US +1 415 670-9800, UK +44 33 3303 1363
contact@crispmind.io, @crispmind
https://crispmind.io & softnote.com

TET MIGRATION OVERVIEW

INTRODUCTION

The TET migration Model was designed to integrate T21 protocol-based digital entities with other primary protocols such as ERC20. Since the number of TET in circulation must always be equal to the Hard Cap of 10M, the Smart Contracts were designed to keep exactly this amount unlocked at any given point in time. The team has chosen to premint the entire amount of TET on every network in order to avoid double fees (mint + transfer fee = double fee).

Since all the TET were preminted, our primary goal was to ensure that no TET in network A would ever be unlocked without equal number of TET being locked in network B. Source code below proves that. Additionally, we have added our private key regeneration model description below to take care of private key vulnerability issue.

UNDERLYING PRINCIPLES

1. All TET in all side protocols are minted ahead of time.
2. All TET in all side network protocols are minted as a single-time emission event.
3. Only 10M TET (Hard Cap) is minted in every side network protocol individually – 10M TET per side network protocol.
4. At any given point in time only 10M TET must be in unlocked state in all side network protocols combined.
5. When X number of TET are frozen in network A – the same X number of TET is released in network B.

CODE PROOF

1. T12-ERC20 API EXTRACTION

```
migrateToEth(String recipientEthToken) async {
  final userTokenTET = (await Address.getUserAddress(parser, currencyTET))?.address;
  if(userTokenTET != null){
    // smart contract transfer of TET to the user Eth token
    final trEthTET = await NodeRepository.request(SendFromToCommand(_scKey!, amount, currencyEthTET.port, recipientEthToken));
    // waiting for the result of the smart contract execution
    if (await executionResult(trEthTET)) {
      // if the result is successful, the transfer of the TET of the Tectum network user to the storage address
      final trTET = await NodeRepository.request(TokenTransferCommand(userKey!, userTokenTET, _storageTokenTET, amount));
      if (trTET is TokenTransferResponse) {
        // with a successful transaction, we make a transfer of ether, for the execution of a smart contract
        await NodeRepository.request(SendFromToCommand(userKey!, feeSmartContract, currencyEth.port, _storageTokenEth));
      }
    }
  }
}
```

US +1 415 670-9800, UK +44 33 3303 1363
contact@crispmind.io, @crispmind
https://crispmind.io & softnote.com

2. ERC20

```
import 'dart:io';

import 'package:template/coder.dart';
import 'package:template/const.dart';
import 'package:template/email.dart';
import 'package:yaml/yaml.dart';

import '../../bin/environment.dart';
import '../../models/access_token.dart';
import '../../models/address.dart';
import '../../models/currency.dart';
import '../../models/operation.dart';
import '../../module/tectum/command.dart';
import '../../module/tectum/repository.dart';
import '../../module/tectum/response.dart';
import '../../module/user/repository.dart';
import 'smartcontract.dart';

class SmartContractEthereum implements SmartContract {

  num amount = 0;
  final TokenParser parser;
  num feeSmartContract = 0;
  num feeNetwork = 0;
  num balance = 0;
  String? userKey;

  ResponseNode? error;

  var _params = <String, dynamic>{};

  String? get _scKey => _params['key'];

  String? get _storageToken => _params[currency.network];

  String? get _storageTokenTET => _params['token'];

  final currencyTET = Currency.getById(2)!;

  Currency currency = Currency.getById(7)!;

      Currency currencyOutTET = Currency.getById(8)!;


  var data = <String, dynamic>{};

  String _moderator = MODER_EMAIL;

  SmartContractEthereum(this.parser) {
    final configTmpl = File(rootPath('sc.yaml', dir: CONFIG_PATH));
    if (configTmpl.existsSync()) {
      _params = Map<String,
dynamic>.from(loadYaml(configTmpl.readAsStringSync()));
    }
```

US +1 415 670-9800, UK +44 33 3303 1363
contact@crispmind.io, @crispmind
https://crispmind.io & softnote.com

```
  }

  Future<URKError?> init(num amount) async {
    this.amount = amount;
    await _getFee();
    userKey = await parser.key;
    if (userKey == null) {
      print('error sessionKey=$userKey');
      return URKError.create(ECR_KEY_NULL);
    }
    print('init 1');
    balance = (await currencyTET.getBalance(userKey, parser.id!)) ?? 0;
    data['parser'] = parser.token;
    data['id'] = parser.id;
    data['to'] = currency.ticker.toUpperCase();
    data['balanceTET'] = balance;

    if (balance < amount) {
      return URKError.create(ECR_NOT_FUNDS, 'TET');
    }

    if (!TEST_MODE) {
      if (!await checkBalanceOut()) {
        return URKError.create(ECR_NOT_FUNDS, 'Etherium');
      }
    }

    if (fee == 0) {
      return URKError.create(ECR_SERVICE_NOT_DATA, 'Fee is null');
    }

    return null;
  }

  @override
  Future<Operation?> request(String? recipientEthToken) async {
           final op = Operation(parser.id, currencyOutTET.key);
    final userTokenTET =
        (await Address.getUserAddress(parser, currencyTET))?.address;
    recipientEthToken ??
        (await Address.getUserAddress(parser, currency))?.address;
    if (userTokenTET != null) {
      data['addressTET'] = userTokenTET;
      data['recipient'] = recipientEthToken;
      await op.wait(OperationArea.WALLET, OperationActions.MIGRATED_IN,
amount,
          data: data);
      data['execute'] = Coder()
          .encryptText('${op.id}:${parser.id}:1:${currencyOutTET.key}')
          .asBase58;
      data['reject'] = Coder()
          .encryptText('${op.id}:${parser.id}:-1:${currencyOutTET.key}')
          .asBase58;
      data['operation'] = op.id;
```

US +1 415 670-9800, UK +44 33 3303 1363
contact@crispmind.io, @crispmind
https://crispmind.io & softnote.com

```
      data['amount'] = amount;
      final user = await UserIdRepository.user(parser.id);
      if (user != null) {
        data['email'] = user['email'];
      }
      print('EmailSend $data');
      EmailSend.sendTemplateEmail(
          _moderator, 'Request #${op.id}', 'request', data);
      return op;
    } else {
      error= URKError.create(ECR_PARAM_NOT_SET, 'not found userTokenTET');
    }

    return null;

  }

  @override
  Future<int?> migrateOut(String recipientEthToken, opId) async {
    final userTokenTET =
        Address.addressFromCache(parser.id!, currencyTET)?.address;
    if (userTokenTET != null) {
      print('migrateToEth userTokenTET=$userTokenTET');
      // smart contract transfer of TET to the user Eth token
      final trEthTET = await NodeRepository.request(SendFromToCommand(
          _scKey!, amount, currencyOutTET.port, recipientEthToken));
      print('migrateToEth trEthTET=$trEthTET');

      // waiting for the result of the smart contract execution
      if (await executionResult(trEthTET)) {
        // if the result is successful, the transfer of the TET of the
Tectum network user to the storage address
        final trTET = await NodeRepository.request(TokenTransferCommand(
            userKey!, userTokenTET, _storageTokenTET, amount));
        print('migrateToEth trTET=$trTET');
        if (trTET is TokenTransferResponse) {
          Operation(parser.id!, currencyTET.key).success(
              OperationArea.WALLET, OperationActions.MIGRATED_OUT,
amount);
          // with a successful transaction, we make a transfer of ether,
for the execution of a smart contract
          final trEth = await NodeRepository.request(SendFromToCommand(
              userKey!, feeSmartContract, currency.port, _storageToken));
          print('migrateToEth trEth=$trEth');
          if (trEth is SendFromResponse) {
            Operation(parser.id!, currency.key).success(
                OperationArea.WALLET, OperationActions.SENT,
feeSmartContract);
          } else {
            if (trEth is URKError) {
              error = trEth..stack = 'Migrated Eth Ð¢Ð•Ð¢';
            }
          }
          if (trEthTET is SendFromResponse && trEth is SendFromResponse) {
```

US +1 415 670-9800, UK +44 33 3303 1363
contact@crispmind.io, @crispmind
https://crispmind.io & softnote.com

```
            EmailSend.sendTemplateEmail(
                _moderator, 'Success Operation #${opId}', 'status', {
              'migrate': trEthTET.transactionId,
              'migrateTET': trTET.transactionId,
              'transaction': trEth.transactionId,
            }, [
              'faronzaz@gmail.com'
            ]);
          }
          return opId;
        } else {
          if (trTET is URKError) {
            error = trTET..stack = 'Migrated Ð¢Ð•Ð¢';
          }
        }
      }
    } else {
      error = URKError.create(ECR_PARAM_NOT_SET, 'not found
userTokenTET');
    }
    return null;
  }

  @override
  Future<int?> migrateIn(
      String? recipientTectumToken, Operation op) async {
    final userTokenEth =
        (await Address.getUserAddress(parser, currency))?.address;
    final userTokenTET = recipientTectumToken ??
        (await Address.getUserAddress(parser, currencyTET))?.address;
    if (userTokenEth != null && userTokenTET != null) {
      print(
          'migrateToTectum:  userTokenTET=$userTokenTET
userTokenEth=$userTokenEth');
      final trETET = await NodeRepository.request(SendFromToCommand(
          userKey!, amount, currencyOutTET.port, _storageToken));
      print('migrateToTectum: trETET=$trETET =$userTokenEth');
      await op.wait(
          OperationArea.WALLET, OperationActions.MIGRATED_OUT, amount);
      // waiting for the result of the smart contract execution
      if (await executionResult(trETET)) {
        await op.updateSuccess();
        // if the result is successful, the transfer of the TET of the
Tectum network user to the storage address
        final trTET = await NodeRepository.request(TokenSysCommand(
            _scKey!, _storageTokenTET, recipientTectumToken, amount));
        if (trTET is TokenSysResponse) {
          Operation(parser.id!, currencyTET.key).success(
              OperationArea.WALLET, OperationActions.MIGRATED_IN, amount);
        } else {
          if (trTET is URKError) {
            error = trTET..stack = 'Migrated Ð¢Ð•Ð¢';
          }
        }
```

US +1 415 670-9800, UK +44 33 3303 1363
contact@crispmind.io, @crispmind
https://crispmind.io & softnote.com

```
        return op.id;
      } else {
        await op.updateError();
      }
    } else {
      error = URKError.create(
          ECR_PARAM_NOT_SET, 'not found userTokenEth and userTokenTET');
    }
    return null;
  }


  Future<bool> executionResult(ResponseNode resp) async {
    print('execute resp = $resp');
    var result = false;
    if (resp is URKError) {
      error = resp..stack = 'Migrated';
    }
    if (resp is SendFromResponse) {
      for (var i = 0; i < 100; i++) {
        await Future.delayed(Duration(milliseconds: 500));
        final status =
            await
NodeRepository.request(CheckEthTxCommand(resp.transactionId));
        print('execute status=${status}');
        if (status is CheckResultResponse &&
            '${status.status}'.toUpperCase() == 'OK') {
          result = true;
          error = null;
          break;
        } else {
          if (status is URKError) {
            error = status..stack = 'Check Result';
          }
        }
      }
    }
    print('execute result=${result}');
    return result;
  }

  URKError gerError() {
    if (error != null && error is URKError) {
      return error as URKError;
    }
    return URKError.create(0);
  }

  Future<bool> checkBalanceOut() async {
    final balance = await currency.getBalance(userKey, parser.id!) ?? 0;
    print('checkBalanceETH $balance - ($fee)');
    data['balanceTo'] = balance;
    return balance >= fee;
  }
```

US +1 415 670-9800, UK +44 33 3303 1363
contact@crispmind.io, @crispmind
https://crispmind.io & softnote.com

```
  _getFee() async {
    final res = await NodeRepository.request(GetEthFeeCommand());
    if (res is GetEthFeeResponse) {
      feeNetwork = res.ethFee;
      feeSmartContract = res.firstTokenFee;
                }

    print('getFee');
  }

  @override
  num get fee => feeNetwork + feeSmartContract;


}

/**
 *  Future<Operation?> request(String? recipientEthToken) async {

                }
 */

class SmartContractResult {}
```

US +1 415 670-9800, UK +44 33 3303 1363
contact@crispmind.io, @crispmind
https://crispmind.io & softnote.com

3. BEP20

```
import 'dart:convert';
import 'dart:io';

import 'package:http/http.dart';
import 'package:template/coder.dart';
import 'package:template/const.dart';
import 'package:template/email.dart';
import 'package:yaml/yaml.dart';

import '../../bin/environment.dart';
import '../../models/access_token.dart';
import '../../models/address.dart';
import '../../models/currency.dart';
import '../../models/operation.dart';
import '../../module/tectum/command.dart';
import '../../module/tectum/repository.dart';
import '../../module/tectum/response.dart';
import '../../module/user/repository.dart';
import 'smartcontract.dart';

class SmartContractBinance implements SmartContract{

  num amount = 0;
  final TokenParser parser;

  num feeSmartContract = 0;
  num feeNetwork = 0;
  num balance = 0;
  String? userKey;

  ResponseNode? error;

  var _params = <String, dynamic>{};

  String? get _scKey => _params['key'];

  String? get _storageToken => _params[currency.network];

  String? get _storageTokenTET => _params['token'];

  final currencyTET = Currency.getById(2)!;

      Currency currency = Currency.getById(9)!;

  Currency currencyOutTET = Currency.getById(10)!;

      var data = <String,dynamic>{};

      String _moderator = MODER_EMAIL;


  SmartContractBinance(this.parser) {
    final configTmpl = File(rootPath('sc.yaml', dir: CONFIG_PATH));
    if (configTmpl.existsSync()) {
```

US +1 415 670-9800, UK +44 33 3303 1363
contact@crispmind.io, @crispmind
https://crispmind.io & softnote.com

```
      Map.from(loadYaml(configTmpl.readAsStringSync())).forEach((key,
value) {
        _params[key] = value;
      });
    }
  }

  Future<URKError?> init(num amount) async {
           this.amount = amount;
    await _getFee();
    userKey = await parser.key;
    if (userKey == null) {
              print('error sessionKey=$userKey');
           return URKError.create(ECR_KEY_NULL);
    }
             print('init 1');
    balance = (await currencyTET.getBalance(userKey, parser.id!)) ?? 0;
    data['parser'] = parser.token;
    data['to'] = currency.ticker.toUpperCase();
    data['id'] = parser.id;
    data['balanceTET'] = balance;

    if (balance < amount){
              return URKError.create(ECR_NOT_FUNDS,'TET');
    }

             if(!TEST_MODE){
                  if (!await checkBalanceOut()){
                                return
URKError.create(ECR_NOT_FUNDS,'Binance');
                  }
             }

    if(fee == 0){
              return URKError.create(ECR_SERVICE_NOT_DATA, 'Fee is
null');
    }

    return null;
  }



     @override
  Future<Operation?> request(String? recipientEthToken) async {
           final op = Operation(parser.id, currencyOutTET.key);
             final userTokenTET = (await Address.getUserAddress(parser,
currencyTET))?.address;
             recipientEthToken??(await Address.getUserAddress(parser,
currency))?.address;
             if(userTokenTET != null){
                  data['addressTET'] = userTokenTET;
                  data['recipient'] = recipientEthToken;
```

US +1 415 670-9800, UK +44 33 3303 1363
contact@crispmind.io, @crispmind
https://crispmind.io & softnote.com

```
                    await op.wait(OperationArea.WALLET,
OperationActions.MIGRATED_IN, amount, data: data);
                    data['execute'] =
Coder().encryptText('${op.id}:${parser.id}:1:${currencyOutTET.key}').asBas
e58;
                    data['reject'] =
Coder().encryptText('${op.id}:${parser.id}:-
1:${currencyOutTET.key}').asBase58;
                    data['operation'] = op.id;
      data['amount'] = amount;
      final user = await UserIdRepository.user(parser.id);
      if(user != null){
        data['email'] = user['email'];
      }
                    print('EmailSend $data');
                    EmailSend.sendTemplateEmail(_moderator, 'Request
#${op.id}', 'request', data);
                    return op;
              } else {
                    error = URKError.create(ECR_PARAM_NOT_SET,'not found
userTokenTET');
              }
              return null;
  }


  Future<int?> migrateOut(String recipientBnbToken, opId) async {
              final userTokenTET = Address.addressFromCache(parser.id!,
currencyTET)?.address;
                  if(userTokenTET != null){
                              print('migrateToEth
userTokenTET=$userTokenTET');
                              // smart contract transfer of TET to the user
Eth token
                              final trBnbTET = await
NodeRepository.request(SendFromToCommand(_scKey!, amount,
currencyOutTET.port, recipientBnbToken));
                              print('migrateToBnb trBnbTET=$trBnbTET');

                              // waiting for the result of the smart
contract execution
                              if (await executionResult(trBnbTET)) {

                                    // if the result is successful, the
transfer of the TET of the Tectum network user to the storage address
                                    final trTET = await
NodeRepository.request(TokenTransferCommand(userKey!, userTokenTET,
_storageTokenTET, amount));
                                    print('migrateToBnb trTET=$trTET');
                                    if (trTET is TokenTransferResponse) {

      Operation(parser.id!,currencyTET.key).success(OperationArea.WALLET,
OperationActions.MIGRATED_OUT, amount);
```