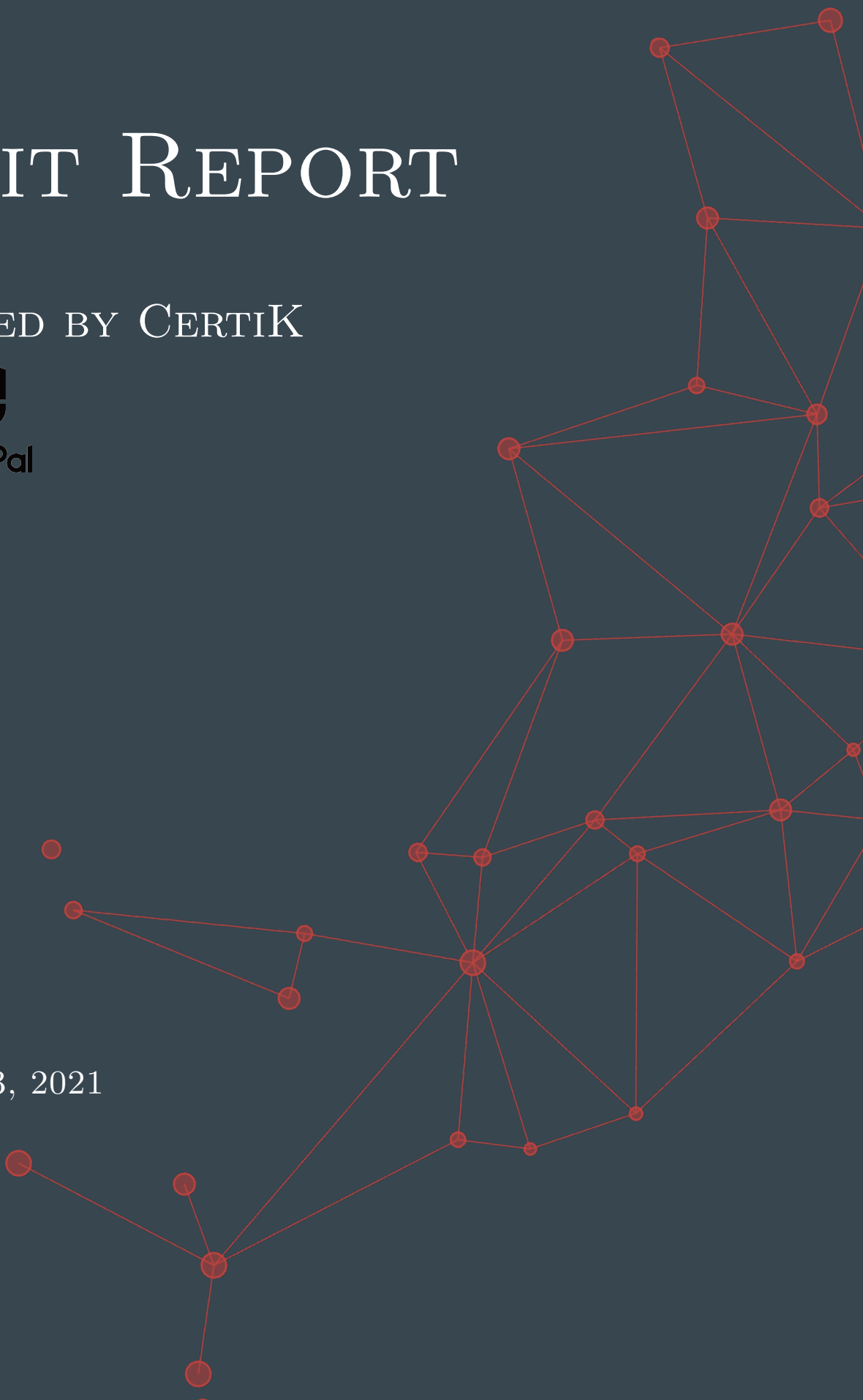
CERTIK

Audit Report

Produced by CertiK

for SafePal



February 3, 2021

CertiK Audit Report For SafePal

Request Date: 2021-02-02
Revision Date: 2021-02-03
Platform Name: Binance Smart Chain

Contents

Disclaimer

CertiK reports are not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security review.

CertiK Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has indeed completed a round of auditing with the intention to increase the quality of the company/product's IT infrastructure and or source code.

About CertiK

CertiK is a technology-led blockchain security company founded by Computer Science professors from Yale University and Columbia University built to prove the security and correctness of smart contracts and blockchain protocols.

CertiK, in partnership with grants from IBM and the Ethereum Foundation, CertiK's mission of every audit is to apply different approaches and detection methods, ranging from manual, static, and dynamic analysis, to ensure that projects are checked against known attacks and potential vulnerabilities. CertiK leverages a team of seasoned engineers and security auditors to apply testing methodologies and assessments to each project, in turn creating a more secure and robust software system.

CertiK has served more than 100 clients with high quality auditing and consulting services, ranging from stablecoins such as Binance's BGBP and Paxos Gold to decentralized oracles such as Band Protocol and Tellor. CertiK customizes its engineering tool kits, while applying cutting-edge research on smart contracts, for each client on its project to offer a high quality deliverable. For more information: https://certik.io.

Executive Summary

This report has been prepared for SafePal to discover issues and vulnerabilities in the source code of their "SafePalToken" smart contracts. A comprehensive examination has been performed, utilizing CertiK's Formal Verification Platform, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

Vulnerability Classification

CertiK categorizes issues into three buckets based on overall risk levels:

Code implementation does not match specification, which could result in the loss of funds for contract owner or users.

Code implementation does not match the specification under certain conditions, which could affect the security standard by loss of access control.

Code implementation does not follow best practices, or uses suboptimal design patterns, which could lead to security vulnerabilities further down the line.

Testing Summary

PASS

CERTIK *believes this smart contract passes security qualifications to be listed on digital asset exchanges.*

Feb 03, 2021

Type of Issues

CertiK's smart label engine applied 100% formal verification coverage on the source code. Our team of engineers has scanned the source code using proprietary static analysis tools and code-review methodologies. The following technical issues were found:

Title	Description	Issues	SWC ID
Integer Overflow/ Underflow	An overflow/underflow occurs when an arithmetic operation reaches the maximum or minimum size of a type.	0	SWC-101
Function Incorrectness	Function implementation does not meet specification, leading to intentional or unintentional vulnerabilities.	0	
Buffer Overflow	An attacker can write to arbitrary storage locations of a contract if array of out bound happens	0	SWC-124
Reentrancy	A malicious contract can call back into the calling contract before the first invocation of the function is finished.	0	SWC-107
Transaction Order Dependence	A race condition vulnerability occurs when code depends on the order of the transactions submitted to it.	0	SWC-114
Timestamp Dependence	Timestamp can be influenced by miners to some degree.	0	SWC-116
Insecure Compiler Version	Using a fixed outdated compiler version or floating pragma can be problematic if there are publicly disclosed bugs and issues that affect the current compiler version used.	1	SWC-102 SWC-103
Insecure Randomness	Using block attributes to generate random numbers is unreliable, as they can be influenced by miners to some degree.	0	SWC-120
"tx.origin" for Authorization	tx.origin should not be used for authorization. Use msg.sender instead.	0	SWC-115

Title	Description	Issues	SWC ID
Delegatecall to Untrusted Callee	Calling untrusted contracts is very dangerous, so the target and arguments provided must be sanitized.	0	SWC-112
State Variable Default Visibility	Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.	0	SWC-108
Function Default Visibility	Functions are public by default, meaning a malicious user can make unauthorized or unintended state changes if a developer forgot to set the visibility.	0	SWC-100
Uninitialized Variables	Uninitialized local storage variables can point to other unexpected storage variables in the contract.	0	SWC-109
Assertion Failure	The assert() function is meant to assert invariants. Properly functioning code should never reach a failing assert statement.	0	SWC-110
Deprecated Solidity Features	Several functions and operators in Solidity are deprecated and should not be used.	0	SWC-111
Unused Variables	Unused variables reduce code quality	0	SWC-131

Vulnerability Details

Critical

No issue found.

Medium

No issue found.

Low

No issue found.

Review Notes

Source Code SHA-256 Checksum

- **SafePalToken.sol**[1]
  `7b7c30f0ec58672fae488841ea4081b0af2af9a5db8bf8fb92fb6415d0b923c8`

Summary

CertiK team is invited by The SafePal team to audit the design and implementations of its to be released ERC20 based smart contract, and the source code has been analyzed under different perspectives and with different tools such as CertiK formal verification checkings as well as manual reviews by smart contract experts. That end-to-end process ensures proof of stability as well as a hands-on, engineering-focused process to close potential loopholes and recommend design changes in accordance with the best practices in the space. We have been actively interacting with client-side engineers when there was any potential loopholes or recommended design changes during the audit process, and SafePalToken team has been actively giving us updates for the source code and feedback about the business logics.

Meanwhile, it is recommended to have a more well-detailed document for the public to describe the source code specifications and implementations.

Overall we found the SafePalToken contract follows good practices, with reasonable amount of features on top of the ERC20 related to administrive controls by the token issuer. With the final update of source code and delivery of the audit report, we conclude that the contract is not vulnerable to any classically known antipatterns or security issues. The audit report itself is not necessarily a guarantee of correctness or trustworthiness, and we always recommend seeking multiple opinions, more test coverage and sandbox deployments before the mainnet release.

Recommendations

Items in this section are low impact to the overall aspects of the smart contracts, thus will let client to decide whether to have those reflected in the final deployed version of source codes.

SafePalToken.sol

- INFO `_setupDecimals()` – Consider removing this function from the code since it's declared with `internal` scope and never called.
- INFO `div(uint256 a, uint256 b)`: `uint` is considered non-negative, prefer `require(b != 0);`.

[1] <https://bscscan.com/address/0xd41fdb03ba84762dd66a0af1a6c8540ff1ba5dfb>

Static Analysis Results

INSECURE_COMPILER_VERSION

Line 7 in File SafePalToken.sol

```
7 pragma solidity >=0.6.0 <0.8.0;
```

Only these compiler versions are safe to compile your code: 0.7.4

Formal Verification Results

How to read

Detail for Request 1

transferFrom to same address

Verification date	20, Oct 2018
Verification timespan	395.38 ms

CERTIK *label location*: Line 30-34 in File howtoread.sol

CERTIK *label*

```
30    /*@CTK FAIL "transferFrom to same address"
31        @tag assume_completion
32        @pre from == to
33        @post __post.allowed[from][msg.sender] ==
34    */
```

Raw code location: Line 35-41 in File howtoread.sol

Raw code

```
35    function transferFrom(address from, address to
          ) {
36        balances[from] = balances[from].sub(tokens
37        allowed[from][msg.sender] = allowed[from][
38        balances[to] = balances[to].add(tokens);
39        emit Transfer(from, to, tokens);
40        return true;
41    }
```

Counterexample: This code violates the specification

Initial environment

```
1  Counter Example:
2  Before Execution:
3      Input = {
4          from = 0x0
5          to = 0x0
6          tokens = 0x6c
7      }
8      This = 0
...
53             balance: 0x0
54         }
55     }
```

Post environment

```
56
57 After Execution:
58     Input = {
59         from = 0x0
60         to = 0x0
61         tokens = 0x6c
```

Formal Verification Request 1

Context _msgSender

📅 03, Feb 2021
⏱ 4.34 ms

Line 22-25 in File SafePalToken.sol

```
22      /*@CTK "Context _msgSender"
23        @tag assume_completion
24        @post __return == msg.sender
25      */
```

Line 26-30 in File SafePalToken.sol

```
26      function _msgSender() internal view
27       returns (address payable) {
28          return msg.sender;
29      }
```

✅ The code meets the specification.

Formal Verification Request 2

Context _msgData

📅 03, Feb 2021
⏱ 5.56 ms

Line 32-35 in File SafePalToken.sol

```
32      /*@CTK "Context _msgData"
33        @tag assume_completion
34        @post __return == msg.data
35      */
```

Line 36-41 in File SafePalToken.sol

```
36      function _msgData() internal view
37       returns (bytes memory) {
38          this; // silence state mutability warning without generating
    ↪  bytecode - see https://github.com/ethereum/solidity/issues/2691
39          return msg.data;
40      }
```

✅ The code meets the specification.

Formal Verification Request 3

If method completes, integer overflow would not happen.

📅 03, Feb 2021
⏱ 16.3 ms

Line 143 in File SafePalToken.sol

```
143     //@CTK NO_OVERFLOW
```

Line 154-159 in File SafePalToken.sol

```
154     function add(uint256 a, uint256 b) internal pure returns (uint256) {
155         uint256 c = a + b;
156         require(c >= a, "SafeMath: addition overflow");
157 
158         return c;
159     }
```

✅ The code meets the specification.

Formal Verification Request 4

SafeMath add

📅 03, Feb 2021
⏱ 3.72 ms

Line 144-152 in File SafePalToken.sol

```
144     /*@CTK "SafeMath add"
145         @tag spec
146         @tag is_pure
147         @post (a + b < a || a + b < b) == __reverted
148         @post !__reverted -> __return == a + b
149         @post !__reverted -> !__has_overflow
150         @post !__reverted -> !__has_assertion_failure
151         @post !(__has_buf_overflow)
152     */
```

Line 154-159 in File SafePalToken.sol

```
154     function add(uint256 a, uint256 b) internal pure returns (uint256) {
155         uint256 c = a + b;
156         require(c >= a, "SafeMath: addition overflow");
157 
158         return c;
159     }
```

✅ The code meets the specification.

Formal Verification Request 5

If method completes, integer overflow would not happen.

📅 03, Feb 2021
⏱ 14.02 ms

Line 186 in File SafePalToken.sol

```
186     //@CTK NO_OVERFLOW
```

Line 198-203 in File SafePalToken.sol

```
198     function sub(uint256 a, uint256 b, string memory errorMessage) internal
    ↪   pure returns (uint256) {
199         require(b <= a, errorMessage);
200         uint256 c = a - b;
201 
202         return c;
203     }
```

✅ The code meets the specification.

Formal Verification Request 6

SafeMath sub

📅 03, Feb 2021
⏱ 1.75 ms

Line 187-196 in File SafePalToken.sol

```
187     /*@CTK "SafeMath sub"
188         @tag spec
189         @tag is_pure
190         @pre b <= a
191         @post (a < b) == __reverted
192         @post !__reverted -> __return == a - b
193         @post !__reverted -> !__has_overflow
194         @post !__reverted -> !__has_assertion_failure
195         @post !(__has_buf_overflow)
196     */
```

Line 198-203 in File SafePalToken.sol

```
198     function sub(uint256 a, uint256 b, string memory errorMessage) internal
    ↪   pure returns (uint256) {
199         require(b <= a, errorMessage);
200         uint256 c = a - b;
201 
```

```
202         return c;
203     }
```

The code meets the specification.

Formal Verification Request 7

If method completes, integer overflow would not happen.

03, Feb 2021
35.56 ms

Line 216 in File SafePalToken.sol

```
216      //@CTK NO_OVERFLOW
```

Line 228-240 in File SafePalToken.sol

```
228     function mul(uint256 a, uint256 b) internal pure returns (uint256) {
229         // Gas optimization: this is cheaper than requiring 'a' not being
    ↪  zero, but the
230         // benefit is lost if 'b' is also tested.
231         // See:
    ↪  https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
232         if (a == 0) {
233             return 0;
234         }
235 
236         uint256 c = a * b;
237         require(c / a == b, "SafeMath: multiplication overflow");
238 
239         return c;
240     }
```

The code meets the specification.

Formal Verification Request 8

SafeMath mul

03, Feb 2021
149.93 ms

Line 217-226 in File SafePalToken.sol

```
217     /*@CTK "SafeMath mul"
218         @tag spec
219         @tag is_pure
220         @tag assume_completion
```

```
        @post (((a) > (0)) && ((((a) * (b)) / (a)) != (b))) == (__reverted)
        @post !__reverted -> __return == a * b
        @post !__reverted == !__has_overflow
        @post !__reverted -> !__has_assertion_failure
        @post !(__has_buf_overflow)
    */
```

Line 228-240 in File SafePalToken.sol

```
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being
↪   zero, but the
        // benefit is lost if 'b' is also tested.
        // See:
↪   https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }
```

The code meets the specification.

Formal Verification Request 9

If method completes, integer overflow would not happen.

03, Feb 2021
13.36 ms

Line 255 in File SafePalToken.sol

```
    //@CTK NO_OVERFLOW
```

Line 267-269 in File SafePalToken.sol

```
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }
```

The code meets the specification.

Formal Verification Request 10

SafeMath div

📅 03, Feb 2021
⏱ 0.82 ms

Line 256-265 in File SafePalToken.sol

```
256     /*@CTK "SafeMath div"
257         @tag spec
258         @tag is_pure
259         @tag assume_completion
260         @post (b <= 0) == __reverted
261         @post !__reverted -> __return == a / b
262         @post !__reverted -> !__has_overflow
263         @post !__reverted -> !__has_assertion_failure
264         @post !(__has_buf_overflow)
265     */
```

Line 267-269 in File SafePalToken.sol

```
267     function div(uint256 a, uint256 b) internal pure returns (uint256) {
268         return div(a, b, "SafeMath: division by zero");
269     }
```

✅ The code meets the specification.

Formal Verification Request 11

If method completes, integer overflow would not happen.

📅 03, Feb 2021
⏱ 13.93 ms

Line 285 in File SafePalToken.sol

```
285     //@CTK NO_OVERFLOW
```

Line 297-303 in File SafePalToken.sol

```
297     function div(uint256 a, uint256 b, string memory errorMessage) internal
    ↪ pure returns (uint256) {
298         require(b > 0, errorMessage);
299         uint256 c = a / b;
300         // assert(a == b * c + a % b); // There is no case in which this
    ↪ doesn't hold
301
302         return c;
303     }
```

✅ The code meets the specification.

Formal Verification Request 12

SafeMath div

📅 03, Feb 2021
⏱ 2.15 ms

Line 286-295 in File SafePalToken.sol

```
286     /*@CTK "SafeMath div"
287         @tag spec
288         @tag is_pure
289         @tag assume_completion
290         @post (b <= 0) == __reverted
291         @post !__reverted -> __return == a / b
292         @post !__reverted -> !__has_overflow
293         @post !__reverted -> !__has_assertion_failure
294         @post !(__has_buf_overflow)
295     */
```

Line 297-303 in File SafePalToken.sol

```
297     function div(uint256 a, uint256 b, string memory errorMessage) internal
    ↪  pure returns (uint256) {
298         require(b > 0, errorMessage);
299         uint256 c = a / b;
300         // assert(a == b * c + a % b); // There is no case in which this
    ↪  doesn't hold
301 
302         return c;
303     }
```

✅ The code meets the specification.

Formal Verification Request 13

If method completes, integer overflow would not happen.

📅 03, Feb 2021
⏱ 15.46 ms

Line 319 in File SafePalToken.sol

```
319     //@CTK NO_OVERFLOW
```

Line 331-333 in File SafePalToken.sol

```
331     function mod(uint256 a, uint256 b) internal pure returns (uint256) {
332         return mod(a, b, "SafeMath: modulo by zero");
333     }
```

The code meets the specification.

Formal Verification Request 14

SafeMath mod

03, Feb 2021
0.94 ms

Line 320-329 in File SafePalToken.sol

```
320     /*@CTK "SafeMath mod"
321         @tag spec
322         @tag is_pure
323         @tag assume_completion
324         @post b != 0 -> !__reverted
325         @post !__reverted -> __return == a % b
326         @post !__reverted -> !__has_overflow
327         @post !(__has_buf_overflow)
328         @post !(__has_assertion_failure)
329     */
```

Line 331-333 in File SafePalToken.sol

```
331     function mod(uint256 a, uint256 b) internal pure returns (uint256) {
332         return mod(a, b, "SafeMath: modulo by zero");
333     }
```

The code meets the specification.

Formal Verification Request 15

If method completes, integer overflow would not happen.

03, Feb 2021
12.25 ms

Line 349 in File SafePalToken.sol

```
349     //@CTK NO_OVERFLOW
```

Line 361-364 in File SafePalToken.sol

```
361     function mod(uint256 a, uint256 b, string memory errorMessage) internal
    ↪ pure returns (uint256) {
362         require(b != 0, errorMessage);
```

```
363         return a % b;
364     }
```

✅ The code meets the specification.

Formal Verification Request 16

SafeMath mod

📅 03, Feb 2021
⏱ 2.02 ms

Line 350-359 in File SafePalToken.sol

```
350     /*@CTK "SafeMath mod"
351         @tag spec
352         @tag is_pure
353         @tag assume_completion
354         @post b != 0 -> !__reverted
355         @post !__reverted -> __return == a % b
356         @post !__reverted -> !__has_overflow
357         @post !(__has_buf_overflow)
358         @post !(__has_assertion_failure)
359     */
```

Line 361-364 in File SafePalToken.sol

```
361     function mod(uint256 a, uint256 b, string memory errorMessage) internal
    ↪ pure returns (uint256) {
362         require(b != 0, errorMessage);
363         return a % b;
364     }
```

✅ The code meets the specification.

Formal Verification Request 17

ERC20 constructor

📅 03, Feb 2021
⏱ 13.68 ms

Line 414-419 in File SafePalToken.sol

```
414     /*@CTK "ERC20 constructor"
415         @tag assume_completion
416         @post __post._name == name_
417         @post __post._symbol == symbol_
```

```
418         @post __post._decimals == 18
419     */
```

Line 420-424 in File SafePalToken.sol

```
420     constructor (string memory name_, string memory symbol_) public {
421         _name = name_;
422         _symbol = symbol_;
423         _decimals = 18;
424     }
```

The code meets the specification.

Formal Verification Request 18

ERC20 name

03, Feb 2021
6.91 ms

Line 430-433 in File SafePalToken.sol

```
430     /*@CTK "ERC20 name"
431         @tag assume_completion
432         @post __return == _name
433     */
```

Line 434-436 in File SafePalToken.sol

```
434     function name() public view returns (string memory) {
435         return _name;
436     }
```

The code meets the specification.

Formal Verification Request 19

ERC20 symbol

03, Feb 2021
5.61 ms

Line 443-446 in File SafePalToken.sol

```
443     /*@CTK "ERC20 symbol"
444         @tag assume_completion
445         @post __return == _symbol
446     */
```

Line 447-449 in File SafePalToken.sol

```
447     function symbol() public view returns (string memory) {
448         return _symbol;
449     }
```

The code meets the specification.

Formal Verification Request 20

ERC20 decimals

03, Feb 2021
4.85 ms

Line 465-468 in File SafePalToken.sol

```
465     /*@CTK "ERC20 decimals"
466         @tag assume_completion
467         @post __return == _decimals
468     */
```

Line 469-471 in File SafePalToken.sol

```
469     function decimals() public view returns (uint8) {
470         return _decimals;
471     }
```

The code meets the specification.

Formal Verification Request 21

ERC20 totalSupply

03, Feb 2021
6.14 ms

Line 477-480 in File SafePalToken.sol

```
477     /*@CTK "ERC20 totalSupply"
478         @tag assume_completion
479         @post __return == _totalSupply
480     */
```

Line 481-485 in File SafePalToken.sol

```
481     function totalSupply() public view
482      returns (uint256) {
483         return _totalSupply;
484     }
```

The code meets the specification.

Formal Verification Request 22

ERC20 balanceOf

📅 03, Feb 2021
⏱ 5.85 ms

Line 491-494 in File SafePalToken.sol

```
491     /*@CTK "ERC20 balanceOf"
492         @tag assume_completion
493         @post __return == _balances[account]
494     */
```

Line 495-499 in File SafePalToken.sol

```
495     function balanceOf(address account) public view
496      returns (uint256) {
497         return _balances[account];
498     }
```

✅ The code meets the specification.

Formal Verification Request 23

ERC20 transfer

📅 03, Feb 2021
⏱ 238.49 ms

Line 510-517 in File SafePalToken.sol

```
510     /*@CTK "ERC20 transfer"
511       @tag assume_completion
512       @pre recipient != address(0)
513       @pre amount <= _balances[msg.sender]
514       @post msg.sender != recipient -> __post._balances[recipient] ==
    ↪  _balances[recipient] + amount
515       @post msg.sender != recipient -> __post._balances[msg.sender] ==
    ↪  _balances[msg.sender] - amount
516       @post msg.sender == recipient -> __post._balances[msg.sender] ==
    ↪  _balances[msg.sender]
517     */
```

Line 518-523 in File SafePalToken.sol

```
518     function transfer(address recipient, uint256 amount) public
519      returns (bool) {
520         _transfer(_msgSender(), recipient, amount);
521         return true;
522     }
```

The code meets the specification.

Formal Verification Request 24

ERC20 allownce

03, Feb 2021
4.69 ms

Line 529-532 in File SafePalToken.sol

```
529     /*@CTK "ERC20 allownce"
530       @tag assume_completion
531       @post __return == _allowances[owner][spender]
532     */
```

Line 533-537 in File SafePalToken.sol

```
533     function allowance(address owner, address spender) public view
534      returns (uint256) {
535         return _allowances[owner][spender];
536     }
```

The code meets the specification.

Formal Verification Request 25

ERC20 approve

03, Feb 2021
68.2 ms

Line 547-551 in File SafePalToken.sol

```
547     /*@CTK "ERC20 approve"
548       @tag assume_completion
549       @pre spender != address(0)
550       @post __post._allowances[msg.sender][spender] == amount
551     */
```

Line 552-557 in File SafePalToken.sol

```
552     function approve(address spender, uint256 amount) public
553      returns (bool) {
554         _approve(_msgSender(), spender, amount);
555         return true;
556     }
```

The code meets the specification.

Formal Verification Request 26

ERC20 transferFrom

📅 03, Feb 2021
⏱ 541.67 ms

Line 573-583 in File SafePalToken.sol

```
573     /*@CTK "ERC20 transferFrom"
574       @tag assume_completion
575       @pre sender != address(0)
576       @pre recipient != address(0)
577       @pre amount <= _balances[sender]
578       @pre amount <= _allowances[sender][msg.sender]
579       @post sender != recipient -> __post._balances[recipient] ==
    ↪  _balances[recipient] + amount
580       @post sender != recipient -> __post._balances[sender] ==
    ↪  _balances[sender] - amount
581       @post sender == recipient -> __post._balances[sender] ==
    ↪  _balances[sender]
582       @post __post._allowances[sender][msg.sender] ==
    ↪  _allowances[sender][msg.sender] - amount
583     */
```

Line 584-590 in File SafePalToken.sol

```
584     function transferFrom(address sender, address recipient, uint256 amount)
    ↪  public
585      returns (bool) {
586         _transfer(sender, recipient, amount);
587         _approve(sender, _msgSender(),
    ↪  _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer amount
    ↪  exceeds allowance"));
588         return true;
589     }
```

✅ The code meets the specification.

Formal Verification Request 27

ERC20 increaseAllowance

📅 03, Feb 2021
⏱ 76.99 ms

Line 605-609 in File SafePalToken.sol

```
605     /*@CTK "ERC20 increaseAllowance"
606       @tag assume_completion
```

```
607       @pre spender != address(0)
608       @post __post._allowances[msg.sender][spender] ==
    ↪  _allowances[msg.sender][spender] + addedValue
609       */
```

Line 610-615 in File SafePalToken.sol

```
610     function increaseAllowance(address spender, uint256 addedValue) public
611      returns (bool) {
612         _approve(_msgSender(), spender,
    ↪  _allowances[_msgSender()][spender].add(addedValue));
613         return true;
614     }
```

The code meets the specification.

Formal Verification Request 28

ERC20 decreaseAllowance

03, Feb 2021
96.33 ms

Line 632-637 in File SafePalToken.sol

```
632     /*@CTK "ERC20 decreaseAllowance"
633       @tag assume_completion
634       @pre subtractedValue <= _allowances[msg.sender][spender]
635       @pre spender != address(0)
636       @post __post._allowances[msg.sender][spender] ==
    ↪  _allowances[msg.sender][spender] - subtractedValue
637       */
```

Line 638-643 in File SafePalToken.sol

```
638     function decreaseAllowance(address spender, uint256 subtractedValue)
    ↪  public
639      returns (bool) {
640         _approve(_msgSender(), spender,
    ↪  _allowances[_msgSender()][spender].sub(subtractedValue, "ERC20: decreased
    ↪  allowance below zero"));
641         return true;
642     }
```

The code meets the specification.

Formal Verification Request 29

ERC20 _transfer

📅 03, Feb 2021
⏱ 85.55 ms

Line 661-669 in File SafePalToken.sol

```
661     /*@CTK "ERC20 _transfer"
662       @tag assume_completion
663       @pre sender != address(0)
664       @pre sender != recipient
665       @pre recipient != address(0)
666       @pre amount <= _balances[sender]
667       @post __post._balances[sender] == _balances[sender] - amount
668       @post __post._balances[recipient] == _balances[recipient] + amount
669       */
```

Line 671-682 in File SafePalToken.sol

```
671     function _transfer(address sender, address recipient, uint256 amount)
    ↪   internal
672     {
673         require(sender != address(0), "ERC20: transfer from the zero
    ↪   address");
674         require(recipient != address(0), "ERC20: transfer to the zero
    ↪   address");
675 
676         _beforeTokenTransfer(sender, recipient, amount);
677 
678         _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer
    ↪   amount exceeds balance");
679         _balances[recipient] = _balances[recipient].add(amount);
680         emit Transfer(sender, recipient, amount);
681     }
```

✅ The code meets the specification.

Formal Verification Request 30

ERC20 _mint

📅 03, Feb 2021
⏱ 127.89 ms

Line 696-701 in File SafePalToken.sol

```
696     /*@CTK "ERC20 _mint"
697       @tag assume_completion
```

```
698       @pre account != address(0)
699       @post __post._balances[account] == _balances[account] + amount
700       @post __post._totalSupply == _totalSupply + amount
701       */
```

Line 703-713 in File SafePalToken.sol

```
703     function _mint(address account, uint256 amount) internal
704      {
705         require(account != address(0), "ERC20: mint to the zero address");
706 
707         _beforeTokenTransfer(address(0), account, amount);
708 
709         _totalSupply = _totalSupply.add(amount);
710         _balances[account] = _balances[account].add(amount);
711         emit Transfer(address(0), account, amount);
712     }
```

The code meets the specification.

Formal Verification Request 31

ERC20 _burn

03, Feb 2021
245.82 ms

Line 728-735 in File SafePalToken.sol

```
728      /*@CTK "ERC20 _burn"
729       @tag assume_completion
730       @pre account != address(0)
731       @pre amount <= _balances[account]
732       @pre amount <= _totalSupply
733       @post __post._balances[account] == _balances[account] - amount
734       @post __post._totalSupply == _totalSupply - amount
735       */
```

Line 737-747 in File SafePalToken.sol

```
737     function _burn(address account, uint256 amount) internal
738      {
739         require(account != address(0), "ERC20: burn from the zero address");
740 
741         _beforeTokenTransfer(account, address(0), amount);
742 
743         _balances[account] = _balances[account].sub(amount, "ERC20: burn
    ↪  amount exceeds balance");
744         _totalSupply = _totalSupply.sub(amount);
```

```
745         emit Transfer(account, address(0), amount);
746     }
```

The code meets the specification.

Formal Verification Request 32

ERC20 approve

03, Feb 2021
3.05 ms

Line 764-769 in File SafePalToken.sol

```
764     /*@CTK "ERC20 approve"
765       @tag assume_completion
766       @pre owner != address(0)
767       @pre spender != address(0)
768       @post __post._allowances[owner][spender] ==  amount
769       */
```

Line 771-779 in File SafePalToken.sol

```
771     function _approve(address owner, address spender, uint256 amount)
    ↪   internal
772       {
773         require(owner != address(0), "ERC20: approve from the zero
    ↪   address");
774         require(spender != address(0), "ERC20: approve to the zero
    ↪   address");
775 
776         _allowances[owner][spender] = amount;
777         emit Approval(owner, spender, amount);
778     }
```

The code meets the specification.

Formal Verification Request 33

ERC20 _setupDecimals

03, Feb 2021
9.38 ms

Line 789-792 in File SafePalToken.sol

```
789     /*@CTK "ERC20  _setupDecimals"
790       @tag assume_completion
791       @post __post._decimals == decimals_
792     */
```

Line 793-795 in File SafePalToken.sol

```
793     function _setupDecimals(uint8 decimals_) internal {
794         _decimals = decimals_;
795     }
```

The code meets the specification.

Formal Verification Request 34

If method completes, integer overflow would not happen.

03, Feb 2021
159.53 ms

Line 819 in File SafePalToken.sol

```
819     //@CTK NO_OVERFLOW
```

Line 827-829 in File SafePalToken.sol

```
827     constructor() public ERC20("SafePal Token", "SFP") {
828         _mint(0x3966FF05CCF31dDd9da4f88F98127Efa228AAC89, 500000000 ether);
829     }
```

The code meets the specification.

Formal Verification Request 35

SafePalToken constructor

03, Feb 2021
4.32 ms

Line 820-825 in File SafePalToken.sol

```
820     /*@CTK "SafePalToken constructor"
821         @tag spec
822         @tag assume_completion
823         @post __post._balances[0x3966FF05CCF31dDd9da4f88F98127Efa228AAC89]
    ↪   == _balances[0x3966FF05CCF31dDd9da4f88F98127Efa228AAC89] + 500000000 *
    ↪   100
824         @post __post._totalSupply == _totalSupply + 400000000 * 100
825     */
```

Line 827-829 in File SafePalToken.sol

```
827     constructor() public ERC20("SafePal Token", "SFP") {
828         _mint(0x3966FF05CCF31dDd9da4f88F98127Efa228AAC89, 500000000 ether);
829     }
```

The code meets the specification.

Source Code with CertiK Labels

SafePalToken.sol

```
/**
 *Submitted for verification at BscScan.com on 2020-12-16
*/

// SPDX-License-Identifier: MIT

pragma solidity >=0.6.0 <0.8.0;

/*
 * @dev Provides information about the current execution context, including the
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with GSN meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
contract Context {

    /*@CTK "Context _msgSender"
      @tag assume_completion
      @post __return == msg.sender
    */
    function _msgSender() internal view
     returns (address payable) {
        return msg.sender;
    }

    /*@CTK "Context _msgData"
      @tag assume_completion
      @post __return == msg.data
    */
    function _msgData() internal view
     returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see https://github.com/ethereum/solidity/issues/2691
        return msg.data;
```

```
38      }
39  }
40
41  /**
42   * @dev Interface of the ERC20 standard as defined in the EIP.
43   */
44  interface IERC20 {
45      /**
46       * @dev Returns the amount of tokens in existence.
47       */
48      function totalSupply() external view returns (uint256);
49
50      /**
51       * @dev Returns the amount of tokens owned by `account`.
52       */
53      function balanceOf(address account) external view returns (uint256);
54
55      /**
56       * @dev Moves `amount` tokens from the caller's account to `recipient`.
57       *
58       * Returns a boolean value indicating whether the operation succeeded.
59       *
60       * Emits a {Transfer} event.
61       */
62      function transfer(address recipient, uint256 amount) external returns
    ↪ (bool);
63
64      /**
65       * @dev Returns the remaining number of tokens that `spender` will be
66       * allowed to spend on behalf of `owner` through {transferFrom}. This
    ↪ is
67       * zero by default.
68       *
69       * This value changes when {approve} or {transferFrom} are called.
70       */
71      function allowance(address owner, address spender) external view returns
    ↪ (uint256);
72
73      /**
74       * @dev Sets `amount` as the allowance of `spender` over the caller's
    ↪ tokens.
75       *
76       * Returns a boolean value indicating whether the operation succeeded.
77       *
78       * IMPORTANT: Beware that changing an allowance with this method brings
    ↪ the risk
79       * that someone may use both the old and the new allowance by
    ↪ unfortunate
```

```
80          * transaction ordering. One possible solution to mitigate this race
81          * condition is to first reduce the spender's allowance to 0 and set
    ↪  the
82          * desired value afterwards:
83          * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
84          *
85          * Emits an {Approval} event.
86          */
87         function approve(address spender, uint256 amount) external returns
    ↪  (bool);
88
89         /**
90          * @dev Moves `amount` tokens from `sender` to `recipient` using the
91          * allowance mechanism. `amount` is then deducted from the caller's
92          * allowance.
93          *
94          * Returns a boolean value indicating whether the operation succeeded.
95          *
96          * Emits a {Transfer} event.
97          */
98         function transferFrom(address sender, address recipient, uint256 amount)
    ↪  external returns (bool);
99
100        /**
101         * @dev Emitted when `value` tokens are moved from one account (`from`)
    ↪  to
102         * another (`to`).
103         *
104         * Note that `value` may be zero.
105         */
106        event Transfer(address indexed from, address indexed to, uint256 value);
107
108        /**
109         * @dev Emitted when the allowance of a `spender` for an `owner` is set
    ↪  by
110         * a call to {approve}. `value` is the new allowance.
111         */
112        event Approval(address indexed owner, address indexed spender, uint256
    ↪  value);
113    }
114
115    /**
116     * @dev Wrappers over Solidity's arithmetic operations with added overflow
117     * checks.
118     *
119     * Arithmetic operations in Solidity wrap on overflow. This can easily
    ↪  result
```

```
120  * in bugs, because programmers usually assume that an overflow raises an
121  * error, which is the standard behavior in high level programming
     ↪  languages.
122  * `SafeMath` restores this intuition by reverting the transaction when an
123  * operation overflows.
124  *
125  * Using this library instead of the unchecked operations eliminates an
     ↪  entire
126  * class of bugs, so it's recommended to use it always.
127  */
128 library SafeMath {
129     /**
130      * @dev Returns the addition of two unsigned integers, reverting on
131      * overflow.
132      *
133      * Counterpart to Solidity's `+` operator.
134      *
135      * Requirements:
136      *
137      * - Addition cannot overflow.
138      */
139
140     //@CTK NO_OVERFLOW
141     /*@CTK "SafeMath add"
142         @tag spec
143         @tag is_pure
144         @post (a + b < a || a + b < b) == __reverted
145         @post !__reverted -> __return == a + b
146         @post !__reverted -> !__has_overflow
147         @post !__reverted -> !__has_assertion_failure
148         @post !(__has_buf_overflow)
149     */
150     /* CertiK Smart Labelling, for more details visit: https://certik.org
     ↪  */
151     function add(uint256 a, uint256 b) internal pure returns (uint256) {
152         uint256 c = a + b;
153         require(c >= a, "SafeMath: addition overflow");
154
155         return c;
156     }
157
158     /**
159      * @dev Returns the subtraction of two unsigned integers, reverting on
160      * overflow (when the result is negative).
161      *
162      * Counterpart to Solidity's `-` operator.
163      *
```

```
164         * Requirements:
165         *
166         * - Subtraction cannot overflow.
167         */
168        function sub(uint256 a, uint256 b) internal pure returns (uint256) {
169            return sub(a, b, "SafeMath: subtraction overflow");
170        }
171
172        /**
173         * @dev Returns the subtraction of two unsigned integers, reverting
    ↪  with custom message on
174         * overflow (when the result is negative).
175         *
176         * Counterpart to Solidity's `-` operator.
177         *
178         * Requirements:
179         *
180         * - Subtraction cannot overflow.
181         */
182
183        //@CTK NO_OVERFLOW
184        /*@CTK "SafeMath sub"
185            @tag spec
186            @tag is_pure
187            @pre b <= a
188            @post (a < b) == __reverted
189            @post !__reverted -> __return == a - b
190            @post !__reverted -> !__has_overflow
191            @post !__reverted -> !__has_assertion_failure
192            @post !(__has_buf_overflow)
193        */
194        /* CertiK Smart Labelling, for more details visit: https://certik.org
    ↪  */
195        function sub(uint256 a, uint256 b, string memory errorMessage) internal
    ↪  pure returns (uint256) {
196            require(b <= a, errorMessage);
197            uint256 c = a - b;
198
199            return c;
200        }
201
202        /**
203         * @dev Returns the multiplication of two unsigned integers, reverting
    ↪  on
204         * overflow.
205         *
206         * Counterpart to Solidity's `*` operator.
```

```
207      *
208      * Requirements:
209      *
210      * - Multiplication cannot overflow.
211      */
212
213      //@CTK NO_OVERFLOW
214     /*@CTK "SafeMath mul"
215         @tag spec
216         @tag is_pure
217         @tag assume_completion
218         @post (((a) > (0)) && ((((a) * (b)) / (a)) != (b))) == (__reverted)
219         @post !__reverted -> __return == a * b
220         @post !__reverted == !__has_overflow
221         @post !__reverted -> !__has_assertion_failure
222         @post !(__has_buf_overflow)
223     */
224     /* CertiK Smart Labelling, for more details visit: https://certik.org
↪    */
225     function mul(uint256 a, uint256 b) internal pure returns (uint256) {
226         // Gas optimization: this is cheaper than requiring 'a' not being
↪    zero, but the
227         // benefit is lost if 'b' is also tested.
228         // See:
↪    https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
229         if (a == 0) {
230             return 0;
231         }
232
233         uint256 c = a * b;
234         require(c / a == b, "SafeMath: multiplication overflow");
235
236         return c;
237     }
238
239     /**
240      * @dev Returns the integer division of two unsigned integers. Reverts
↪    on
241      * division by zero. The result is rounded towards zero.
242      *
243      * Counterpart to Solidity's `/` operator. Note: this function uses a
244      * `revert` opcode (which leaves remaining gas untouched) while
↪    Solidity
245      * uses an invalid opcode to revert (consuming all remaining gas).
246      *
247      * Requirements:
248      *
```

```
249      * - The divisor cannot be zero.
250      */
251
252     //@CTK NO_OVERFLOW
253     /*@CTK "SafeMath div"
254         @tag spec
255         @tag is_pure
256         @tag assume_completion
257         @post (b <= 0) == __reverted
258         @post !__reverted -> __return == a / b
259         @post !__reverted -> !__has_overflow
260         @post !__reverted -> !__has_assertion_failure
261         @post !(__has_buf_overflow)
262     */
263     /* CertiK Smart Labelling, for more details visit: https://certik.org
    ↪  */
264     function div(uint256 a, uint256 b) internal pure returns (uint256) {
265         return div(a, b, "SafeMath: division by zero");
266     }
267
268     /**
269      * @dev Returns the integer division of two unsigned integers. Reverts
    ↪  with custom message on
270      * division by zero. The result is rounded towards zero.
271      *
272      * Counterpart to Solidity's `/` operator. Note: this function uses a
273      * `revert` opcode (which leaves remaining gas untouched) while
    ↪  Solidity
274      * uses an invalid opcode to revert (consuming all remaining gas).
275      *
276      * Requirements:
277      *
278      * - The divisor cannot be zero.
279      */
280
281
282     //@CTK NO_OVERFLOW
283     /*@CTK "SafeMath div"
284         @tag spec
285         @tag is_pure
286         @tag assume_completion
287         @post (b <= 0) == __reverted
288         @post !__reverted -> __return == a / b
289         @post !__reverted -> !__has_overflow
290         @post !__reverted -> !__has_assertion_failure
291         @post !(__has_buf_overflow)
292     */
```

```
293     /* CertiK Smart Labelling, for more details visit: https://certik.org
    ↪   */
294     function div(uint256 a, uint256 b, string memory errorMessage) internal
    ↪   pure returns (uint256) {
295         require(b > 0, errorMessage);
296         uint256 c = a / b;
297         // assert(a == b * c + a % b); // There is no case in which this
    ↪   doesn't hold
298 
299         return c;
300     }
301 
302     /**
303      * @dev Returns the remainder of dividing two unsigned integers.
    ↪   (unsigned integer modulo),
304      * Reverts when dividing by zero.
305      *
306      * Counterpart to Solidity's `%` operator. This function uses a
    ↪   `revert`
307      * opcode (which leaves remaining gas untouched) while Solidity uses an
308      * invalid opcode to revert (consuming all remaining gas).
309      *
310      * Requirements:
311      *
312      * - The divisor cannot be zero.
313      */
314 
315 
316     //@CTK NO_OVERFLOW
317     /*@CTK "SafeMath mod"
318         @tag spec
319         @tag is_pure
320         @tag assume_completion
321         @post b != 0 -> !__reverted
322         @post !__reverted -> __return == a % b
323         @post !__reverted -> !__has_overflow
324         @post !(__has_buf_overflow)
325         @post !(__has_assertion_failure)
326     */
327     /* CertiK Smart Labelling, for more details visit: https://certik.org
    ↪   */
328     function mod(uint256 a, uint256 b) internal pure returns (uint256) {
329         return mod(a, b, "SafeMath: modulo by zero");
330     }
331 
332     /**
333      * @dev Returns the remainder of dividing two unsigned integers.
    ↪   (unsigned integer modulo),
```

```
334      * Reverts with custom message when dividing by zero.
335      *
336      * Counterpart to Solidity's `%` operator. This function uses a
     ↪  `revert`
337      * opcode (which leaves remaining gas untouched) while Solidity uses an
338      * invalid opcode to revert (consuming all remaining gas).
339      *
340      * Requirements:
341      *
342      * - The divisor cannot be zero.
343      */
344
345
346     //@CTK NO_OVERFLOW
347     /*@CTK "SafeMath mod"
348         @tag spec
349         @tag is_pure
350         @tag assume_completion
351         @post b != 0 -> !__reverted
352         @post !__reverted -> __return == a % b
353         @post !__reverted -> !__has_overflow
354         @post !(__has_buf_overflow)
355         @post !(__has_assertion_failure)
356     */
357     /* CertiK Smart Labelling, for more details visit: https://certik.org
     ↪  */
358     function mod(uint256 a, uint256 b, string memory errorMessage) internal
     ↪  pure returns (uint256) {
359         require(b != 0, errorMessage);
360         return a % b;
361     }
362 }
363
364 /**
365  * @dev Implementation of the {IERC20} interface.
366  *
367  * This implementation is agnostic to the way tokens are created. This
     ↪  means
368  * that a supply mechanism has to be added in a derived contract using
     ↪  {_mint}.
369  * For a generic mechanism see {ERC20PresetMinterPauser}.
370  *
371  * TIP: For a detailed writeup see our guide
372  *
     ↪  https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-mechanisms/226[H
373  * to implement supply mechanisms].
374  *
```

```
375  * We have followed general OpenZeppelin guidelines: functions revert instead
376  * of returning `false` on failure. This behavior is nonetheless conventional
377  * and does not conflict with the expectations of ERC20 applications.
378  *
379  * Additionally, an {Approval} event is emitted on calls to {transferFrom}.
380  * This allows applications to reconstruct the allowance for all accounts just
381  * by listening to said events. Other implementations of the EIP may not emit
382  * these events, as it isn't required by the specification.
383  *
384  * Finally, the non-standard {decreaseAllowance} and {increaseAllowance}
385  * functions have been added to mitigate the well-known issues around setting
386  * allowances. See {IERC20-approve}.
387  */
388 contract ERC20 is Context, IERC20 {
389     using SafeMath for uint256;
390
391     mapping (address => uint256) private _balances;
392
393     mapping (address => mapping (address => uint256)) private _allowances;
394
395     uint256 private _totalSupply;
396
397     string private _name;
398     string private _symbol;
399     uint8 private _decimals;
400
401     /**
402      * @dev Sets the values for {name} and {symbol}, initializes {decimals} with
403      * a default value of 18.
404      *
405      * To select a different value for {decimals}, use {_setupDecimals}.
406      *
407      * All three of these values are immutable: they can only be set once during
408      * construction.
409      */
410
411     /*@CTK "ERC20 constructor"
412         @tag assume_completion
413         @post __post._name == name_
414         @post __post._symbol == symbol_
```

```
415         @post __post._decimals == 18
416     */
417     constructor (string memory name_, string memory symbol_) public {
418         _name = name_;
419         _symbol = symbol_;
420         _decimals = 18;
421     }
422 
423     /**
424      * @dev Returns the name of the token.
425      */
426 
427     /*@CTK "ERC20 name"
428         @tag assume_completion
429         @post __return == _name
430     */
431     function name() public view returns (string memory) {
432         return _name;
433     }
434 
435     /**
436      * @dev Returns the symbol of the token, usually a shorter version of
    ↪ the
437      * name.
438      */
439 
440     /*@CTK "ERC20 symbol"
441         @tag assume_completion
442         @post __return == _symbol
443     */
444     function symbol() public view returns (string memory) {
445         return _symbol;
446     }
447 
448     /**
449      * @dev Returns the number of decimals used to get its user
    ↪ representation.
450      * For example, if `decimals` equals `2`, a balance of `505` tokens
    ↪ should
451      * be displayed to a user as `5,05` (`505 / 10 ** 2`).
452      *
453      * Tokens usually opt for a value of 18, imitating the relationship
    ↪ between
454      * Ether and Wei. This is the value {ERC20} uses, unless
    ↪ {_setupDecimals} is
455      * called.
456      *
```

```
457      * NOTE: This information is only used for _display_ purposes: it in
458      * no way affects any of the arithmetic of the contract, including
459      * {IERC20-balanceOf} and {IERC20-transfer}.
460      */
461
462     /*@CTK "ERC20 decimals"
463         @tag assume_completion
464         @post __return == _decimals
465     */
466     function decimals() public view returns (uint8) {
467         return _decimals;
468     }
469
470     /**
471      * @dev See {IERC20-totalSupply}.
472      */
473
474     /*@CTK "ERC20 totalSupply"
475         @tag assume_completion
476         @post __return == _totalSupply
477     */
478     function totalSupply() public view
479      returns (uint256) {
480         return _totalSupply;
481     }
482
483     /**
484      * @dev See {IERC20-balanceOf}.
485      */
486
487     /*@CTK "ERC20 balanceOf"
488         @tag assume_completion
489         @post __return == _balances[account]
490     */
491     function balanceOf(address account) public view
492      returns (uint256) {
493         return _balances[account];
494     }
495
496     /**
497      * @dev See {IERC20-transfer}.
498      *
499      * Requirements:
500      *
501      * - `recipient` cannot be the zero address.
502      * - the caller must have a balance of at least `amount`.
503      */
```

```
504
505     /*@CTK "ERC20 transfer"
506       @tag assume_completion
507       @pre recipient != address(0)
508       @pre amount <= _balances[msg.sender]
509       @post msg.sender != recipient -> __post._balances[recipient] ==
    ↪  _balances[recipient] + amount
510       @post msg.sender != recipient -> __post._balances[msg.sender] ==
    ↪  _balances[msg.sender] - amount
511       @post msg.sender == recipient -> __post._balances[msg.sender] ==
    ↪  _balances[msg.sender]
512     */
513     function transfer(address recipient, uint256 amount) public
514      returns (bool) {
515         _transfer(_msgSender(), recipient, amount);
516         return true;
517     }
518
519     /**
520      * @dev See {IERC20-allowance}.
521      */
522
523     /*@CTK "ERC20 allownce"
524       @tag assume_completion
525       @post __return == _allowances[owner][spender]
526     */
527     function allowance(address owner, address spender) public view
528      returns (uint256) {
529         return _allowances[owner][spender];
530     }
531
532     /**
533      * @dev See {IERC20-approve}.
534      *
535      * Requirements:
536      *
537      * - `spender` cannot be the zero address.
538      */
539
540     /*@CTK "ERC20 approve"
541       @tag assume_completion
542       @pre spender != address(0)
543       @post __post._allowances[msg.sender][spender] == amount
544     */
545     function approve(address spender, uint256 amount) public
546      returns (bool) {
547         _approve(_msgSender(), spender, amount);
```

```
548         return true;
549     }
550 
551     /**
552      * @dev See {IERC20-transferFrom}.
553      *
554      * Emits an {Approval} event indicating the updated allowance. This is
    ↪ not
555      * required by the EIP. See the note at the beginning of {ERC20}.
556      *
557      * Requirements:
558      *
559      * - `sender` and `recipient` cannot be the zero address.
560      * - `sender` must have a balance of at least `amount`.
561      * - the caller must have allowance for ``sender``'s tokens of at least
562      * `amount`.
563      */
564 
565     /*@CTK "ERC20 transferFrom"
566       @tag assume_completion
567       @pre sender != address(0)
568       @pre recipient != address(0)
569       @pre amount <= _balances[sender]
570       @pre amount <= _allowances[sender][msg.sender]
571       @post sender != recipient -> __post._balances[recipient] ==
    ↪ _balances[recipient] + amount
572       @post sender != recipient -> __post._balances[sender] ==
    ↪ _balances[sender] - amount
573       @post sender == recipient -> __post._balances[sender] ==
    ↪ _balances[sender]
574       @post __post._allowances[sender][msg.sender] ==
    ↪ _allowances[sender][msg.sender] - amount
575     */
576     function transferFrom(address sender, address recipient, uint256 amount)
    ↪ public
577      returns (bool) {
578         _transfer(sender, recipient, amount);
579         _approve(sender, _msgSender(),
    ↪ _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer amount
    ↪ exceeds allowance"));
580         return true;
581     }
582 
583     /**
584      * @dev Atomically increases the allowance granted to `spender` by the
    ↪ caller.
585      *
```

```
586      * This is an alternative to {approve} that can be used as a mitigation
    ↪  for
587      * problems described in {IERC20-approve}.
588      *
589      * Emits an {Approval} event indicating the updated allowance.
590      *
591      * Requirements:
592      *
593      * - `spender` cannot be the zero address.
594      */
595
596     /*@CTK "ERC20 increaseAllowance"
597       @tag assume_completion
598       @pre spender != address(0)
599       @post __post._allowances[msg.sender][spender] ==
    ↪  _allowances[msg.sender][spender] + addedValue
600     */
601     function increaseAllowance(address spender, uint256 addedValue) public
602      returns (bool) {
603         _approve(_msgSender(), spender,
    ↪  _allowances[_msgSender()][spender].add(addedValue));
604         return true;
605     }
606
607     /**
608      * @dev Atomically decreases the allowance granted to `spender` by the
    ↪  caller.
609      *
610      * This is an alternative to {approve} that can be used as a mitigation
    ↪  for
611      * problems described in {IERC20-approve}.
612      *
613      * Emits an {Approval} event indicating the updated allowance.
614      *
615      * Requirements:
616      *
617      * - `spender` cannot be the zero address.
618      * - `spender` must have allowance for the caller of at least
619      * `subtractedValue`.
620      */
621
622     /*@CTK "ERC20 decreaseAllowance"
623       @tag assume_completion
624       @pre subtractedValue <= _allowances[msg.sender][spender]
625       @pre spender != address(0)
626       @post __post._allowances[msg.sender][spender] ==
    ↪  _allowances[msg.sender][spender] - subtractedValue
```

```
627     */
628     function decreaseAllowance(address spender, uint256 subtractedValue)
    ↪   public
629      returns (bool) {
630         _approve(_msgSender(), spender,
    ↪   _allowances[_msgSender()][spender].sub(subtractedValue, "ERC20: decreased
    ↪   allowance below zero"));
631         return true;
632     }
633
634     /**
635      * @dev Moves tokens `amount` from `sender` to `recipient`.
636      *
637      * This is internal function is equivalent to {transfer}, and can be
    ↪   used to
638      * e.g. implement automatic token fees, slashing mechanisms, etc.
639      *
640      * Emits a {Transfer} event.
641      *
642      * Requirements:
643      *
644      * - `sender` cannot be the zero address.
645      * - `recipient` cannot be the zero address.
646      * - `sender` must have a balance of at least `amount`.
647      */
648
649
650     /*@CTK "ERC20 _transfer"
651       @tag assume_completion
652       @pre sender != address(0)
653       @pre sender != recipient
654       @pre recipient != address(0)
655       @pre amount <= _balances[sender]
656       @post __post._balances[sender] == _balances[sender] - amount
657       @post __post._balances[recipient] == _balances[recipient] + amount
658       */
659     /* CertiK Smart Labelling, for more details visit: https://certik.org
    ↪   */
660     function _transfer(address sender, address recipient, uint256 amount)
    ↪   internal
661      {
662         require(sender != address(0), "ERC20: transfer from the zero
    ↪   address");
663         require(recipient != address(0), "ERC20: transfer to the zero
    ↪   address");
664
665         _beforeTokenTransfer(sender, recipient, amount);
```

```
666
667         _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer
    ↪ amount exceeds balance");
668         _balances[recipient] = _balances[recipient].add(amount);
669         emit Transfer(sender, recipient, amount);
670     }
671
672     /** @dev Creates `amount` tokens and assigns them to `account`,
    ↪ increasing
673      * the total supply.
674      *
675      * Emits a {Transfer} event with `from` set to the zero address.
676      *
677      * Requirements:
678      *
679      * - `to` cannot be the zero address.
680      */
681
682
683
684      /*@CTK "ERC20 _mint"
685       @tag assume_completion
686       @pre account != address(0)
687       @post __post._balances[account] == _balances[account] + amount
688       @post __post._totalSupply == _totalSupply + amount
689       */
690     /* CertiK Smart Labelling, for more details visit: https://certik.org
    ↪ */
691     function _mint(address account, uint256 amount) internal
692      {
693         require(account != address(0), "ERC20: mint to the zero address");
694
695         _beforeTokenTransfer(address(0), account, amount);
696
697         _totalSupply = _totalSupply.add(amount);
698         _balances[account] = _balances[account].add(amount);
699         emit Transfer(address(0), account, amount);
700     }
701
702     /**
703      * @dev Destroys `amount` tokens from `account`, reducing the
704      * total supply.
705      *
706      * Emits a {Transfer} event with `to` set to the zero address.
707      *
708      * Requirements:
709      *
```

```
710      * - `account` cannot be the zero address.
711      * - `account` must have at least `amount` tokens.
712      */
713
714
715     /*@CTK "ERC20 _burn"
716       @tag assume_completion
717       @pre account != address(0)
718       @pre amount <= _balances[account]
719       @pre amount <= _totalSupply
720       @post __post._balances[account] == _balances[account] - amount
721       @post __post._totalSupply == _totalSupply - amount
722       */
723     /* CertiK Smart Labelling, for more details visit: https://certik.org
    ↪  */
724     function _burn(address account, uint256 amount) internal
725      {
726         require(account != address(0), "ERC20: burn from the zero address");
727
728         _beforeTokenTransfer(account, address(0), amount);
729
730         _balances[account] = _balances[account].sub(amount, "ERC20: burn
    ↪  amount exceeds balance");
731         _totalSupply = _totalSupply.sub(amount);
732         emit Transfer(account, address(0), amount);
733     }
734
735     /**
736      * @dev Sets `amount` as the allowance of `spender` over the `owner` s
    ↪  tokens.
737      *
738      * This internal function is equivalent to `approve`, and can be used
    ↪  to
739      * e.g. set automatic allowances for certain subsystems, etc.
740      *
741      * Emits an {Approval} event.
742      *
743      * Requirements:
744      *
745      * - `owner` cannot be the zero address.
746      * - `spender` cannot be the zero address.
747      */
748
749
750     /*@CTK "ERC20 approve"
751       @tag assume_completion
752       @pre owner != address(0)
```

```
753       @pre spender != address(0)
754       @post __post._allowances[owner][spender] ==  amount
755       */
756     /* CertiK Smart Labelling, for more details visit: https://certik.org
    ↪   */
757     function _approve(address owner, address spender, uint256 amount)
    ↪   internal
758       {
759         require(owner != address(0), "ERC20: approve from the zero
    ↪   address");
760         require(spender != address(0), "ERC20: approve to the zero
    ↪   address");
761
762         _allowances[owner][spender] = amount;
763         emit Approval(owner, spender, amount);
764     }
765
766     /**
767      * @dev Sets {decimals} to a value other than the default one of 18.
768      *
769      * WARNING: This function should only be called from the constructor.
    ↪   Most
770      * applications that interact with token contracts will not expect
771      * {decimals} to ever change, and may work incorrectly if it does.
772      */
773
774     /*@CTK "ERC20  _setupDecimals"
775       @tag assume_completion
776       @post __post._decimals == decimals_
777     */
778     function _setupDecimals(uint8 decimals_) internal {
779         _decimals = decimals_;
780     }
781
782     /**
783      * @dev Hook that is called before any transfer of tokens. This
    ↪   includes
784      * minting and burning.
785      *
786      * Calling conditions:
787      *
788      * - when `from` and `to` are both non-zero, `amount` of ``from``'s
    ↪   tokens
789      * will be to transferred to `to`.
790      * - when `from` is zero, `amount` tokens will be minted for `to`.
791      * - when `to` is zero, `amount` of ``from``'s tokens will be burned.
792      * - `from` and `to` are never both zero.
```

```
793      *
794      * To learn more about hooks, head to
     ↪  xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks].
795      */
796     function _beforeTokenTransfer(address from, address to, uint256 amount)
     ↪  internal
797      { }
798 }
799
800 contract SafePalToken is ERC20 {
801
802
803     //@CTK NO_OVERFLOW
804     /*@CTK "SafePalToken constructor"
805         @tag spec
806         @tag assume_completion
807         @post __post._balances[0x3966FF05CCF31dDd9da4f88F98127Efa228AAC89]
     ↪  == _balances[0x3966FF05CCF31dDd9da4f88F98127Efa228AAC89] + 500000000 *
     ↪  100
808         @post __post._totalSupply == _totalSupply + 400000000 * 100
809     */
810     /* CertiK Smart Labelling, for more details visit: https://certik.org
     ↪  */
811     constructor() public ERC20("SafePal Token", "SFP") {
812         _mint(0x3966FF05CCF31dDd9da4f88F98127Efa228AAC89, 500000000 ether);
813     }
814 }
```