

PUBLIC

# Security Audit

## of PAXOS's Smart Contracts

September 13, 2018

Produced for



by



# Table Of Content

Foreword	1
Executive Summary	1
Audit Overview	2
1. Scope of the Audit	2
2. Depth of Audit	2
3. Terminology	2
Limitations	4
System Overview	5
1. Token Overview	5
Best Practices in PAXOS's project	6
1. Hard Requirements	6
2. Soft Requirements	6
Security Issues	7
1. totalSupply_ might include lost tokens  	7
2. Locked ETH and Locked Tokens  	7
3. Tokens can be minted in paused state  	7
4. Remarks on Upgradeability 	8
5. Unsafe Dependence On Block Information 	8
6. Unsafe Dependence On Block Gas 	8
7. Use Of Origin 	8
8. Gas-dependent Reentrancy 	8
9. Reentrant method call 	8
10. Transaction Order Affects Execution of Ether Transfer 	8
11. Unhandled Exception 	8

Design Issues . . . . .	9
1. Visibility is not explicit   . . . . .	9
2. Old SafeMath version   . . . . .	9
3. Supply can be moved by zero   . . . . .	9
Recommendations / Suggestions . . . . .	10
Disclaimer . . . . .	11

# Foreword

We first and foremost thank PAXOS for giving us the opportunity to audit their smart contracts. This documents outlines our methodology, limitations, and results.

– ChainSecurity

## Executive Summary

The PAXOS Standard smart contracts have been analyzed under different aspects, with a variety of tools for automated security analysis of Ethereum smart contracts and manual expert review.

Overall, we found that PAXOS employs very good coding practices and has clean, well-documented code which is well covered by a corresponding test suite. Nonetheless, CHAINSECURITY was able to uncover several minor issues which PAXOS successfully resolved.

# Audit Overview

## Scope of the Audit

The scope of the audit is limited to the following source code files. All of these source code files were initially received on August 1, 2018 and updated for review on September 12, 2018<sup>1</sup>:

File	SHA-256 checksum
Proxy.sol	cc8e66b5acf1b9981729ab26c85475bdfad8dcae861d70eb2559b4dc7ddb859a
UpgradeabilityProxy.sol	7db89b037ee71fea81631d79a9bc779dd34d440e23991e9ba9ffd24de2ba8104
AdminUpgradeabilityProxy.sol	738b38c385591763d5ce1a747b777c6dbdb27acc44601fa9d0d0908fbb087091
PAXImplementation.sol	7c29c59e43ed56432c5e5108f6d3e6d99f60b6f01201e72182ef91318772a96d

## Depth of Audit

The scope of the security audit conducted by CHAINSECURITY was restricted to:

- Scan the contracts listed above for generic security issues using automated systems and manually inspect the results.
- Manual audit of the contracts listed above for security issues.

## Terminology





For the purpose of this audit, we adopt the following terminology. For security vulnerabilities, we specify the *likelihood*, *impact* and *severity* (inspired by the OWASP risk rating methodology<sup>2</sup>).

**Likelihood** represents the likelihood of a security vulnerability to be encountered or exploited in the wild.










**Impact** specifies the technical and business related consequences of an exploit.

**Severity** is derived based on the likelihood and the impact calculated previously.

We categorize the findings into 4 distinct categories, depending on their severities:

-  Low: can be considered as less important
-  Medium: should be fixed
-  High: we strongly suggest to fix it before release
-  Critical: needs to be fixed before release

These severities are derived from the likelihood and the impact using the following table, following a standard approach in risk assessment.

LIKELIHOOD	IMPACT		
	High	Medium	Low
High			
Medium			
Low			

<sup>1</sup><https://github.com/paxosglobal/pax-contracts/tree/042ad92fa27876b37149d4315a04b84dc813cb8c>

<sup>2</sup>[https://www.owasp.org/index.php/OWASP\\_Risk\\_Rating\\_Methodology](https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology)

During the audit concerns might arise or tools might flag certain security issues. If our careful inspection reveals no security impact, we label it as **✓ No Issue**. If during the course of the audit process, an issue has been addressed technically, we label it as **✓ Fixed**, while if it has been addressed otherwise by improving documentation or further specification, we label it as **✓ Addressed**. Finally, if an issue is meant to be fixed in the future without immediate changes to the code, we label it as **✓ Acknowledged**.

Findings that are labelled as either **✓ Fixed** or **✓ Addressed** are resolved and therefore pose no security threat. Their severity is still listed, but just to give the reader a quick overview what kind of issues were found during the audit.

# Limitations

Security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a secure smart contract. However, auditing allows to discover vulnerabilities that were overlooked during development and areas where additional security measures are necessary.

In most cases, applications are either fully protected against a certain type of attack, or they lack protection against it completely. Some of the issues may affect the entire smart contract application, while some lack protection only in certain areas. We therefore carry out a source code review trying to determine all locations that need to be fixed. Within the customer-determined timeframe, CHAINSECURITY has performed auditing in order to discover as many vulnerabilities as possible.

# System Overview

Token Name & Symbol	PAXOS STANDARD, PAX
Decimals	18 decimals
Exchange Rate	1 USD = 1 PAXs
Refund	None
Token Type	ERC20
Token Generation	Mintable, Burnable
Pausable	Yes
KYC	None/Off-chain

Table 1: Facts about the PAX token.

In the following we describe the PAXOS STANDARD (PAX). The table above gives the general overview.

## Token Overview

The smart contract system consists of the ERC20 PAXOS STANDARD which is supposed to represent a US Dollar in PAXOS' reserve and a set of contracts from the ZeppelinOS library. These ensure that the contract logic can be upgraded in the future.

Apart from the usual ERC20 functionality, it is important to acknowledge the roles and their capabilities encoded in the PAXOS STANDARD contract:

- **Owner**  
Can pause trading and assign the `supplyController` and `lawEnforcementRole`. Thereby, the owner also possesses all powers of these two roles.
- **supplyController**  
Can increase or decrease the token supply and transfer its own role.
- **lawEnforcementRole**  
Can freeze, unfreeze and wipe an address, meaning that the target will not be able to move or will lose all of its funds. This role is also transferable.
- **Upgradeability Admin**  
Can exchange the actual token implementation. Thereby, the most powerful role.



# Best Practices in PAXOS's project

Projects of good quality follow best practices. In doing so, they make audits more meaningful, by allowing efforts to be focused on subtle and project-specific issues rather than the fulfillment of general guidelines.

Avoiding code duplication is a good example of a good engineering practice which increases the potential of any security audit.

We now list a few points that should be enforced in any good project that aims to be deployed on the Ethereum blockchain. The corresponding box is ticked when PAXOS's project fitted the criterion when the audit started.

## Hard Requirements

These requirements ensure that the PAXOS's project can be audited by CHAINSECURITY.

- The code is provided as a Git repository to allow the review of future code changes.
- Code duplication is minimal, or justified and documented.
- Libraries are properly referred to as package dependencies, including the specific version(s) that are compatible with PAXOS's project. No library file is mixed with PAXOS's own files.
- The code compiles with the latest Solidity compiler version. If PAXOS uses an older version, the reasons are documented.
- There are no compiler warnings, or warnings are documented.

## Soft Requirements

Although these requirements are not as important as the previous ones, they still help to make the audit more valuable to PAXOS.

- There are migration scripts.
- There are tests.
- The tests are related to the migration scripts and a clear separation is made between the two.
- The tests are easy to run for CHAINSECURITY, using the documentation provided by PAXOS.
- The test coverage is available or can be obtained easily.
- The output of the build process (including possible flattened files) is not committed to the Git repository.
- The project only contains audit-related files, or, if not possible, a meaningful separation is made between modules that have to be audited and modules that CHAINSECURITY should assume correct and out of scope.
- There is no dead code.
- The code is well documented.
- The high-level specification is thorough and allow a quick understanding of the project without looking at the code.
- Both the code documentation and the high-level specification are up to date with respect to the code version CHAINSECURITY audits.
- There are no getter functions for public variables, or the reason why these getters are in the code is given.
- Function are grouped together according either to the Solidity guidelines<sup>3</sup>, or to their functionality.

<sup>3</sup><https://solidity.readthedocs.io/en/latest/style-guide.html#order-of-functions>

# Security Issues

In the following, we discuss our investigation into security issues. Therefore, we highlight whenever we found specific issues but also mention what vulnerability classes do not appear, if relevant.

## totalSupply\_ might include lost tokens M ✓ Addressed

Given that exact control over the `totalSupply_` is important for PAXOS's business case, we note the following: PAXOS (as the contract owner) can reduce the `totalSupply_` using `decreaseSupply`. However, this function does not allow a correction in case a certain amount of tokens is lost, e.g. due to an incorrectly executed transfer or lost private key.

In these cases the `lawEnforcementRole` would have to be used in order to call `wipeFrozenAddress`. This complicates the correction of the `totalSupply_` and might lead to temporary inconsistencies.

**Likelihood:** High

**Impact:** Low

**Fixed:** PAXOS remarks that the total supply will inevitably include lost tokens so long as the risk of losing private keys exists. It will be PAXOS's policy to keep the US dollars in the reserve account in the case of allegedly lost tokens and to neither freeze nor wipe wallets unless otherwise required to by court order or similar legal directive. Therefore, PAXOS does not foresee there being temporary inconsistencies between the number of tokens and the US dollars held in the reserve account due to lost tokens.

## Locked ETH and Locked Tokens L ✓ Addressed

In the current version of the contracts, ETH and ERC-20 tokens can be sent to the contracts. First, both contracts can receive any ERC-20 tokens. Numerous historic cases have shown that accidental ERC-20 transfers to token contracts occur frequently. Currently, such ERC-20 tokens would be locked.

The proxied contract can also receive ETH through a generic fallback function:

```
15 function() public payable external {  
                                     Proxy.sol
```

In the current design received ETH will lead to a **revert** as none of the functions inside `PAXImplementation` is **payable**. ETH which is received indirectly (e.g. through `selfdestruct`) would be locked. Additionally, future implementations that contain **payable** functions, need to ensure that they contain capabilities to forward or spend such funds.

**Likelihood:** Medium

**Impact:** Low

**Fixed:** PAXOS responded that the most likely case for locked funds is that ETH or ERC-20 tokens are sent to the proxy address, which is the address that is passed around throughout the network, representing the token. Because the Proxy's logic layer is upgradeable, PAXOS could always add token recovery capabilities through an upgrade to recover locked funds at the proxy address, if it occurs, allowing Paxos to tailor the methods to whatever recovery process is appropriate.

## Tokens can be minted in paused state M ✓ Addressed

The PAXOS STANDARD can be paused by the owner of the contract which is supposed to trigger a stopped state. This state is enforced by the `whenNotPaused` modifier which guards all token transfers and blocks them if `paused` is `true`. However, the token can still be minted and transferred to the `supplyController` address through the `increaseSupply` function. Since the `supplyController` is a separate role that can be different from the owner, it can circumvent the paused state the owner wanted to impose.

**Likelihood:** Medium

**Impact:** Medium

**Fixed:** PAXOS elaborates that the owner can change the supply controller, meaning that in the case a supply controller adjusted supply against the wishes of the owner, the owner could seize the supply control role. In the case of truly illegitimate supply changes, it would likely be appropriate to use the law enforcement role to correct them. PAXOS also notes that the supply controller is a trusted address in this system.

### Remarks on Upgradeability ✓ Acknowledged

PAXOS makes use of the ZeppelinOS Upgradeability with unstructured storage library<sup>4</sup>. While CHAINSECURITY found no issues in the current implementation, it is important to note some assumptions and remarks:

- Total functional upgradeability, as is given in this case, always comes with security and trust issues. In particular, an upgraded contract could contain new vulnerabilities and rearrange balances. Therefore, all statements in this report only refer to the current version and no guarantees can be given by CHAINSECURITY about future versions.
- For the given security guarantees to hold, no variable (in a current or future version) should be able to overwrite the variables `_implementation` in the `UpgradeabilityProxy` and `_admin` from the overarching `AdminUpgradeabilityProxy` contract.  
For the current version, this is the case (under the standard assumption that there are no hash collisions) as there are no unbounded arrays with arbitrary write access.
- The `_admin` role in the Upgradeability contracts is separated from the actual contract logic, meaning that the `upgradeability_admin` needs to be a separate role.

### Unsafe Dependence On Block Information ✓ No Issue

Security-sensitive operations must not depend on block information.

### Unsafe Dependence On Block Gas ✓ No Issue

Security-sensitive operations must not depend on gas-related information.

### Use Of Origin ✓ No Issue

The origin statement must not be used for authorization.

### Gas-dependent Reentrancy ✓ No Issue

Calls into external contracts that receive all remaining gas and are followed by state changes may be reentrant.

### Reentrant method call ✓ No Issue

Method calls that are followed by state changes may be reentrant.

### Transaction Order Affects Execution of Ether Transfer ✓ No Issue

Ether transfers whose execution can be manipulated by other transactions must be inspected for unintended behavior.

### Unhandled Exception ✓ No Issue

The return value of statements that may return error values must be explicitly checked.

---

<sup>4</sup><https://github.com/zeppelinos/zos/tree/master/packages/lib/contracts/upgradeability>

# Design Issues

The points listed here are general recommendations about the design and style of PAXOS's project. They highlight possible ways for PAXOS to further improve the code.

## Visibility is not explicit

Both the state variable `totalSupply_` and the mappings `balances` and `frozen` have no explicitly defined visibility and default to `internal`. The visibility should be explicitly declared as such. If it desired to have these as `public`, the corresponding getter functions should be removed.

**Fixed:** PAXOS adopted the recommendation and restricted the visibility.

## Old SafeMath version

PAXOS makes use of the `SafeMath` library. However, an older version is used. CHAINSECURITY recommends to upgrade to the newest version, which introduces `require` instead of `assert` in its internal checks.

**Fixed:** The `SafeMath` version was updated.

## Supply can be moved by zero

It is possible to increase or decrease the `totalSupply` by zero in the functions `increaseSupply` and `decreaseSupply`. Since this would have no impact on the contract state but would incur computational costs, a check can be introduced to avoid it.

**Fixed:** PAXOS clarifies that this was done for consistency with ERC-20, which explicitly requires transfers of zero to be legal. These transactions can be used to prove ownership of a key and therefore this ability has been left in.

## Recommendations / Suggestions

- Functions should be in order: constructor, fallback, external, public, internal, private. This is not the case in the PAXImplementation contract, where the constructor follows a public function.
- As discussed before, the owner can reset the lawEnforcementRole at any time. Thereby, the owner is more powerful than the lawEnforcementRole, which has to conform to regulation.
- The following code comment:

```
370 function decreaseSupply(uint256 _value) public onlySupplyController
    returns (bool success) {
371     require(_value <= balances[supplyController], "not_enough_supply");
372     // no need to require value <= totalSupply, since that would imply the
373     // sender's balance is greater than the totalSupply, which *should* be
        an assertion failure
374
375     balances[supplyController] = balances[supplyController].sub(_value);
```

PAXImplementation.sol

is not consistent with the code given that the `_value <= balances[supplyController]` is performed.

**Post-audit comment:** PAXOS has fixed some of the issues above and is aware of all the implications of those points which were not addressed. Given this awareness, PAXOS has to perform no more code changes with regards to these recommendations.

## Disclaimer

UPON REQUEST BY PAXOS, CHAINSECURITY LTD. AGREES MAKING THIS AUDIT REPORT PUBLIC. THE CONTENT OF THIS AUDIT REPORT IS PROVIDED "AS IS", WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND, AND CHAINSECURITY LTD. DISCLAIMS ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT. COPYRIGHT OF THIS REPORT REMAINS WITH CHAINSECURITY LTD..