 issues were identified. Leveraging a combination of Manual Review & Static Analysis the following findings were uncovered:

ID	Title	Category	Severity	Status
USD-02	Potential Underflow Issue And Reentrancy Issue Introduced By Uncertain Gem Token	Coding Issue, Volatile Code	Minor	● Acknowledged
USD-03	Potential Precision Loss Causing Unfair Bid	Inconsistency, Logical Issue	Minor	● Acknowledged
USD-04	Potential Inconsistent Decimals Between `wad` And `gem`	Volatile Code	Minor	● Acknowledged
USD-05	Fee-On-Transfer Token May Break The Assumption Of PSM Contract	Volatile Code	Minor	● Resolved
USD-01	Confirmation Of New Deployment On Ethereum And Cross Chain Design	Design Issue	Informational	● Resolved
USD-06	Potential Incorrect And Unused Interface For USDT/Upgradable Token	Design Issue	Informational	● Resolved
USD-07	Potential Inaccurate Event	Inconsistency	Informational	● Resolved
USD-08	Potential Missing Optimization With `toInt`	Coding Issue	Informational	● Resolved

USD-02 Potential Underflow Issue And Reentrancy Issue Introduced By Uncertain Gem Token

Category	Severity	Location	Status
Coding Issue, Volatile Code	● Minor	dss/flop.sol (usddv2): 203, 235	● Acknowledged

Description

Since gem token can be updated dynamically to any address (non-standard token), there could be several risks below:

Potential Underflow Issue

The `convertFrom18(uint256 wad)` function performs token amount conversion assuming the gem token has `<= 18` decimals. In Solidity `^0.6.12` , unsigned integer subtraction (e.g., `18 - gem.decimals()`) is unchecked and wraps around on underflow. If the token's decimals are greater than 18, this may create a risk of denial-of-service (DoS) or incorrect token transfers in `deal()` .

```
function convertFrom18(uint256 wad) internal view returns (uint256 amt) {
    amt = wad / (10 ** uint256(18 - gem.decimals()));
}
```

Potential Reentrancy Issue

In the `deal(uint id)` function, an external call is made to `gem.transfer(bids[id].guy, convertFrom18(bids[id].lot))` before updating contract state (specifically, before `delete bids[id])` . This violates the checks-effects-interactions (CEI) pattern. If the gem's transfer has any callback mechanism, the attacker can potentially repeat actions to drain funds.

```
    function deal(uint id) external {
        ...
        gem.transfer(bids[id].guy, convertFrom18(bids[id].lot));
        delete bids[id];
        ...
    }
```

Recommendation

It is recommended to ensure the gem's decimals are always below 18 before setting its address. Additionally, it is advisable to follow the checks-effects-interactions (CEI) pattern or avoid using tokens with potential callback mechanisms.

Alleviation

[USDD, 09/02/2025]: The team will avoid using tokens with potential callback mechanisms when setting the gem. To address the underflow issue, the team will ensure they use a gem token with decimals of 18 or fewer. Since these check is performed off-chain. We therefore mark this finding as Partially Resolved.

USD-03 | Potential Precision Loss Causing Unfair Bid

Category	Severity	Location	Status
Inconsistency, Logical Issue	● Minor	dss/flop.sol (usddv2): 234~236	● Acknowledged

Description

The current Flopper contract, part of debt auction contract, allows dynamic replacement of the `gem` token via the `file()` function. The `convertFrom18` function assumes the `gem` token uses a fixed 18-decimal precision (WAD) for internal `lot` amounts but converts to the `gem` 's actual decimals for transfers in `deal()` . This introduces precision loss via integer division when `gem.decimals() < 18` , potentially causing auction winners to receive fewer tokens than bid for, leading to unfair outcomes.

Recommendation

Recommend the team double-check if this design is intended.

Alleviation

[USDD, 09/01/2025]: The team confirms that this behavior is intentional, as the system is primarily designed for gem tokens with 18 decimals. In the event that a token with 6 decimals is used, the resulting loss would be less than 0.000001 gem, which is within acceptable limits.

USD-04 | Potential Inconsistent Decimals Between `wad` And `gem`

Category	Severity	Location	Status
Volatile Code	● Minor	esm/ESM.sol (usddv2): 155	● Acknowledged

Description

The ESM contract allows updating the `gem` token address, but accumulates transferred amounts (`wad`) into `Sum` without normalizing for token decimals. This can lead to inconsistent comparisons against the `min` threshold if `gem` is changed to a token with `decimals != 18` , potentially breaking the emergency shutdown logic.

```
    function join(uint256 wad) external {
        ...
        sum[msg.sender] = add(sum[msg.sender], wad);
        Sum = add(Sum, wad);

        require(gem.transferFrom(msg.sender, address(this), wad), "ESM/transfer-
failed");
        ...
    }
```

Recommendation

Recommend that the team ensure the gem token's decimals are always 18.

Alleviation

[USDD, 09/01/2025]: We will ensure that the gem token for ESM uses 18 decimals.

[CertiK, 09/02/2025]: Since this check is performed off-chain, it's possible that the gem token on-chain may not equal 18 decimals. We therefore mark this finding as Acknowledged.

USD-05 | Fee-On-Transfer Token May Break The Assumption Of PSM Contract

Category	Severity	Location	Status
Volatile Code	● Minor	join-7-auth.sol (psm): 103~105	● Resolved

Description

In the current `AuthGemJoin7` contract, when users deposit the external token (“gem”), the adapter credits the Vat with the actual net amount received (post-fee) by measuring the token balance delta and converting it to 18 decimals. This design is trying to support fee-on-transfer tokens.

```
// AuthGemJoin7
// For an upgradable token (like USDT) which doesn't return bool on transfers and
may charge fees
//  If the token is deprecated changing the implementation behind, this prevents
joins
//   and exits until the implementation is reviewed and approved by governance.

    function join(address usr, uint256 amt, address msgSender) external auth {
        ...
        uint256 bal = gem.balanceOf(address(this));
        gem.transferFrom(msgSender, address(this), amt);
        uint256 wad = mul(sub(gem.balanceOf(address(this)), bal), 10 ** (18 - dec));
        ...
        vat.slip(ilk, usr, int256(wad));
        ...
    }
```

However, the `UsddPsm` contract currently does not know this scenario. It assumes the full requested `gemAmt` was credited and uses that inflated figure in `vat.frob`, which may cause the unintended behavior (like transaction revert) in the `vat` contract.

```
    // Sell gem for USDD
    function sellGem(address usr, uint256 gemAmt) external {
        ...
        uint256 gemAmt18 = mul(gemAmt, to18ConversionFactor);
        uint256 fee = mul(gemAmt18, tin) / WAD;
        uint256 usddAmt = sub(gemAmt18, fee);

        // Transfer gem in and mint USDD
        gemJoin.join(address(this), gemAmt, msg.sender);
        vat.frob(ilk, address(this), address(this), address(this), int256(gemAmt18),
int256(gemAmt18));
        ...
    }
```

Recommendation

It is recommended not to support fee-on-transfer tokens, as the DSS system may have other parts that are incompatible with such tokens. If this is the intended design, advise the team to fix the sell path to use the actual credited amount.

Alleviation

[USDD, 09/01/2025]: As for the USDT-related PSM and the PSM GemJoin contract, we modified them to account for the USDT fee as follows:

In the AuthGemJoin7 contract, we return the actual wad value that the user sends into the contract.

We introduced another UsddPsm7 contract specific to USDT. In this PSM contract, we obtain the wad value from the AuthGemJoin7 contract, then use it to calculate the amount of USDD minted to the user, and credit the actual amount in the vat.

Fixed at commit: fb2b1fa2f0f93b273a51755bea2c874a371b4c20 and 4e53d61b10e6d6424aae41ab744bbba07cc74497.

USD-01 | Confirmation Of New Deployment On Ethereum And Cross Chain Design

Category	Severity	Location	Status
Design Issue	● Informational		● Resolved

Description

We would like to confirm whether the plan is to deploy an entirely new, Ethereum-native DSS stack (new USDD and JST supplies) with no bridging from TRON, rather than using cross-chain bridging (Teleport/DAI Bridge analogue) for USDD between TRON and Ethereum.

- Will Ethereum have a brand-new USDD (fresh mint on L1) with its own policy and supply, independent of TRON USDD? Could you provide the metadata, like the decimals, for this token?
- Will a brand-new JST (governance) be minted on Ethereum, or will an existing ERC-20 JST be reused? Could you provide the metadata, like the decimals, for this token?
- Based on the updates to the `Flopper` and `ESM` contracts, the team appears to prioritize deploying these contracts initially and attaching the JST token later. Without this governance token, certain functions of the `Flopper` , `ESM` contracts, and the governance voting process will be inactive. We would like to know if this design is intended.
- Will the team deploy a full DSS stack on Ethereum: vat, vow, jug, dog, spot, esm, end, pot (new), joins, flops/flaps, etc.?

Recommendation

Alleviation

[USDD, 09/01/2025]:

1. USDD will be deployed natively on Ethereum. In addition, cross-chain bridges may be used to link TRON USDD with Ethereum, but the Ethereum-native USDD contract is independent and freshly minted on L1, with 18 decimals.
2. The governance design is part of our roadmap. The current architecture allows for governance integration at a later stage, and the specific implementation approach is under evaluation.
3. Yes, this design is intentional: the core contracts will be deployed first, with certain configurations to be added later.
4. Yes, the full DSS stack modules will be deployed.

USD-06 | Potential Incorrect And Unused Interface For USDT/Upgradable Token

Category	Severity	Location	Status
Design Issue	● Informational	join-7-auth.sol (psm): 33~34, 78	● Resolved

Description

The `setImplementation` and `adjustFee` are not part of USDT's interface.

```
interface GemLike {
    function decimals() external view returns (uint256);
    function transfer(address, uint256) external;
    function transferFrom(address, address, uint256) external;
    function balanceOf(address) external view returns (uint256);
    function upgradedAddress() external view returns (address);
    function setImplementation(address, uint256) external;
    function adjustFee(uint256) external;
}
```

Additionally, upgradable tokens that don't have `upgradedAddress()` function (which is not a standard) will not work with this adapter.

Recommendation

Recommend removing the unused interface and having another adapter for the other upgradable tokens.

Alleviation

[USDD, 09/01/2025]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/usdd-network/psm/commit/ec50495606e7fe02491e0471d876844ff4f7f663

USD-07 | Potential Inaccurate Event

Category	Severity	Location	Status
Inconsistency	● Informational	join-7-auth.sol (psm): 108	● Resolved

Description

The events are defined to log `wad` (18-decimal internal amount):

```
event Join(address indexed usr, uint256 wad, address indexed msgSender);
event Exit(address indexed usr, uint256 wad);
```

However, the code emits `amt` (token's native decimals, pre-scaling)

```
emit Join(usr, amt, msgSender);  // should be wad
emit Exit(usr, amt);             // should be wad
```

Recommendation

Recommend the team double-check if this design is intended or ensure the consistency.

Alleviation

[USDD, 09/01/2025]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/usdd-network/psm/commit/d202ef9f2605b3c0ca5ab9e48095aba88fed44d2

USD-08 | Potential Missing Optimization With `toInt`

Category	Severity	Location	Status
Coding Issue	● Informational	manager/DssProxyActions.sol (usddv2): 565, 576	● Resolved

Description

Compared with the last audited commit adb799135095620b2909abe449af174102bf38e2, the team attempted to replace `-int(art)` with `-toInt(art)` to prevent any possible int overflow.

```
         frob(
             manager,
             cdp,
-            -toInt(convertTo18(trxJoin, amtC)),
-            -int(art)
+            -toInt(amtC),
+            -toInt(art)
         );

@@ -770,7 +770,7 @@ contract DssProxyActions is Common {
             manager,
             cdp,
             -toInt(wadC),
-            -int(art)
+            -toInt(art)
         );
```

We would like to remind the team that the following code is still using `-int`.

```
    function wipeAll(
        address manager,
        address usddJoin,
        uint cdp
    ) public {
        ...
        if (own == address(this) || ManagerLike(manager).cdpCan(own, cdp,
address(this)) == 1) {
            ...
            frob(manager, cdp, 0, -int(art));
        } else {
            ...
            VatLike(vat).frob(
                ilk,
                urn,
                address(this),
                address(this),
                0,
                -int(art)
            );
        }
    }
```

Recommendation

Recommend preventing unintended overflow.

Alleviation

[USDD, 09/01/2025]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/usdd-network/usddv2-contracts/commit/d104e662e2db472a493452466dc74a828e225378

APPENDIX | USDD - ETHEREUM UPDATE

Audit Scope

usdd-network/usddv2-contracts
esm/ESM.sol
dss/flop.sol
manager/DssProxyActions.sol
dsr/SavingsUsdd.sol
dsr/pot.sol
esm/end.sol
proxy/proxy.sol
dsr/SavingsUsdd.sol
dsr/pot.sol
dss/flop.sol
esm/ESM.sol
esm/end.sol
manager/DssProxyActions.sol
proxy/proxy.sol

usdd-network/psm
join-7-auth.sol
join-5-auth.sol
join-5-auth.sol
join-7-auth.sol

Finding Categories

Categories	Description
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
Inconsistency	Inconsistency findings refer to different parts of code that are not consistent or code that does not behave according to its specification.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Design Issue	Design Issue findings indicate general issues at the design level beyond program logic that are not covered by other finding categories.

DISCLAIMER | CERTIK

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

Elevating Your Web3 Journey

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is the largest blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.