



**Unicrypt.Network**  
**Presale Contracts**  
**SMART CONTRACT AUDIT**  
**27.01.2021**

**Made in Germany by Chainsulting.de**



## Table of contents

1. Disclaimer.....	3
2. About the Project and Company .....	4
2.1 Project Overview.....	5
3. Vulnerability & Risk Level .....	6
4. Auditing Strategy and Techniques Applied.....	7
4.1 Methodology .....	7
4.2 Used Code from other Frameworks/Smart Contracts .....	8
4.3 Tested Contract Files .....	9
4.4 Metrics / CallGraph.....	10
4.5 Metrics / Source Lines .....	11
4.6 Metrics / Capabilities .....	12
4.7 Metrics / Source Unites in Scope.....	13
5. Scope of Work.....	14
5.1 Manual and Automated Vulnerability Test.....	15
5.1.1 Overpowered owner.....	15
5.1.2 No process/function in place to prevent malicious actor to use the presale contract .....	16
5.2. SWC Attacks & Special Checks.....	17
7. Verify Claims.....	21
8. Executive Summary.....	24
9. Deployed Smart Contract .....	25

## 1. Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of Unicrypt.network. If you are not the intended receptor of this document, remember that any disclosure, copying or dissemination of it is forbidden.

Major Versions / Date	Description
0.1 (23.01.2021)	Layout
0.5 (25.01.2021)	Automated Security Testing Manual Security Testing
0.8 (25.01.2021)	Testing SWC Checks
0.9 (26.01.2021)	Summary and Recommendation
1.0 (27.01.2021)	Final document

## 2. About the Project and Company

Company address: NA (ANON)

Website: <https://unicrypt.network/>

GitHub: NA

Twitter: [https://twitter.com/UNCX\\_token](https://twitter.com/UNCX_token)

Telegram: [https://t.me/uncx\\_token](https://t.me/uncx_token)

Etherscan (UNCX Token): <https://etherscan.io/token/0xaDB2437e6F65682B85F814fBc12FeC0508A7B1D0>

Medium: <https://unicrypt.medium.com/>

## 2.1 Project Overview

The Unicrypt platform allows yield farming virtually any ERC20 token. It provides safe vault contracts for other tokens to deposit the farm rewards into, and a dApp that's targeted for mobile and desktop use with connections to all major wallets for users to farm their favourite tokens on. Unicrypt is one of the major platforms for Proof of liquidity, which helps users find new pairs on Uniswap that have locked their liquidity (Uniswap LP Token). This means it is impossible for that liquidity to be pulled until the unlock date expires. For taking part in this program, tokens are awarded a trust score, and are highly visible to investors searching on the platform for new tokens.

Any project can run their presale on the UniCrypt Launchpad in the same way as any token is free to list on Uniswap, essentially turning the UniCrypt platform into Uniswap for pre-sales. The platform intends to give investors an alternative to centralized launchpads such as Binance pad, Swap pad, Unilayer pad, and Ddim pad, where new projects must apply and wait for acceptance.

The revolutionary UniCrypt platform closes the final safety gap between presales and market initialization. The next update on the Unicrypt pad after the pre-sales will introduce whitelisted auditors who contribute and flag tokens as 'audited.'

Tokens/contracts that appear malicious will be visible within the presale itself. Furthermore, malicious minting functions, proxies, or blacklists will be quickly flagged and alerted to users in a decentralized way.

### 3. Vulnerability & Risk Level

Risk represents the probability that a certain source-threat will exploit vulnerability, and the impact of that event on the organization or system. Risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 – 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon as possible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 – 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

## 4. Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. To do so, reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as there were discovered.

### 4.1 Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
  - i. Review of the specifications, sources, and instructions provided to Chainsulting to make sure we understand the size, scope, and functionality of the smart contract.
  - ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
  - iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Chainsulting describe.
2. Testing and automated analysis that includes the following:
  - i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
  - ii. Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.

## 4.2 Used Code from other Frameworks/Smart Contracts (direct imports)

1. SafeMath.sol (0.6.0)

<https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/math/SafeMath.sol>

2. IERC20.sol (0.6.0)

<https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/IERC20.sol>

3. EnumerableSet.sol (0.6.0)

<https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/EnumerableSet.sol>

4. Ownable.sol (0.6.0)

<https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable.sol>

5. ReentrancyGuard.sol (0.6.0)

<https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/ReentrancyGuard.sol>

6. Context.sol (0.6.0)

<https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/GSN/Context.sol>

7. TransferHelper.sol

<https://github.com/Uniswap/uniswap-lib/blob/master/contracts/libraries/TransferHelper.sol>

8. UniswapV2Locker.sol

Audited Contract (Verified)

<https://docs.unicrypt.network/contracts/locker/univ2>

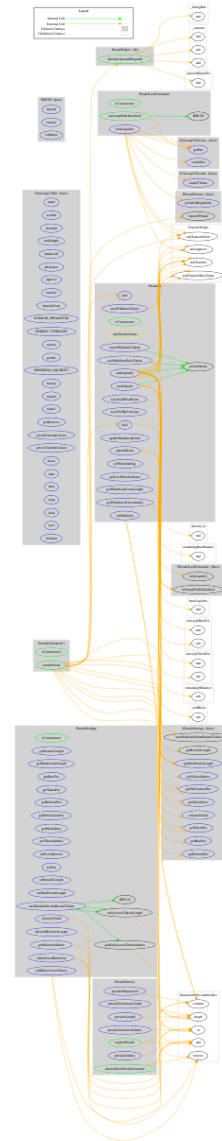


## 4.3 Tested Contract Files

The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review

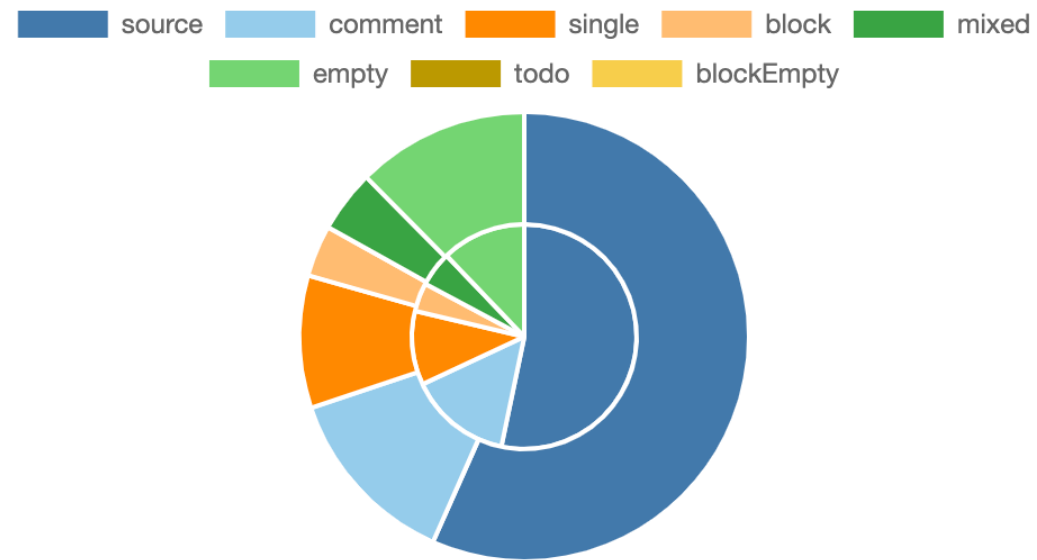
File	Fingerprint (SHA256)
Presale01.sol	D34951A10A88C0357E8C127854D90836
PresaleFactory.sol	58A05A09B4513209EDA3B2CADFD982C7
PresaleGenerator01.sol	5042D9027D2D2C451739E15B004EA55D
PresaleHelper.sol	20333D00A175128DC7C85C8002712398
PresaleLockForwarder.sol	8B406297A68872A1584481673B08AD41
PresaleSettings.sol	1C830AB321B10CAFFAC3563ED9A5AE66

## 4.4 Metrics / CallGraph













View Full Report: [https://chainsulting.de/wp-content/uploads/2021/01/solidity-metrics\\_unicrypt\\_presale.html](https://chainsulting.de/wp-content/uploads/2021/01/solidity-metrics_unicrypt_presale.html)












## 4.5 Metrics / Source Lines



## 4.6 Metrics / Capabilities

Solidity Versions observed		 Experimental Features	 Can Receive Funds	 Uses Assembly	 Has Destroyable Contracts
0.6.12			yes	**** (0 asm blocks)	
 Transfers ETH	 Low-Level Calls	 DelegateCall	 Uses Hash Functions	 ECTRecover	 New/Create/Create2
yes					yes → NewContract:Presale01

## 4.7 Metrics / Source Unites in Scope

Type	File	Logic Contract s	Interfaces	Line s	nLine s	nSLO C	Comme nt Lines	Comple x. Score	Capabilities
	PresaleLockForwarder.sol	1	4	128	70	47	11	123	
	PresaleGenerator01.sol	1	2	99	86	62	12	69	
	PresaleSettings.sol	1	_____	150	150	114	17	78	_____
	Presale01.sol	1	4	395	349	257	83	258	
	PresaleHelper.sol	1	_____	19	19	13	2	13	_____
	PresaleFactory.sol	1	_____	74	74	37	22	33	_____
	<b>Totals</b>	<b>6</b>	<b>10</b>	<b>865</b>	<b>748</b>	<b>530</b>	<b>147</b>	<b>574</b>	

## 5. Scope of Work

The Unicrypt team provided us with the files that needs to be tested. The scope of the audit are the Presale contracts.

Following contracts with the direct imports been tested

Presale01.sol

PresaleFactory.sol

PresaleGenerator01.sol

PresaleHelper.sol

PresaleLockerForwarder.sol

PresaleSettings.sol

The team put forward the following assumptions regarding the security, usage of the contracts:

- Presale investors are able to receive the sale token (S\_TOKEN) after the buying process
- The presale project is able to withdraw the base token (B\_TOKEN) after the presale successfully ends.
- If the presale funding goal is not met, investors are able to withdraw funds
- If the presale was successful, it creates a Uniswap pair and locks liquidity on Unicrypt
- Checking the overall security of the contracts

The main goal of this audit was to verify these claims. The auditors can provide additional feedback on the code upon the client's request.

## 5.1 Manual and Automated Vulnerability Test

### CRITICAL ISSUES

During the audit, Chainsulting's experts found **no Critical issues** in the code of the smart contract.

### HIGH ISSUES

During the audit, Chainsulting's experts found **no High issues** in the code of the smart contract.

### MEDIUM ISSUES

During the audit, Chainsulting's experts found **no Medium issues** in the code of the smart contract.

### LOW ISSUES

#### 5.1.1 Overpowered owner

Severity: LOW

Status: **ADDRESSED**

File(s) affected: All

Attack / Description	Code Snippet	Result/Recommendation
This function is callable only from one address. Therefore, the presale / funds depends heavily on this address. In this case, there are scenarios that may lead to undesirable consequences for investors or project owner, e.g. if the private key of this address becomes compromised.	<pre>PRESALE_OWNER</pre>	We recommend thinking about possible functions to give project owner (presale creators) the possibility (Frontend) to setup MultiSig wallet for that address. (Educate) Sample: <a href="https://gnosis-safe.io">https://gnosis-safe.io</a>

## INFORMATIONAL ISSUES

5.1.2 No process/function in place to prevent malicious actor to use the presale contract

Severity: INFORMATIONAL









Status: **ADDRESSED**

File(s) affected: All

Attack / Description	Code Snippet	Result/Recommendation
Potential usage of the presale contract by malicious actors, that are able to use the secure and trusted environment to scam investors. Or as seen on Uniswap creating misleading imposter/ fake token of other projects.	NA	We recommend including enough disclaimer for investors and maybe use a second layer together with the governance token to verify presales by a counsel or group of researchers.







## 5.2. SWC Attacks & Special Checks

ID	Title	Relationships	Test Result
<a href="#">SWC-131</a>	Presence of unused variables	<a href="#">CWE-1164: Irrelevant Code</a>	
<a href="#">SWC-130</a>	Right-To-Left-Override control character (U+202E)	<a href="#">CWE-451: User Interface (UI) Misrepresentation of Critical Information</a>	
<a href="#">SWC-129</a>	Typographical Error	<a href="#">CWE-480: Use of Incorrect Operator</a>	
<a href="#">SWC-128</a>	DoS With Block Gas Limit	<a href="#">CWE-400: Uncontrolled Resource Consumption</a>	
<a href="#">SWC-127</a>	Arbitrary Jump with Function Type Variable	<a href="#">CWE-695: Use of Low-Level Functionality</a>	
<a href="#">SWC-125</a>	Incorrect Inheritance Order	<a href="#">CWE-696: Incorrect Behavior Order</a>	
<a href="#">SWC-124</a>	Write to Arbitrary Storage Location	<a href="#">CWE-123: Write-what-where Condition</a>	
<a href="#">SWC-123</a>	Requirement Violation	<a href="#">CWE-573: Improper Following of Specification by Caller</a>	

ID	Title	Relationships	Test Result
<a href="#">SWC-122</a>	Lack of Proper Signature Verification	<a href="#">CWE-345: Insufficient Verification of Data Authenticity</a>	✓
<a href="#">SWC-121</a>	Missing Protection against Signature Replay Attacks	<a href="#">CWE-347: Improper Verification of Cryptographic Signature</a>	✓
<a href="#">SWC-120</a>	Weak Sources of Randomness from Chain Attributes	<a href="#">CWE-330: Use of Insufficiently Random Values</a>	✓
<a href="#">SWC-119</a>	Shadowing State Variables	<a href="#">CWE-710: Improper Adherence to Coding Standards</a>	✓
<a href="#">SWC-118</a>	Incorrect Constructor Name	<a href="#">CWE-665: Improper Initialization</a>	✓
<a href="#">SWC-117</a>	Signature Malleability	<a href="#">CWE-347: Improper Verification of Cryptographic Signature</a>	✓
<a href="#">SWC-116</a>	Timestamp Dependence	<a href="#">CWE-829: Inclusion of Functionality from Untrusted Control Sphere</a>	✓
<a href="#">SWC-115</a>	Authorization through tx.origin	<a href="#">CWE-477: Use of Obsolete Function</a>	✓
<a href="#">SWC-114</a>	Transaction Order Dependence	<a href="#">CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')</a>	✓

ID	Title	Relationships	Test Result
<a href="#">SWC-113</a>	DoS with Failed Call	<a href="#">CWE-703: Improper Check or Handling of Exceptional Conditions</a>	✓
<a href="#">SWC-112</a>	Delegatecall to Untrusted Callee	<a href="#">CWE-829: Inclusion of Functionality from Untrusted Control Sphere</a>	✓
<a href="#">SWC-111</a>	Use of Deprecated Solidity Functions	<a href="#">CWE-477: Use of Obsolete Function</a>	✓
<a href="#">SWC-110</a>	Assert Violation	<a href="#">CWE-670: Always-Incorrect Control Flow Implementation</a>	✓
<a href="#">SWC-109</a>	Uninitialized Storage Pointer	<a href="#">CWE-824: Access of Uninitialized Pointer</a>	✓
<a href="#">SWC-108</a>	State Variable Default Visibility	<a href="#">CWE-710: Improper Adherence to Coding Standards</a>	✓
<a href="#">SWC-107</a>	Reentrancy	<a href="#">CWE-841: Improper Enforcement of Behavioral Workflow</a>	✓
<a href="#">SWC-106</a>	Unprotected SELFDESTRUCT Instruction	<a href="#">CWE-284: Improper Access Control</a>	✓
<a href="#">SWC-105</a>	Unprotected Ether Withdrawal	<a href="#">CWE-284: Improper Access Control</a>	✓
<a href="#">SWC-104</a>	Unchecked Call Return Value	<a href="#">CWE-252: Unchecked Return Value</a>	✓

ID	Title	Relationships	Test Result
<a href="#">SWC-103</a>	Floating Pragma	<a href="#">CWE-664: Improper Control of a Resource Through its Lifetime</a>	
<a href="#">SWC-102</a>	Outdated Compiler Version	<a href="#">CWE-937: Using Components with Known Vulnerabilities</a>	
<a href="#">SWC-101</a>	Integer Overflow and Underflow	<a href="#">CWE-682: Incorrect Calculation</a>	
<a href="#">SWC-100</a>	Function Default Visibility	<a href="#">CWE-710: Improper Adherence to Coding Standards</a>	

## 7. Verify Claims

### 7.1 Presale investors are able to receive the sale token (S\_TOKEN) after the buying process

**Status:** tested and verified

**Code:** Ln 244 – 253 Presale01.sol (Generated for each presale)

```
// withdraw presale tokens
// percentile withdrawls allows fee on transfer or rebasing tokens to still work
function userWithdrawTokens () external nonReentrant {
    require(STATUS.LP_GENERATION_COMPLETE, 'AWAITING LP GENERATION');
    BuyerInfo storage buyer = BUYERS[msg.sender];
    uint256 tokensRemainingDenominator = STATUS.TOTAL_TOKENS_SOLD.sub(STATUS.TOTAL_TOKENS_WITHDRAWN);
    uint256 tokensOwed = PRESALE_INFO.S_TOKEN.balanceOf(address(this)).mul(buyer.tokensOwed).div(tokensRemainingDenominator);
    require(tokensOwed > 0, 'NOTHING TO WITHDRAW');
    STATUS.TOTAL_TOKENS_WITHDRAWN = STATUS.TOTAL_TOKENS_WITHDRAWN.add(buyer.tokensOwed);
    buyer.tokensOwed = 0;
    TransferHelper.safeTransfer(address(PRESALE_INFO.S_TOKEN), msg.sender, tokensOwed);
}
```

### 7.2 The presale owner is able to withdraw the remaining base token (B\_TOKEN) after the presale successfully ends.

**Status:** tested and verified

**Code:** Ln 346 – 350 Presale01.sol (Generated for each presale)

```
// send remaining base tokens to presale owner
uint256 remainingBaseBalance = PRESALE_INFO.PRESALE_IN_ETH ? address(this).balance :
PRESALE_INFO.B_TOKEN.balanceOf(address(this));
TransferHelper.safeTransferBaseToken(address(PRESALE_INFO.B_TOKEN), PRESALE_INFO.PRESALE_OWNER, remainingBaseBalance,
!PRESALE_INFO.PRESALE_IN_ETH);

STATUS.LP_GENERATION_COMPLETE = true;
}
```

### 7.3 If the presale funding goal is not met, investors are able to withdraw funds

**Status:** tested and verified

**Code:** Ln 257 - 267 Presale01.sol (Generated for each presale)

```
// on presale failure
// percentile withdrawls allows fee on transfer or rebasing tokens to still work
function userWithdrawBaseTokens () external nonReentrant {
    require(presaleStatus() == 3, 'NOT FAILED'); // FAILED
    BuyerInfo storage buyer = BUYERS[msg.sender];
    uint256 baseRemainingDenominator = STATUS.TOTAL_BASE_COLLECTED.sub(STATUS.TOTAL_BASE_WITHDRAWN);
    uint256 remainingBaseBalance = PRESALE_INFO.PRESALE_IN_ETH ? address(this).balance :
PRESALE_INFO.B_TOKEN.balanceOf(address(this));
    uint256 tokensOwed = remainingBaseBalance.mul(buyer.baseDeposited).div(baseRemainingDenominator);
    require(tokensOwed > 0, 'NOTHING TO WITHDRAW');
    STATUS.TOTAL_BASE_WITHDRAWN = STATUS.TOTAL_BASE_WITHDRAWN.add(buyer.baseDeposited);
    buyer.baseDeposited = 0;
    TransferHelper.safeTransferBaseToken(address(PRESALE_INFO.B_TOKEN), msg.sender, tokensOwed, !PRESALE_INFO.PRESALE_IN_ETH);
}
```

#### 7.4 If the presale was successful, it creates a Uniswap pair and locks liquidity on Unicrypt

**Status:** tested and verified

**Code:** Ln 298 - 350 Presale01.sol (Generated for each presale) and using PresaleLockForwarder.sol

```
// on presale success, this is the final step to end the presale, lock liquidity and enable withdrawls of the sale token.
// This function does not use percentile distribution. Rebasing mechanisms, fee on transfers, or any deflationary logic
// are not taken into account at this stage to ensure stated liquidity is locked and the pool is initialised according to
// the presale parameters and fixed prices.
function addLiquidity() external nonReentrant {

**
**
// sale token liquidity
    uint256 tokenLiquidity = baseLiquidity.mul(PRESALE_INFO.LISTING_RATE).div(10 ** uint256(PRESALE_INFO.B_TOKEN.decimals()));
    TransferHelper.safeApprove(address(PRESALE_INFO.S_TOKEN), address(PRESALE_LOCK_FORWARDER), tokenLiquidity);

    PRESALE_LOCK_FORWARDER.lockLiquidity(PRESALE_INFO.B_TOKEN, PRESALE_INFO.S_TOKEN, baseLiquidity, tokenLiquidity, block.timestamp
+ PRESALE_INFO.LOCK_PERIOD, PRESALE_INFO.PRESALE_OWNER);
**
**
    STATUS.LP_GENERATION_COMPLETE = true;
```

#### 7.5 Checking the overall security of the contracts

## 8. Executive Summary

The overall code quality of the project is very good, not overloaded with unnecessary functions, these is greatly benefiting the security of the contract. It correctly implemented widely-used and reviewed contracts from OpenZeppelin and for safe mathematical operations.

The main goal of the audit was to verify the claims regarding the security of the smart contract and the functions. During the audit, no critical or minor issues were found after the manual and automated security testing. The previous audits with our team greatly benefiting the outcome, as the development team adopted the findings in the previous reports. The only concern that we have is the usage of the presale contract by malicious actors, that are able to use the secure and trusted environment to scam investors. Or as seen on Uniswap creating misleading imposter/ fake token of other projects. We recommend including enough disclaimer for investors and maybe use a second layer together with the governance token to verify presales by a counsel or group of researchers.



## 9. Deployed Smart Contract

PENDING

Presale Settings

0x00

Presale Factory

0x00

Presale Lock Forwarder

0x00

Presale Generator

0x00

