



Security Audit

RocketPool (DeFi)

Table of Contents

Executive Summary	4
Project Context	4
Audit Scope	7
Security Rating	8
Intended Smart Contract Functions	9
Code Quality	12
Audit Resources	12
Dependencies	12
Severity Definitions	13
Status Definitions	14
Audit Findings	15
Centralisation	17
Conclusion	18
Our Methodology	19
Disclaimers	21
About Hashlock	22

CAUTION

THIS DOCUMENT IS A SECURITY AUDIT REPORT AND MAY CONTAIN CONFIDENTIAL INFORMATION. THIS INCLUDES IDENTIFIED VULNERABILITIES AND MALICIOUS CODE WHICH COULD BE USED TO COMPROMISE THE PROJECT. THIS DOCUMENT SHOULD ONLY BE FOR INTERNAL USE UNTIL ISSUES ARE RESOLVED. ONCE VULNERABILITIES ARE REMEDIATED, THIS REPORT CAN BE MADE PUBLIC. THE CONTENT OF THIS REPORT IS OWNED BY HASHLOCK PTY LTD FOR USE OF THE CLIENT.

Executive Summary

The RocketPool team partnered with Hashlock to conduct a security audit of their smart contracts. Hashlock manually and proactively reviewed the code in order to ensure the project's team and community that the deployed contracts are secure.

Project Context

Rocket Pool is a decentralised liquid staking protocol built for the Ethereum ecosystem that allows users to stake ETH and earn rewards without needing to meet the standard 32 ETH solo validator requirement. Users deposit any amount (as low as 0.01 ETH in some cases) to mint the protocol's liquid staking token rETH, which represents their staked ETH plus accumulated rewards and remains tradable in DeFi. Meanwhile, node operators participate by staking a smaller amount of ETH (e.g., 16 ETH) together with tokenised collateral in the native governance/governance-insurance token RPL to run validators and earn additional commissions and rewards. The protocol is designed to promote decentralisation, reduce barriers to staking, and enhance liquidity in the Ethereum staking layer.

Project Name: RocketPool

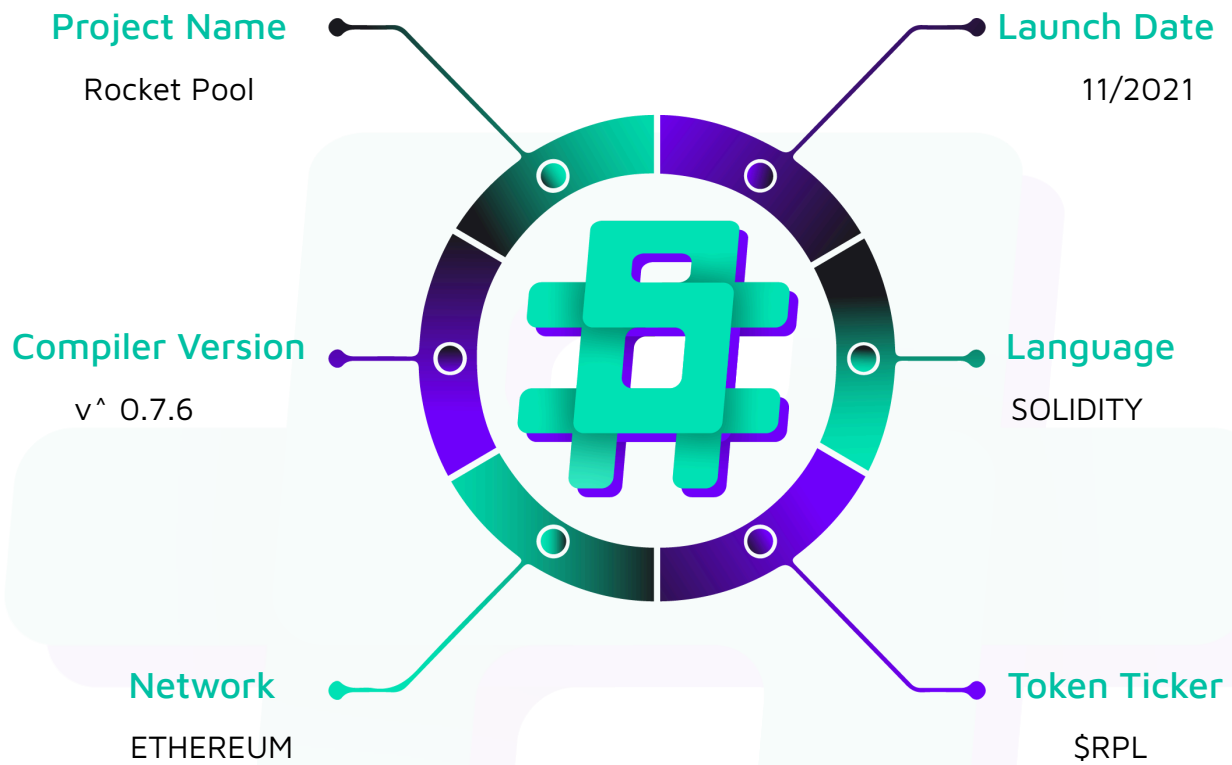
Project Type: Defi

Compiler Version: ^ 0.7.6

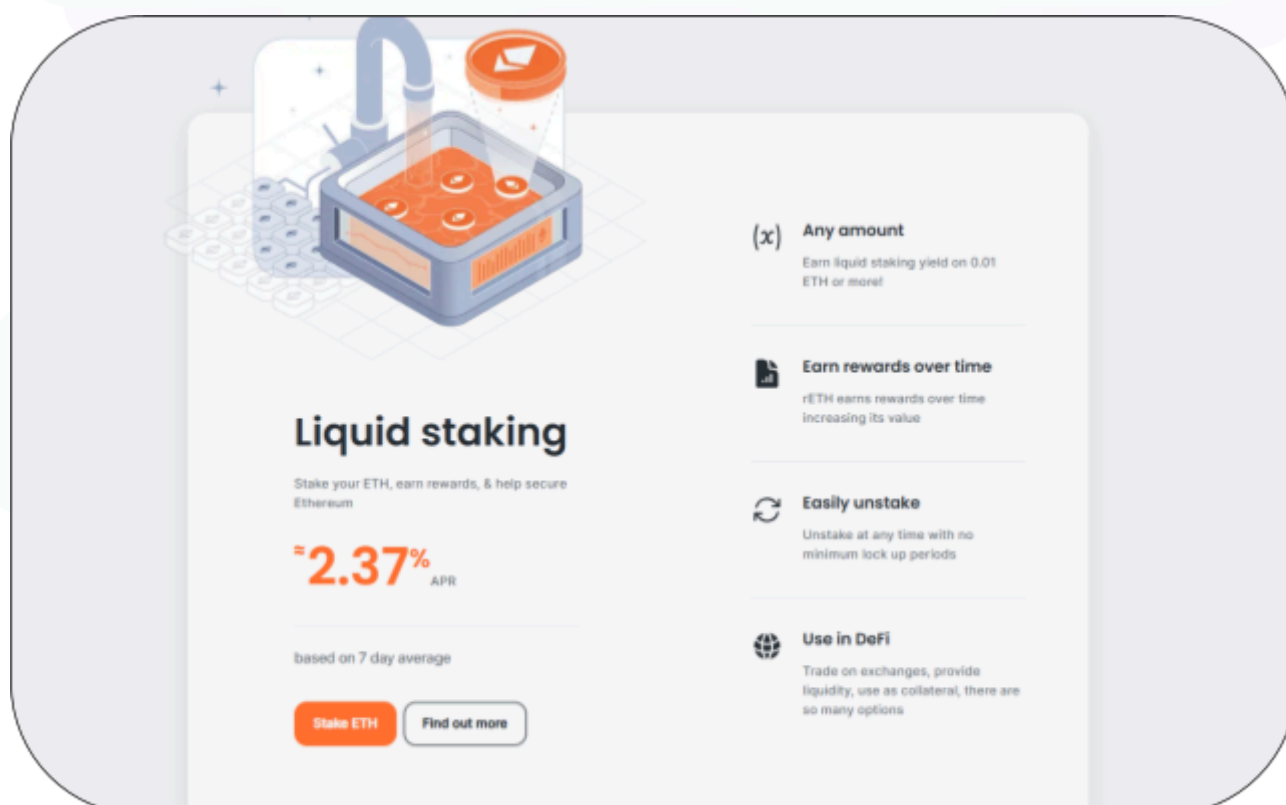
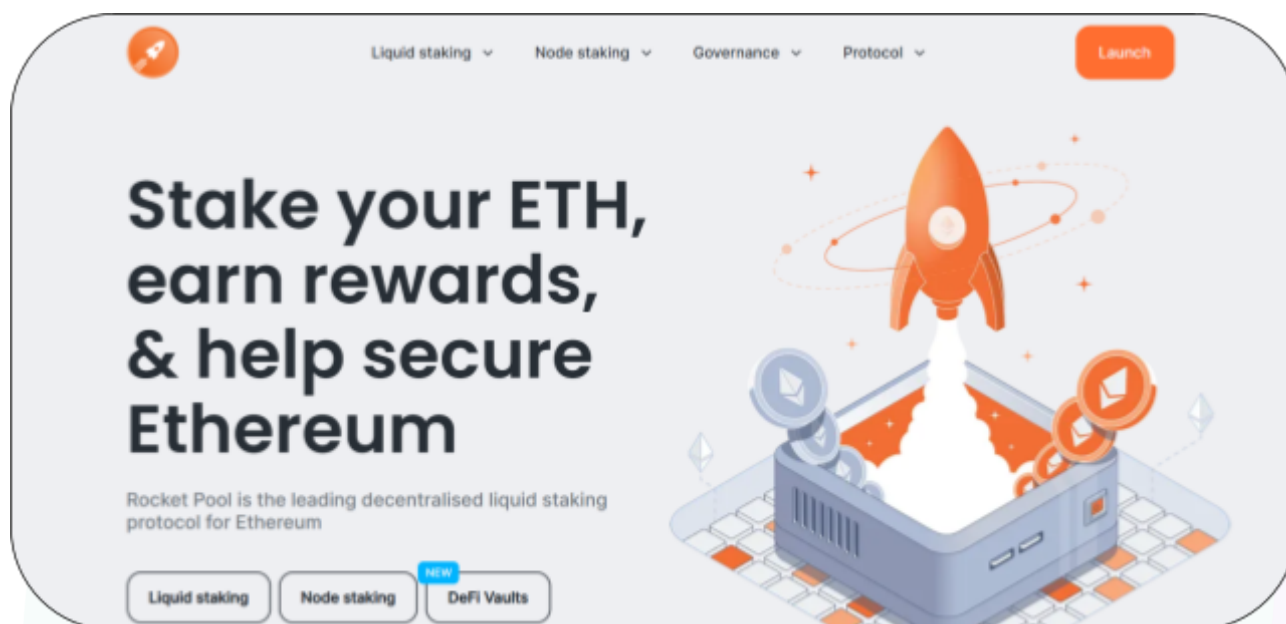
Website: <https://rocketpool.net/>

Logo:



Visualised Context:

Project Visuals:



Audit Scope

We at Hashlock audited the solidity code within the RocketPool project. The scope of work included a comprehensive review of the smart contracts listed below. We tested the smart contracts to check for their security and efficiency. These tests were undertaken primarily through manual line-by-line analysis and were supported by software-assisted testing.

Description	RocketPool Smart Contracts
Platform	Ethereum / Solidity
Audit Date	January, 2026
Contract 1	contracts/contract/megapool/*.sol
Contract 2	contracts/contract/deposit/*.sol
Contract 3	contracts/contract/node/*.sol
Contract 4	contracts/contract/token/*.sol
Contract 5	contracts/contract/network/*.sol
Contract 6	contracts/contract/dao/*.sol
Contract 7	contracts/contract/util/*.sol
Contract 8	contracts/contract/rewards/*.sol
Contract 9	contracts/contract/upgrade/*.sol
Audited GitHub Commit Hash	c9d9cbf2288ccbffe014a442b0ee458dcfc3dfe1

Security Rating

After Hashlock's Audit, we found the smart contracts to be **"Secure"**. The contracts all follow simple logic, with correct and detailed ordering. They use a series of interfaces, and the protocol uses a list of Open Zeppelin contracts.



Not Secure

Vulnerable

Secure

Hashlocked

The 'Hashlocked' rating is reserved for projects that ensure ongoing security via bug bounty programs or on chain monitoring technology.

All issues uncovered during automated and manual analysis were meticulously reviewed and applicable vulnerabilities are presented in the [Audit Findings](#) section. The list of audited assets is presented in the [Audit Scope](#) section and the project's contract functionality is presented in the [Intended Smart Contract Functions](#) section.

All vulnerabilities initially identified have now been resolved.

Hashlock found:

1 Medium severity vulnerabilities

Caution: Hashlock's audits do not guarantee a project's success or ethics, and are not liable or responsible for security. Always conduct independent research about any project before interacting.

Intended Smart Contract Functions

Claimed Behaviour	Actual Behaviour
contracts/contract/megapool/*.sol - Allows users to: - View megapool validator status and balances - Manage validator lifecycle and claims - Repay debt and distribute rewards - Allows admins to: - Deploy and upgrade megapool proxies - Stake/exit validators via manager proofs - Apply penalties and challenge exits	Contract achieves this functionality.
contracts/contract/deposit/*.sol - Allows users to: - Deposit ETH and mint rETH - View pool, queue, and credit balances - Withdraw node credits when available - Allows admins to: - Assign deposits to pools and queues - Recycle excess or dissolved funds - Process queue assignments and exits	Contract achieves this functionality.
contracts/contract/node/*.sol - Allows users to: - Register nodes and set withdrawal addresses - Deposit ETH for validator creation - Stake, unstake, and withdraw RPL	Contract achieves this functionality.

- Allows admins to: - Lock, burn, slash, or transfer RPL - Create node distributor proxies - Manage node reward and queue settings	
contracts/contract/token/*.sol - Allows users to: - Burn rETH to redeem ETH - Swap fixed-supply RPL for new RPL - Trigger inflation minting of RPL - Allows admins to: - Mint rETH from deposit pool - Accept excess collateral deposits - Mint dummy RPL tokens (owner)	Contract achieves this functionality.
contracts/contract/dao/*.sol - Allows users to: - Propose, vote, and execute DAO actions - Join or leave trusted/security councils - Challenge proposals and verify votes - Allows admins to: - Bootstrap settings and upgrades - Change protocol parameters via proposals - Veto upgrades through security DAO	Contract achieves this functionality.
contracts/contract/util/*.sol - Allows users to: - Library/Interface — callable helpers only - Allows admins to: - Library/Interface — callable helpers only	Utility libraries/interfaces only
contracts/contract/rewards/*.sol - Allows users to: - Claim rewards via merkle proofs	Contract achieves this functionality.

- Stake claimed RPL or withdraw ETH - View reward indices and balances - Allows admins to: - Submit and execute reward snapshots - Relay reward trees to distributor - Create and manage DAO payouts	
contracts/contract/upgrade/*.sol - Allows users to: - None - Allows admins to: - Configure upgrade addresses and ABIs - Execute protocol upgrades and inits - Add or upgrade registered contracts	Contract achieves this functionality.

Code Quality

This audit scope involves the smart contracts of the RocketPool project, as outlined in the Audit Scope section. All contracts, libraries, and interfaces mostly follow standard best practices and to help avoid unnecessary complexity that increases the likelihood of exploitation, however, some refactoring were recommended to optimize security measures.

The code is very well commented on and closely follows best practice nat-spec styling. All comments are correctly aligned with code functionality.

Audit Resources

We were given the RocketPool project smart contract code in the form of GitHub access.

As mentioned above, code parts are well commented. The logic is straightforward, and therefore it is easy to quickly comprehend the programming flow as well as the complex code logic. The comments are helpful in providing an understanding of the protocol's overall architecture.

Dependencies

As per our observation, the libraries used in this smart contracts infrastructure are based on well-known industry standard open source projects.

Apart from libraries, its functions are used in external smart contract calls.

Severity Definitions

The severity levels assigned to findings represent a comprehensive evaluation of both their potential impact and the likelihood of occurrence within the system. These categorizations are established based on Hashlock's professional standards and expertise, incorporating both industry best practices and our discretion as security auditors. This ensures a tailored assessment that reflects the specific context and risk profile of each finding.

Significance	Description
High	High-severity vulnerabilities can result in loss of funds, asset loss, access denial, and other critical issues that will result in the direct loss of funds and control by the owners and community.
Medium	Medium-level difficulties should be solved before deployment, but won't result in loss of funds.
Low	Low-level vulnerabilities are areas that lack best practices that may cause small complications in the future.
Gas	Gas Optimisations, issues, and inefficiencies.
QA	Quality Assurance (QA) findings are informational and don't impact functionality. Supports clients improve the clarity, maintainability, or overall structure of the code.

Status Definitions

Each identified security finding is assigned a status that reflects its current stage of remediation or acknowledgment. The status provides clarity on the handling of the issue and ensures transparency in the auditing process. The statuses are as follows:

Significance	Description
Resolved	The identified vulnerability has been fully mitigated either through the implementation of the recommended solution proposed by Hashlock or through an alternative client-provided solution that demonstrably addresses the issue.
Acknowledged	The client has formally recognized the vulnerability but has chosen not to address it due to the high cost or complexity of remediation. This status is acceptable for medium and low-severity findings after internal review and agreement. However, all high-severity findings must be resolved without exception.
Unresolved	The finding remains neither remediated nor formally acknowledged by the client, leaving the vulnerability unaddressed.

Audit Findings

Medium

[M-01] RocketNodeStaking#slashRPL - Minimum stake check blocks all slashing

Description

The slashing path enforces that legacy RPL stake must remain above the minimum even after the slash. If a node keeps only the minimum legacy stake, any slashing attempt reverts and no penalty is applied. This allows a minimally staked node to become effectively slash-immune, undermining slashing as an economic deterrent.

Vulnerability Details

`slashRPL` computes `rplSlashAmount`, sends tokens to the auction manager, and then calls `_decreaseNodeLegacyRPLStake`. That helper requires `legacyStakedRPL >= _amount + minimumLegacyStakedRPL`. When `legacyStakedRPL` equals the minimum, any positive `_amount` fails the require, so slashing always reverts. The code caps `_amount` to the current stake but never allows reducing below the minimum, so nodes at the floor cannot be penalized.

```
function slashRPL(address _nodeAddress, uint256 _ethSlashAmount) override external
onlyRegisteredMinipool(msg.sender) {
    uint256 rplSlashAmount = calcBase * _ethSlashAmount /
rocketNetworkPrices.getRPLPrice();
    uint256 rplStake = getNodeLegacyStakedRPL(_nodeAddress);
    if (rplSlashAmount > rplStake) { rplSlashAmount = rplStake; }
    if (rplSlashAmount > 0) rocketVault.transferToken("rocketAuctionManager",
IERC20(getContractAddress("rocketTokenRPL")), rplSlashAmount);
    _decreaseNodeLegacyRPLStake(_nodeAddress, rplSlashAmount); // enforces post-slash >=
minimum
}
function _decreaseNodeLegacyRPLStake(address _nodeAddress, uint256 _amount) internal {
    uint256 legacyStakedRPL = getUint(legacyKey);
```

```

uint256 minimumLegacyStakedRPL = getNodeMinimumLegacyRPLStake(_nodeAddress);
    require(legacyStakedRPL >= _amount + minimumLegacyStakedRPL, "Insufficient legacy
staked RPL");
    ...
}

```

Impact

A node that stakes only the minimum legacy RPL becomes immune to slashing: any attempt to penalize misbehavior reverts, preventing economic punishment and weakening protocol safety assumptions around collateral-backed behavior.

Recommendation

Allow slashing to bypass the minimum requirement (or reduce below and flag the node). For example, add a slashing-specific path that skips the minimum check or adjusts the minimum check to only apply to voluntary reductions:

```

function slashRPL(address _nodeAddress, uint256 _ethSlashAmount) override external
onlyRegisteredMinipool(msg.sender) {
    ...
-   _decreaseNodeLegacyRPLStake(_nodeAddress, rplSlashAmount);
+   _decreaseNodeLegacyRPLStake(_nodeAddress, rplSlashAmount, true); }
- function _decreaseNodeLegacyRPLStake(address _nodeAddress, uint256 _amount) internal {
+ function _decreaseNodeLegacyRPLStake(address _nodeAddress, uint256 _amount, bool
isSlash) internal {
    uint256 legacyStakedRPL = getUint(legacyKey);
    uint256 minimumLegacyStakedRPL = getNodeMinimumLegacyRPLStake(_nodeAddress);
-   require(legacyStakedRPL >= _amount + minimumLegacyStakedRPL, "Insufficient legacy
staked RPL");
+   if (!isSlash) {
+       require(legacyStakedRPL >= _amount + minimumLegacyStakedRPL, "Insufficient
legacy staked RPL");
+   } else {
+       require(legacyStakedRPL >= _amount, "Insufficient legacy staked RPL");
+   } ... }

```

Status

Resolved



Hashlock Pty Ltd

Centralisation

The Rocket Pool protocol is architected with decentralisation as a foundational principle, supported by DAO-based governance mechanisms that oversee protocol parameters and evolution. These governance structures are designed to maintain security and operational robustness while aligning with the protocol's decentralised ethos.



Centralised

Decentralised

Conclusion

After Hashlock's analysis, the RocketPool project seems to have a sound and well-tested code base, now that our vulnerability findings have been resolved. Overall, most of the code is correctly ordered and follows industry best practices. The code is well commented on as well. To the best of our ability, Hashlock is not able to identify any further vulnerabilities.

Our Methodology

Hashlock strives to maintain a transparent working process and to make our audits a collaborative effort. The objective of our security audits is to improve the quality of systems and upcoming projects we review and to aim for sufficient remediation to help protect users and project leaders. Below is the methodology we use in our security audit process.

Manual Code Review:

In manually analysing all of the code, we seek to find any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behaviour when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our methodologies include manual code analysis, user interface interaction, and white box penetration testing. We consider the project's website, specifications, and whitepaper (if available) to attain a high-level understanding of what functionality the smart contract under review contains. We then communicate with the developers and founders to gain insight into their vision for the project. We install and deploy the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We undergo a robust, transparent process for analysing potential security vulnerabilities and seeing them through to successful remediation. When a potential issue is discovered, we immediately create an issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is vast because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, and then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyse the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinised by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the contract details are made public.

Disclaimers

Hashlock's Disclaimer

Hashlock's team has analysed these smart contracts in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in the smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Hashlock is not responsible for the safety of any funds and is not in any way liable for the security of the project.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to attacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.

About Hashlock

Hashlock is an Australian-based company aiming to help facilitate the successful widespread adoption of distributed ledger technology. Our key services all have a focus on security, as well as projects that focus on streamlined adoption in the business sector.

Hashlock is excited to continue to grow its partnerships with developers and other web3-oriented companies to collaborate on secure innovation, helping businesses and decentralised entities alike.

Website: hashlock.com.au

Contact: info@hashlock.com.au



#hashlock.