

TOSHIBA

サンプルコード

AWS KMSを利用した BC+用のウォレットの実装例

本資料について

AWS(Amazon Web Services)の鍵管理サービスであるKMS(Key Management Service)を使用して、BC+(DNCWARE Blockchain+)のウォレットを実装する方法を、サンプルコードを示しながら説明します。

KMSの鍵の作成、KMSの動作確認

KMSの鍵の作成方法と動作確認について

環境設定、ウォレットの実装

RSA鍵およびECDSA鍵の場合のウォレット実装方法について

KMS利用回数の削減

KMSの利用料を節約するための、ファイルウォレットとKMSウォレットの併用方法について

「DNCWARE」、「DNCWARE Blockchain+」は、東芝デジタルソリューションズ株式会社の日本またはその他の国における登録商標または商標です。
Amazon Web Services、AWS、その他のAWS マークは、Amazon.com, Inc. またはその関連会社の商標です。
その他、本資料に掲載の会社名もしくは 商品名等は、それぞれ各社が商標として使用している場合があります。

- 1 KMSの鍵の作成
- 2 KMSの動作確認
- 3 環境設定
- 4 ウォレットの実装
- 5 ウォレットの実装(ECDSAの場合)
- 6 KMS利用回数の削減

1

KMSの鍵の作成

KMSの鍵の作成

概要

- ◆ ウォレットに利用するKMSの鍵を作成する手順を示します。
- ◆ ここでは、AWSコンソールを利用して作成する手順を示します。

※すでに作成された鍵がある場合には、この手順をスキップしても構いません。

- ◆ KMSのスタートページなどから[Create a key]をクリックします。

The screenshot shows the AWS Key Management Service (KMS) console homepage. At the top, it says "Security, Identity & Compliance". The main heading is "AWS Key Management Service" in large white text on a dark blue background. Below this, it says "Easily create keys and control encryption across AWS and beyond". A paragraph of text describes KMS as a managed service for creating and managing keys. On the right side, there is a "Get started now" section with a "Create a key" button. Below that is a "Pricing" section with a "Learn more" link. At the bottom, there is a "How it works" section with a paragraph of text, and a "Getting started" section with a "What is AWS Key Management Service" link.

Security, Identity & Compliance

AWS Key Management Service

Easily create keys and control encryption across AWS and beyond

AWS Key Management Service (KMS) is a managed service that makes it easy for you to create and manage keys and control the use of encryption across a wide range of AWS services. KMS is a secure and resilient service that uses FIPS 140-2 validated hardware security modules to isolate and protect your keys.

Get started now

You can create a key by clicking the button below.

[Create a key](#)

Pricing

[Learn more](#)

How it works

AWS KMS helps you centrally manage and securely store your keys. You can generate keys in AWS KMS or import them from your own key management infrastructure. You can submit data directly to AWS KMS to be encrypted, or decrypted. Other AWS services can use your keys on your behalf to protect data encryption keys that they use to protect

Getting started

[What is AWS Key Management Service](#)

鍵の作成2

◆ 構成の画面で、以下のとおり選択します。

Key type: Asymmetric

Key usage: Sign and verify

Key spec: RSA_2048

◆ [Next]をクリックします。

Configure key

Key type [Help me choose](#)

☐ Symmetric

A single key used for encrypting and decrypting data or generating and verifying HMAC codes

☒ Asymmetric

A public and private key pair used for encrypting and decrypting data, signing and verifying messages, or deriving shared secrets

Key usage [Help me choose](#)

☐ Encrypt and decrypt

Use the key only to encrypt and decrypt data.

☒ Sign and verify

Key pairs for digital signing
Uses the private key for signing and the public key for verification.

☐ Key agreement

Key pairs for deriving shared secrets.

Key spec [Help me choose](#)

Choose the type of key material in your signing key.

☒ RSA_2048

☐ RSA_3072

☐ RSA_4096

☐ ECC_NIST_P256

☐ ECC_NIST_P384

☐ ECC_NIST_P521

☐ ECC_SECG_P256K1

► Advanced options

Cancel

Next

鍵の作成3

- ◆ 任意のエイリアス名を入力します。
この例では、testkey1とします。
このときのKeyIdは、
alias/testkey1となります。
(このKeyIdは、後で使います)
- ◆ [Next]をクリックします。

Add labels

Alias
You can change the alias at any time. [Learn more](#)

Alias

testkey1

Description - optional
You can change the description at any time.

Description

Description of the key

Tags - optional

You can use tags to categorize and identify your KMS keys and help you track your AWS costs. When you add tags to AWS resources, AWS generates a cost allocation report for each tag. [Learn more](#)

This key has no tags.

Add tag

You can add up to 50 more tags.

Cancel Previous Next

鍵の作成4

◆ 鍵の管理権限を任意に設定します。

この例では、なにも設定しません。

◆ [Next]をクリックします。

Define key administrative permissions

Key administrators (53)

Choose the IAM users and roles who can administer this key through the KMS API. You may need to add additional permissions for the users or roles to administer this key from this console. [Learn more](#)

Search Key administrators

< 1 2 3 4 5 6 >

<input type="checkbox"/>	Name ▾	Path ▾	Type ▾
<input type="checkbox"/>	AWSCodePipelineServiceRole-...	/service-role/	Role
<input type="checkbox"/>	AWSDataLifecycleManagerDef...	/service-role/	Role
<input type="checkbox"/>	AWSServiceRoleForAccessAnal...	/aws-service-role/access-analy...	Role
<input type="checkbox"/>	AWSServiceRoleForAmazonGu...	/aws-service-role/guardduty.a...	Role
<input type="checkbox"/>	AWSServiceRoleForAmazonIns...	/aws-service-role/inspector.a...	Role
<input type="checkbox"/>	AWSServiceRoleForAmazonMa...	/aws-service-role/macie.amaz...	Role
<input type="checkbox"/>	AWSServiceRoleForAmazonSSM	/aws-service-role/ssm.amazon...	Role
<input type="checkbox"/>	AWSServiceRoleForAWSLicens...	/aws-service-role/license-man...	Role
<input type="checkbox"/>	AWSServiceRoleForClientVPN	/aws-service-role/clientvpn.a...	Role
<input type="checkbox"/>	AWSServiceRoleForClientVPN...	/aws-service-role/clientvpn-co...	Role

Key deletion

☒ Allow key administrators to delete this key.

Cancel

Previous

Next

鍵の作成5

◆ 鍵の使用権限を設定します。

この例では、
ウォレットを利用するEC2インスタンスの
IAMロールを指定します。

◆ [Next]をクリックします。

Define key usage permissions

Key users (53)

Select the IAM users and roles that can use the KMS key in cryptographic operations. [Learn more](#)

< 1 2 3 4 5 6 >

<input type="checkbox"/>	Name ▾	Path ▾	Type ▾
<input type="checkbox"/>	AWSCodePipelineServiceRole-...	/service-role/	Role
<input type="checkbox"/>	AWSDataLifecycleManagerDef...	/service-role/	Role
<input type="checkbox"/>	AWSServiceRoleForAccessAnal...	/aws-service-role/access-analy...	Role
<input type="checkbox"/>	AWSServiceRoleForAmazonGu...	/aws-service-role/guardduty.a...	Role
<input type="checkbox"/>	AWSServiceRoleForAmazonIns...	/aws-service-role/inspector.a...	Role
<input type="checkbox"/>	AWSServiceRoleForAmazonMa...	/aws-service-role/maciek.amaz...	Role
<input type="checkbox"/>	AWSServiceRoleForAmazonSSM	/aws-service-role/ssm.amazon...	Role
<input type="checkbox"/>	AWSServiceRoleForAWSLicens...	/aws-service-role/license-man...	Role
<input type="checkbox"/>	AWSServiceRoleForClientVPN	/aws-service-role/clientvpn.a...	Role
<input type="checkbox"/>	AWSServiceRoleForClientVPN...	/aws-service-role/clientvpn-co...	Role

Other AWS accounts

Specify the AWS accounts that can use this key. Administrators of the accounts you specify are responsible for managing the permissions that allow their IAM users and roles to use this key. [Learn more](#)

Add another AWS account

Cancel

Previous

Next

鍵の作成6

◆ レビュー画面で設定を確認します。

◆ [Finish]をクリックします。

Review

Key configuration

Key type Asymmetric	Key spec RSA_2048	Key usage Sign and verify
Origin AWS KMS	Regionality Single-Region key	

You cannot change the key configuration after the key is created.

Alias and description

Alias testkey3	Description -
-------------------	------------------

Tags

Key	Value
No data No tags to display	

Key policy

To change this policy, return to previous steps or edit the text here.

```
1 1
2 2 "Id": "key-consolepolicy-3",
3 3 "Version": "2012-10-17",
4 4 "Statement": [
5 5   {
6 6     "Sid": "Enable IAM User Permissions",
7 7     "Effect": "Allow",
8 8     "Principal": {
9 9       "AWS": "arn:aws:iam::837394877448:root"
10 10    },
11 11     "Action": "kms:*",
12 12     "Resource": "*"
13 13   },
14 14   {
15 15     "Sid": "Allow use of the key",
16 16     "Effect": "Allow",
17 17     "Principal": {
18 18       "AWS": "arn:aws:iam::837394877448:role/EC2Default"
19 19     },
20 20     "Action": [
21 21       "kms:DescribeKey",
22 22       "kms:GetPublicKey",
23 23       "kms:Sign",
24 24       "kms:Verify"

```

JSON Ln 1, Col 1 Errors: 0 Warnings: 0

Cancel Previous **Finish**

2

KMSの動作確認

概要

- ◆ AWSのEC2インスタンス上からKMSの鍵が利用できるか、動作を確認します。
- ◆ このセクションでの操作は、EC2インスタンス上で行います。
 - EC2インスタンスにはawsコマンドがインストールされている前提です。
(なお、awsコマンドは動作確認用途のみに使用し、本番動作には必要ありません)

KMSの鍵のパーミッションの確認1

◆ KeyIdを指定して、公開鍵を取得できるかを確認します。

```
$ aws kms get-public-key --key-id alias/testkey1
{
  "KeyId": "arn:aws:kms:ap-northeast-1:837394877440:key/7b6effee-5611-4a38-bf05-60209f522b9d",
  "PublicKey":
"MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEASgESjC+pGdVdZTg+4NjFiTSMt2iZqUXn6h5elh/T2laE6NEx3Qt1xZ8xdCtkV
3vPtCKRne2oErgoinpOCIUvXNIox4QVdnldAbXigzyRSOMFMskZ3DRSLsb9gkxof+RJ7sNF4PNS/r9PZUVaPJJFxU1EulvIE9nHIGcRR9Zlr
gDR7bYrBXeJdhp8yzkP0r8s+xCSOsFC6DEtg+dxYTJNaZtUPcqxa6caDG3xAH7Q9xNQjC6BveEH2XFSf+YnTNPWEGaOYLOjv9gLJsr3lt
s4r2p2vQaWs9xYtqe2AWzJk3vDANHxH1OvXMHSkzTeW82tgb9H/MuCodojn+BcklR7swIDAQAB",
  "CustomerMasterKeySpec": "RSA_2048",
  "KeySpec": "RSA_2048",
  "KeyUsage": "SIGN_VERIFY",
  "SigningAlgorithms": [
    "RSASSA_PKCS1_V1_5_SHA_256", ... 中略 ...
  ]
}
```

◆ 下記のようなエラーが出る場合は、鍵のKey usersを適切に設定してください。 たとえば、EC2インスタンスにIAMロールを割り当て、 そのロールを鍵のKey usersに追加します。

```
An error occurred (AccessDeniedException) when calling the GetPublicKey operation: User:
arn:aws:sts::837394877440:assumed-role/EC2Default/i-00bbfa53328392ba3 is not authorized to perform: kms:GetPublicKey on
resource: arn:aws:kms:ap-northeast-1:837394877440:key/7b6effee-5611-4a38-bf05-60209f522b9d because no identity-based
policy allows the kms:GetPublicKey action
```

KMSの鍵のパーミッションの確認2

◆ 続いて、デジタル署名ができるかを確認します。

```
$ aws kms sign --key-id alias/testkey1 --signing-algorithm RSASSA_PKCS1_V1_5_SHA_256 --message "abcdefgh"
{
  "KeyId": "arn:aws:kms:ap-northeast-1:837394877440:key/7b6effee-5611-4a38-bf05-60209f522b9d",
  "Signature":
    "VnTLP0PNYQQHZ4rr8ny5jHxMAu2FtSINrhUNFD0lgmZ4TGFfCf8k4wfA0q8HGGrjPJCJwNPUTBHXhQ3cCtmA+b5h+AicLOVM+sso7cv1KVeWvNLeyZ/foLK
    Ps+fndb5lZrkJTCLJ2vp0SwyY5yMi+bKckkvsqVTZsZSGIZWhE6z1AkcnG0bQWcLuACLvamcozKV62NV089SW0yLi+u8THJgEHSUaThUo6eN6H4P0ygBLt
    ealnKHiCl8bYEmSxfk8S8uLrYK9NY+Xi/LP1MaIDvUH7TBoUIVqo5jwqesq8mPUR8ASawmGNycCwum2apPysXpQYPQjfH1hSOAfrorUYcQ==",
  "SigningAlgorithm": "RSASSA_PKCS1_V1_5_SHA_256"
}
```

注) --messageの後にはbase64エンコードされた署名対象データを渡しています。
ここではパーミッションの確認だけが目的なので、意味のないデータ"abcdefgh"を渡しています。

◆ エラーが出る場合は、鍵の設定を見直してください。

3

環境設定

概要

- ◆ AWSのEC2インスタンス上に環境設定をします。
- ◆ このセクションでの操作は、EC2インスタンス上で行います。
 - EC2インスタンスにはNode.jsがインストールされている前提です。
 - この作業に専用のディレクトリを作成し、その下で実行してください。

- ◆ Node.jsのバージョンを確認します（version20以降を推奨）

```
$ node --version  
v20.12.2
```

- ◆ aws-sdkのKMSクライアントとasn1.jsをインストールします。

```
$ npm install @aws-sdk/client-kms asn1.js  
  
up to date, audited 83 packages in 2s  
  
found 0 vulnerabilities
```

- ◆ BC+のAPIライブラリ "dncware-blockchain-nodejs-api.js"
をカレントディレクトリに配置します

※BC+のAPIライブラリの入手方法については別途お問い合わせください

動作環境の確認(Node.jsからKMSへの接続の確認)

◆ 下記内容のテストプログラム "test1.mjs" をカレントディレクトリに配置します。

```
import { KMSClient, SignCommand } from "@aws-sdk/client-kms";
var client = new KMSClient({region: "ap-northeast-1"});
var command = new SignCommand({
  KeyId: "alias/testkey1",
  Message: Buffer.from("xyz"),
  MessageType: "RAW",
  SigningAlgorithm: "RSASSA_PKCS1_V1_5_SHA_256",
});
var response = await client.send(command);
console.log(response);
```

←東京リージョンの場合

←KeyIdを指定する

- new KMSClient()の引数のregionは、必要に応じて変更してください。
- new SignCommand()の引数のKeyIdは、適切な値に変更してください。

◆ test1.mjsを実行し、KMSの署名動作を確認します。

```
$ node test1.mjs
{
  '$metadata': {
    httpStatusCode: 200,
    requestId: 'e82ac45e-45d6-4622-94c8-605f3ce8f4db', ... 中略 ...
  },
  KeyId: 'arn:aws:kms:ap-northeast-1:837394877440:key/7b6effee-5611-4a38-bf05-60209f522b9d',
  Signature: Uint8Array(256) [ 48, 69, 2, 32, 38, 195, 1 ... 中略 ... ],
  SigningAlgorithm: 'RSASSA_PKCS1_V1_5_SHA_256'
}
```

4

ウォレットの実装

概要

- ◆ AWSのEC2インスタンス上に、KMSの鍵を利用するウォレットを実装します。
 - 通常のファイルウォレットを使った場合について、コードの理解があることを前提に説明しています。
- ◆ このセクションでの操作は、EC2インスタンス上で行います。
 - 前のセクション「環境設定」を完了し、同じディレクトリの下で実行してください。
- ◆ 動作確認として、トランザクションを発行するサンプルコードを動かします。
 - 動作確認用のブロックチェーンとしてトライアル環境に接続しに行きます。
 - トライアル環境のURLは `https://trial*.dncware-blockchain.biz` です。
 - トライアル環境が使用できない場合、動作確認用のブロックチェーンを適宜に変更してください。

- ◆ 下記内容のモジュール "kms-wallet.mjs" をカレントディレクトリに配置します。

```
import { KMSClient, GetPublicKeyCommand, SignCommand } from "@aws-sdk/client-kms";
import * as crypto from "crypto";
function decodeBase64url(x) {
  return Buffer.from(x.replace(/-/g, '+').replace(/_/g, '/'), 'base64');
}
export async function importPublicData_r({ KeyId, region }) {
  var client = new KMSClient({ region });
  var command = new GetPublicKeyCommand({ KeyId });
  var response = await client.send(command);
  var keyobj = crypto.createPublicKey({ key: response.PublicKey, format: 'der', type: 'spki' });
  var jwk = keyobj.export({ format: 'jwk' });
  return decodeBase64url(jwk.n);
}
export async function signSignature_r({ KeyId, region }, Message) {
  var client = new KMSClient({ region });
  var command = new SignCommand({ KeyId, Message, MessageType: 'RAW', SigningAlgorithm: 'RSASSA_PKCS1_V1_5_SHA_256' });
  var response = await client.send(command);
  return response.Signature;
}
```

サンプルコードの実行

◆ 下記内容のサンプルコード "test-wallet.mjs" をカレントディレクトリに配置します。

```
import * as api from './dncware-blockchain-nodejs-api.js';
import { importPublicData_r, signSignature_r } from './kms-wallet.mjs';
api.pluginExternalWalletModule('aws-kms', { importPublicData_r, signSignature_r });
var uw = await api.importSigningWallet('rs', { external: 'aws-kms', region: 'ap-northeast-1', KeyId: 'alias/testkey1' });
console.log('address:', uw.address);

var rpc = new api.RPC('trial.dncware-blockchain.biz');
rpc.connect('https://trial1.dncware-blockchain.biz');
rpc.connect('https://trial2.dncware-blockchain.biz');
rpc.connect('https://trial3.dncware-blockchain.biz');
var resp = await rpc.call(uw, 'c1query', { type: 'dashboard' });
console.log(resp);
```

←東京リージョンの場合

←KeyIdを指定する

←接続先がトライアル環境の場合

- importSigningWallet()の引数のregionとKeyIdは、適切な値に変更してください。
- RPC(), rpc.connect()の引数は、必要に応じて、実際に接続するブロックチェーンのものに変更してください。

◆ test-wallet.mjsを実行し、ウォレットの動作を確認します。

```
$ node test-wallet.mjs
address: rkRa96LAQ7KBzcuFZUm2rJtALNgfHc
{
  txid: 'xSHEsfb42WB9Md5Ans5TpYTgizqDaFutFBRfo6VhQTsDz',
  status: 'ok',
  value: { ... 省略 ... }
}
```

◆ 前ページのtest-wallet.mjsをコメント(赤字)で解説します。

```
// BC+のAPIライブラリをインポートします
import * as api from './dncware-blockchain-nodejs-api.js';

// KMSウォレットのモジュールをインポートします
import { importPublicData_r, signSignature_r } from './kms-wallet.mjs';

// KMSウォレットのモジュールを外部ウォレットとしてAPIライブラリに登録します
api.pluginExternalWalletModule('aws-kms', { importPublicData_r, signSignature_r });

// KMSの鍵を利用したウォレットを読み込みます。ウォレットの型として'rs'を指定しています。(RSA鍵なので)
var uw = await api.importSigningWallet('rs', { external: 'aws-kms', region: 'ap-northeast-1', KeyId: 'alias/testkey1' });

// ウォレット・アドレスを表示します
console.log('address:', uw.address);

// 以降の手順は、通常のファイルウォレットの場合と同じです

// ブロックチェーンに接続します
var rpc = new api.RPC('trial.dncware-blockchain.biz');
rpc.connect('https://trial1.dncware-blockchain.biz');
rpc.connect('https://trial2.dncware-blockchain.biz');
rpc.connect('https://trial3.dncware-blockchain.biz');

// ブロックチェーンにトランザクションを要求します
var resp = await rpc.call(uw, 'c1query', { type: 'dashboard' });
console.log(resp);
```


5

ウォレットの実装(ECDSAの場合)

概要

- ◆ RSAの場合と概略は同じです。
- ◆ KMSの鍵は、Key spec: ECC_NIST_P256で作成します。
 - この例では、鍵のエイリアスはtestkey2としています。
- ◆ 橢円暗号鍵用のモジュールkms-wallet-ec.mjsは次ページに示します。
- ◆ 橢円暗号鍵用のサンプルコードtest-wallet-ec.mjsは次々ページに示します。

注) 橢円暗号鍵のKMSの利用料は、RSA鍵の場合と異なります。

- ◆ 下記内容のモジュール "kms-wallet-ec.mjs" をカレントディレクトリに配置します。

```
import { KMSClient, GetPublicKeyCommand, SignCommand } from "@aws-sdk/client-kms";
import * as crypto from "crypto";
import { default as asn1 } from "asn1.js";
var ECDerSignature = asn1.define('ECDerSignature', function() {
  this.seq().obj(
    this.key('r').int(),
    this.key('s').int()
  );
});

function decodeBase64url(x) {
  return Buffer.from(x.replace(/-/g, '+').replace(/_/g, '/'), 'base64');
}

export async function importPublicData_e({ KeyId, region }) {
  var client = new KMSClient({ region });
  var command = new GetPublicKeyCommand({ KeyId });
  var response = await client.send(command);
  var keyobj = crypto.createPublicKey({ key: response.PublicKey, format: 'der', type: 'spki' });
  var jwk = keyobj.export({ format: 'jwk' });
  return Buffer.concat([decodeBase64url(jwk.x), decodeBase64url(jwk.y)]);
}

export async function signSignature_e({ KeyId, region }, Message) {
  var client = new KMSClient({ region });
  var command = new SignCommand({ KeyId, Message, MessageType: 'RAW', SigningAlgorithm: 'ECDSA_SHA_256' });
  var response = await client.send(command);
  var { r, s } = ECDerSignature.decode(Buffer.from(response.Signature), 'der');
  return Buffer.concat([r.toBuffer('be', 32), s.toBuffer('be', 32)]);
}
```

サンプルコードの実行

◆ 下記内容のサンプルコード "test-wallet-ec.mjs" をカレントディレクトリに配置します。

```
import * as api from './dncware-blockchain-nodejs-api.js';
import { importPublicData_e, signSignature_e } from './kms-wallet-ec.mjs';
api.pluginExternalWalletModule('aws-kms', { importPublicData_e, signSignature_e });
var uw = await api.importSigningWallet('es', { external: 'aws-kms', region: 'ap-northeast-1', KeyId: 'alias/testkey2' });
console.log('address:', uw.address);
```

←東京リージョンの場合

←KeyIdを指定する

```
var rpc = new api.RPC('trial.dncware-blockchain.biz');
rpc.connect('https://trial1.dncware-blockchain.biz');
rpc.connect('https://trial2.dncware-blockchain.biz');
rpc.connect('https://trial3.dncware-blockchain.biz');
var resp = await rpc.call(uw, 'c1query', { type: 'dashboard' });
console.log(resp);
```

←接続先がトライアル環境の場合

- importSigningWallet()の引数のregionとKeyIdは、適切な値に変更してください。
- RPC(), rpc.connect()の引数は、必要に応じて、実際に接続するブロックチェーンのものに変更してください。
- pluginExternalWalletModule()の引数の名前が、RSA鍵の場合と変わっていることに注意してください。

◆ test-wallet-ec.mjsを実行し、ウォレットの動作を確認します。

```
$ node test-wallet-ec.mjs
address: eeP2Wf5CVbT2UiWhHVSSVKdAWwvEAu

txid: 'xSHEsfb42WB9Md5Ans5TpYTgizqDaFutFBRfo6VhQTsDz',
status: 'ok',
value: { ... 省略 ... }
}
```

6

KMS利用回数の削減

概要

◆ 背景の説明

- BC+の場合、リードアクセス、書き込みアクセスの両方ともトランザクションにデジタル署名を行います。
- リードアクセスにまでKMSを利用するとコストがかさむ可能性があります。

◆ 1つのユーザに2つのウォレットを設定して、KMSの利用回数を減らす運用を紹介します。

- 運用として、ファイルウォレットとKMSウォレットを併用し、書き込みの時のみKMSウォレットを利用します。
- 書き込みの時だけKMSを利用するので、KMSの利用料を節約できます。
- BC+のマルチシグ機能を設定します。
 - リードアクセスには、ファイルウォレットを使う運用とします。
 - 書き込みアクセスには、両方のウォレットが必要となります。
- 書き込みは、マルチシグ・トランザクションとなるため、プログラミングが少し複雑になります。

◆ 事前に必要なもの

- BC+の通常のファイル形式のウォレット（ファイルウォレットと呼びます）
- そのウォレットが紐づけられたBC+のユーザ
- そのユーザの管理権限を持つウォレット（以降、管理ウォレットと呼びます）
- セクション4で説明したRSA鍵のKMSを利用するウォレット（KMSウォレットと呼びます）
- 動作確認用のスマートコントラクト(例ではtest@sampleという名前のもの；中身のコードは空でよい)

◆ ここで示す手順の概略

- KMSウォレットのアドレスをユーザに追加する
- ユーザのマルチシグを2に設定する
- ファイルウォレットでのライトがinsufficient signersエラーになることを確認する
- 両方のウォレットのマルチシグでライトが可能であることを確認する

KMSウォレットアドレスの登録

◆ 管理GUIを使ってアドレスを登録する方法を説明します。

- 前のセクションで説明したtest-wallet.mjsの実行結果から、KMSウォレットのアドレスを取得します。
- 対象のユーザの管理権限を有する管理ウォレットで管理GUIにログインします。
- 対象のユーザのプロファイル画面を表示します。
- Wallets欄の[Edit]をクリックします。
- ダイアログにKMSウォレットのアドレスを入力して、[Add]をクリックします。
- [Submit]をクリックします。

DNCWARE/BlockChain3 demo.dncware-blockchain.com

WALLET: eUBXm9auCeyLXYUZcSoPwsZx32Wkc Log Out

User Profile

u59051626

Configuration

Name	Edit	test-user
Description	Edit	
Wallets	Edit	eDtt8izA6gsQ3obMUrTC84PgzcJtC
Callable to	Edit	all
Disclosed to	Edit	
Multi SIG	Edit	1
Domain		@samples1

DNCWARE/BlockChain3

Edit Wallets

test-user@samples1

Wallets

eDtt8izA6gsQ3obMUrTC84PgzcJtC

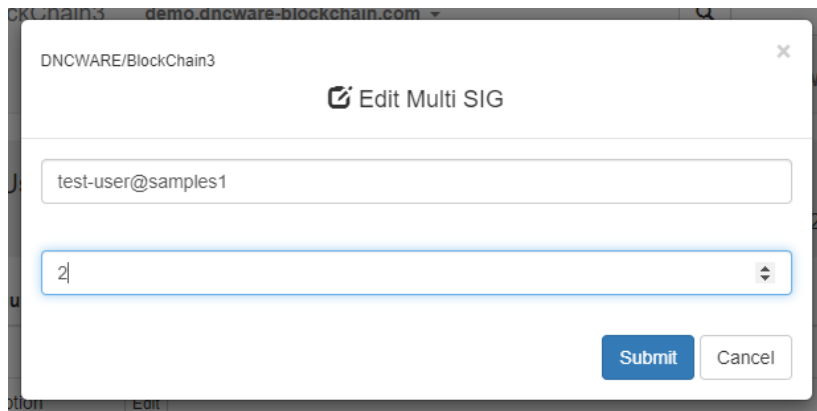
rkRa96LAQ7KBzcuFZUm2rJtALNgfHq

Add

Submit Cancel

ユーザのマルチシグの設定

- ◆ 次に、ユーザのマルチシグを 2 に変更します。
 - Multi SIG欄の[Edit]をクリックします。
 - ダイアログに 2 を入力して、[Submit]をクリックします。
- ◆ 以上で、設定は完了です。



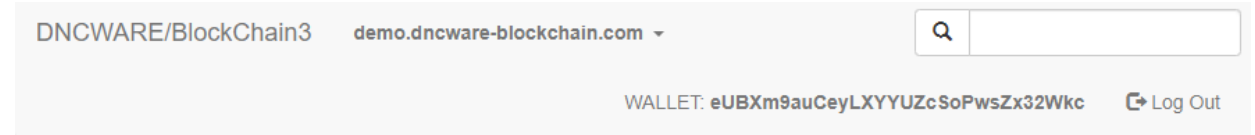
DNCWARE/BlockChain3

Edit Multi SIG

test-user@samples1

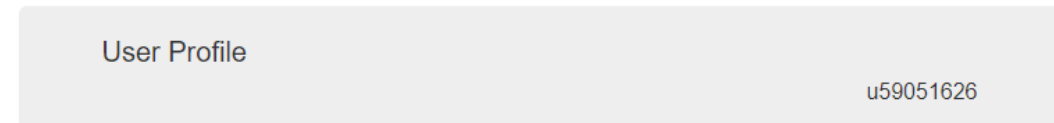
2

Submit Cancel



DNCWARE/BlockChain3 demo.dncware-blockchain.com

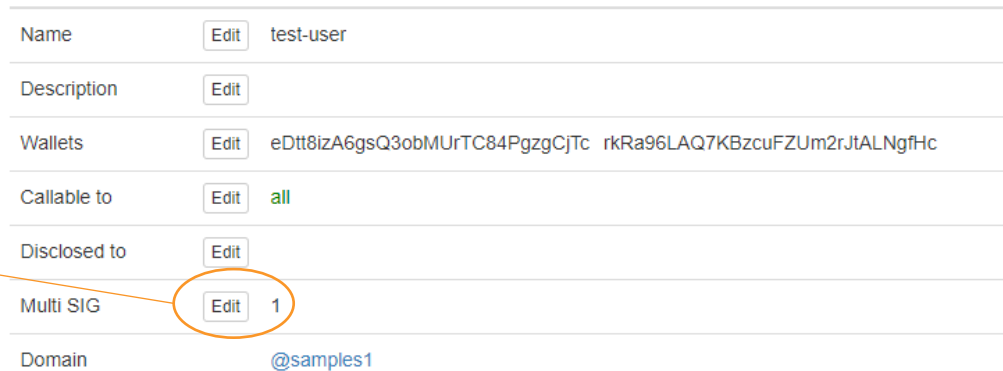
WALLET: eUBXm9auCeyLXYUZcSoPwsZx32Wkc Log Out



User Profile

u59051626

Configuration



Name	Edit	test-user
Description	Edit	
Wallets	Edit	eDtt8izA6gsQ3obMUrTC84PgzcJTC rkRa96LAQ7KBzcuFZUm2rJtALNgfHc
Callable to	Edit	all
Disclosed to	Edit	
Multi SIG	Edit	1
Domain		@samples1

動作確認 1

◆ ファイルウォレット単独でのリードとライトの動作を確認します。

- カレントディレクトリにファイルウォレット "user-wallet.json" を配置します。
- 次ページのテストプログラム "test-single.mjs" をカレントディレクトリに配置し、実行します。

```
$ node test-single.mjs
address: eDtt8izA6gsQ3obMUrTC84PgzcJtC
read: {
  txno: 6438,
  txid: 'xqe5sDM7d9uvRuQYJcnhLhUuqGTw2ydKaevCbnmVnLV8t',
  status: 'read',
  value: null
}
write: {
  txno: 6438,
  txid: 'xA889dxRY3CeKPxtBpzdTRzfx7DYwg5mrApMx648xBBwFB',
  status: 'denied',
  value: 'insufficient signers for multisig'
}
```

- リードアクセスは成功し(status:'read')、ライトアクセスはエラーとなることを確認します。
- 'insufficient signers for multisig' は、マルチシグの署名が不足していることを示しています。

テストプログラム1

◆ test-single.mjsの内容

```
import * as api from './dncware-blockchain-nodejs-api.js';
import * as fs from 'fs';

// ファイルウォレットを読み込みます。ウォレットを開くためのパスワードは適切な文字列に変更してください。
var wf = await api.parseWalletFile(fs.readFileSync('user-wallet.json', 'utf-8'));
var uw = await api.unlockWalletFile(wf, 'wallet-password');
console.log('address:', uw.address);

// ブロックチェーンに接続します。接続先は適切なものに変更してください。
var rpc = new api.RPC('trial.dncware-blockchain.biz');
rpc.connect('https://trial1.dncware-blockchain.biz');
rpc.connect('https://trial2.dncware-blockchain.biz');
rpc.connect('https://trial3.dncware-blockchain.biz');

// リードアクセスでスマートコントラクトを呼び出します。呼び出すコントラクトと引数を適切なものに変更してください。
var contract = 'test@sample';
var args = { /* if any */ };
var resp = await rpc.call(uw, contract, args, { readmode: 'fast' });
console.log('read:', resp);

// ライトアクセスで同じスマートコントラクトを呼び出します。(署名不足のエラーになります)
var resp = await rpc.call(uw, contract, args);
console.log('write:', resp);
```

動作確認2

◆ 続いて、マルチシグでのリードとライトの動作を確認します。

- 次ページのテストプログラム "test-multi.mjs" をカレントディレクトリに配置し、実行します。

```
$ node test-multi.mjs
address1: eDtt8izA6gsQ3obMUrTC84PgzgCjTc
address2: rkRa96LAQ7KBzcuFZUm2rJtALNgfHc
write: {
  txno: 6439,
  txid: 'xYqWLSQSL8vYBhpmKYtBSXMno8MXgJTgcFdfnyYhcmtup',
  status: 'ok',
  value: null
}
```

- マルチシグでのライトアクセスが成功（status:'ok'）することを確認します。

◆ test-multi.mjsの内容

```
import * as api from './dncware-blockchain-nodejs-api.js';
import * as fs from 'fs';

// ファイルウォレットを読み込みます。パスワードは適切な文字列を設定してください。
var wf = await api.parseWalletFile(fs.readFileSync('user-wallet.json', 'utf-8'));
var uw1 = await api.unlockWalletFile(wf, 'wallet-password');
console.log('address1:', uw1.address);

// KMSウォレットを読み込みます。regionとKeyIdは適切に設定してください。
import { importPublicData_r, signSignature_r } from './kms-wallet.mjs';
api.pluginExternalWalletModule('aws-kms', { importPublicData_r, signSignature_r });
var uw2 = await api.importSigningWallet('rs', { external: 'aws-kms', region: 'ap-northeast-1', KeyId: 'alias/testkey1' });
console.log('address2:', uw2.address);

// ブロックチェーンに接続します。接続先は適切なものを設定してください。
var rpc = new api.RPC('trial.dncware-blockchain.biz');
rpc.connect('https://trial1.dncware-blockchain.biz');
rpc.connect('https://trial2.dncware-blockchain.biz');
rpc.connect('https://trial3.dncware-blockchain.biz');

// トランザクション要求を作成します。呼び出すコントラクトと引数を適切なものに変更してください。
var contract = 'test@sample';
var args = { /* if any */ };
var request = api.createRequest([uw1.address, uw2.address], contract, args);

// つづいて、両方のウォレットで署名します。(マルチシグ署名)
await api.signRequest(request, uw1, rpc.chainID);
await api.signRequest(request, uw2, rpc.chainID);

// トランザクション要求を送信し、スマートコントラクトを呼び出します。
var resp = await rpc._call_request(request);
console.log('write:', resp);
```

TOSHIBA