

# Weldentity 的多签及限量 CPT 的实现

---

本文档描述的合约均适用于：

FISCO BCOS / CITA。

## Weldentity 的多签及限量 CPT 的实现

- 0. 项目特性
- 1. Weldentity 介绍
- 2. Weldentity 升级点
- 3. Weldentity 合约结构与升级部分示意图
- 4. 权限控制（多签）的简介、接口描述与使用示范
  - 4.1 合约接口描述
  - 4.2 合约使用示范
- 5. 限量 CPT 的简介、接口描述与使用示范
  - 5.1 合约接口描述
  - 5.2 合约使用示范
- 6. 总结与未来计划

## 0. 项目特性

---

- 智能合约创意新颖、设计巧妙和通用性强

Weldentity 方案基于 Solidity 编写，适用于许多需要数字身份的场景，如教育、金融等。

本项目写的多签方案与 CredentialController 同理。

其中，**CredentialController** 为独立模块，因此不仅适用于 **Weldentity** 方案，也适用于其它基于 **Solidity** 的 **DID** 方案。

- 智能合约函数、事件、结构等设计规范性、合理性、执行效率与安全性

本项目遵循 Solidity 开发规范，且每一个新增函数均包含测试，符合 TDD 原则。

- 代码编写简洁规范、可读性强，注释清晰准确

保证每一个函数均有标准注释，且变量命名清晰规范，具有高可读性。

- 设计说明文档规范、结构完整、内容详细准确

设计文档完整，包括其实现功能、结构设计、流程设计、使用演示与未来规划。

- 多底层框架版本

不仅适用于 FISCO BCOS，还适用于 CITA 等使用 EVM 的联盟链。

## 1. Weldentity 介绍

---

Weldentity是一套分布式多中心的技术解决方案，可承载实体对象（人或者物）的现实身份与链上身份的可信映射、以及实现实体对象之间安全的访问授权与数据交换。Weldentity由微众银行自主研发并完全开源，秉承公众联盟链整合资源、交换价值、服务公众的理念，致力于成为链接多个垂直行业领域的分布式商业基础设施，促进泛行业、跨机构、跨地域间的身份认证和数据合作。

目前主要包括两大模块：**Weldentity DID** 以及 **Weldentity Credential**，一个是去中心化身份，另一个是链上凭证。

Weldentity 使用 Solidity 进行合约开发，主要包括以下两个部分：

- **Weldentity DID智能合约**，负责链上 ID 体系建立，具体包括生成 DID（Distributed Identity）、生成 DID Document、DID 在链上的读取与更新。
- **Weldentity Authority智能合约**，负责进行联盟链权限管理，具体包括链上 DID 角色的定义、操作与权限的定义与控制。

## 2. Weldentity 升级点

要讲 Weldentity 合约使用于正式生产环境，还有一些地方需要升级。

- **权限控制方向**

目前 Weldentity 合约的权限控制方案：

操作	一般DID	Authority Issuer	Committee Member	Administrator
增删改Administrator	N	N	N	Y
增删改Committee Member	N	N	N	Y
增删改Authority Issuer	N	N	Y	Y
发行授权CPT	N	Y	Y	Y

在实际生产环境中，我们需要包括多签功能的更精准的权限控制。

在进行修改后，权限控制方案如下：

操作	一般DID	Authority Issuer	Committee Member	Administrator
增删改 Adminnistrator	N	N	N	Y
增删改 Committee Member	N	N	N	Y
增删改 Authority Issuer	N	N	Y（可多签）	Y
发行授权 CPT	N	Y	Y	Y

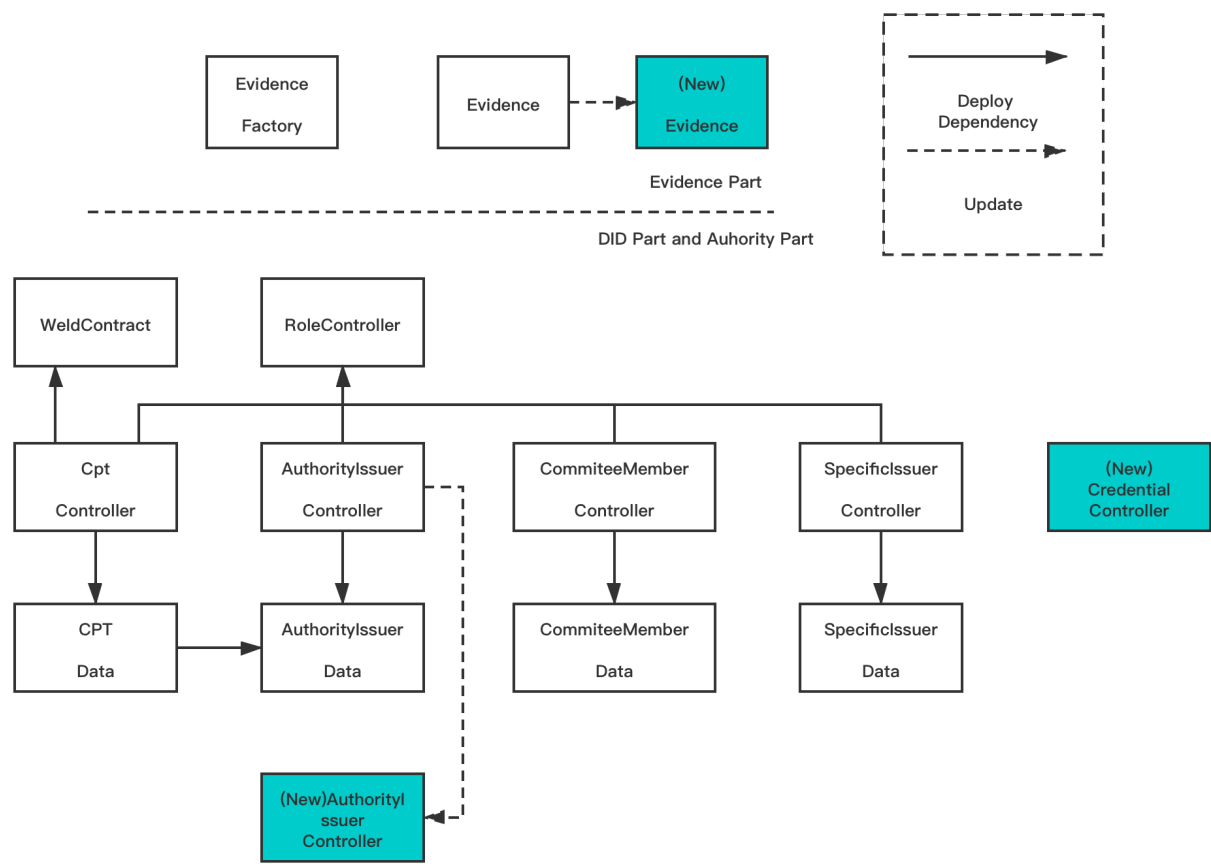
我们通过对 `AuthorityIssuer.sol` 合约模块的升级，来实现多签的功能。

- **限量发行方向**

在当前的流程中，我们会通过合约定义 CPT（Claim Protocol Type，凭证的声明类型），然后根据自定义的 CPT 模板发布 Credential（可验证的数字凭证）。

Credential 和 ERC721 有类似之处，因此我们可以对发行 Credential 的流程进行升级，并加入新的合约，使其支持某一类型的 CPT 「限量发行」。例如某合格证书只发行一千个。

### 3. Weldidentity 合约结构与升级部分示意图



### 4. 权限控制（多签）的简介、接口描述与使用示范

`AuthorityIssuerController` 合约主要功能是对 AuthorityIsser 角色的增加、删除、修改与查询，本次升级在增加中加入多签功能。

#### 4.1 合约接口描述

- **addAuthorityIssuer** 功能：新增 AuthorityIssuer

类型：原函数修改

参数：

变量名	类型	含义
addr	Address	新增authorityIssuer的地址
attribBytes32	bytes32[16]	名字
attribInt	int[16]	创建日期
accValue	bytes	值（预留，目前无用）

返回值： 无

- **signTransaction 功能：** 实现多签功能

类型： 新增函数

参数：

变量名	类型	含义
transactionId	uint	需要进行多签的交易 id

返回值： 无

- **getTxIDNeedMultiSigNum 功能：** 查询某交易还需要的签名数

类型： 新增函数

参数：

变量名	类型	含义
transactionId	uint	需要查询的交易id

返回值：

变量名	类型	含义
TxIDNeedMultiSigNum	uint	查询的交易还需要的多签数量

- **getPendingTransactions 功能：** 获取当前所有需要被多签的交易

类型： 新增函数

参数： 无

返回值：

变量名	类型	含义
PendingTransactions	uint[]	当前所有需要被多签的交易

- **setTxIdMultiSig 功能：** 设置某交易的多签数量

类型： 新增函数

说明： 因为该函数决定了新增 Authority Issuer 所需要的 Committee Member 数量，因此调用该函数需要 Administrator 权限【[见2](#)】。

参数：

变量名	类型	含义
transactionId	uint	需要设置的交易id
minNumber	uint	要求的多签数量

返回值：无

## 4.2 合约使用示范

### 1. 使用账户1部署合约

Environment

JavaScript VM

✎ VM (-) ⓘ

Account ⓘ

0xca3...a733c (99.9999999999993168 ⓘ)

📄

Gas limit

3000000

⌵

Value

0

wei ⓘ

AuthorityIssuerController ⓘ

Deploy

^

authorityIssue

0xb05df6af0073fd592d8f6ee3d5c8ac0ff12e3967

rDataAddress:

roleController

Address:

0xdc77b866fe07451e8f89871edb27b27af9f2afc

📁

transact

### 2. 部署成功

```
• Checking transactions details and start debugging.
• Running JavaScript scripts. The following libraries are accessible:
  ◦ web3 version 1.0.0
  ◦ ethers.js
  ◦ swarmgw
  ◦ compilers - contains currently loaded compiler
• Executing common command to interact with the Remix interface (see list of commands above). Note that these commands can also be included and run from a JavaScript script.
• Use exports/.register(key, obj)/.remove(key)/.clear() to register and reuse object across script executions.
```

```
creation of AuthorityIssuerController pending...
```

✓

[vm] from:0xca3...a733c to:AuthorityIssuerController.(constructor) value:0 wei data:0x608...a733c logs:0 hash:0x9ab...1c47c

Debug ▼

```
> |
```

### 3. 测试新增 authorityIssuer 函数

addAuthorityIssuer

^

addr:

attribBytes32:

attribInt:

accValue:

transact

其中，address 是新增为 authorityIssuer 的账户地址，attribBytes32 是名字，attribInt 是创建的日期。

#### 4. 填写参数，进行测试

transact后，该交易会被放入等待交易队列中，结果返回的交易序号为3。

logs	<pre>[   {     "from": "0xb87213121fb89cbd8b877cb1bb3ff84dd2869cfa",     "topic": "0x207dd9a87ea6a0b199619879b1b0b939bcfd3d101b402ce4887e6f6733bb37ff",     "event": "TransactionIdLog",     "args": {       "0": "3",       "id": "3",       "length": 1     }   } ]</pre>
value	0 wei

#### 5. 测试查询等待队列函数

由于上述交易逻辑上会被保存进等待队列，因此这里我们去查看一下等待队列中是否真正被放入了该交易：

getPendingTransactions

0: uint256[]: 3

查询结果与逻辑无误。


#### 6. 设置多签要求

在多签之前，我们需要先为该等待交易设置多签数量，若不设置，则无法进行多签，这里我们为上述等待交易设置多签限制为2。

setTranIDMultiSig

transactionId: 3

minNumber: 2

 transact

## 7. 测试多签函数

设置完成后，我们使用当前账户（具备权限）对上述交易序号为3的交易进行多签：

logs	[ { "from": "0xb87213121fb89cbd8b877cb1bb3ff84dd2869cfa", "topic": "0x06197bb1fbb2c0ae31e90ea52d83d878773267fd41dac321a5ba22e145bcd23", "event": "signTransactionLog", "args": { "0": "3", "1": "1", "id": "3", "neednumber": "1", "length": 2 } } ]
value	0 wei

结果返回签名成功，我们再切换其他具有权限的账户进行签名，这里是否具有权限是通过这段代码进行判断。

```
if(!roleController.checkPermission(tx.origin,roleController.MODIFY_AUTHORITY_ISSUER())) {  
  return;  
}
```

✓ 0xca3...a733c (99.999999999997718267 ether)

0x147...c160c (100 ether)

0x4b0...4d2db (100 ether)

0x583...40225 (100 ether)

0xdd8...92148 (100 ether)

使用第二个有权限的账户签名成功，由于已满足多签要求，因此新增成功：

logs	<pre>[   {     "from": "0xb87213121fb89cbd8b877cb1bb3ff84dd2869cfa",     "topic": "0xfcb730e1916430caf8b1752daaa14b59e59354b89170fde494afa6a5f1190fa6",     "event": "AuthorityIssuerRetLog",     "args": {       "0": "0",       "1": "1",       "2": "0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c",       "operation": "0",       "retCode": "1",       "addr": "0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c",       "length": 3     }   } ]</pre>
------	--

## 5. 限量 CPT 的简介、接口描述与使用示范

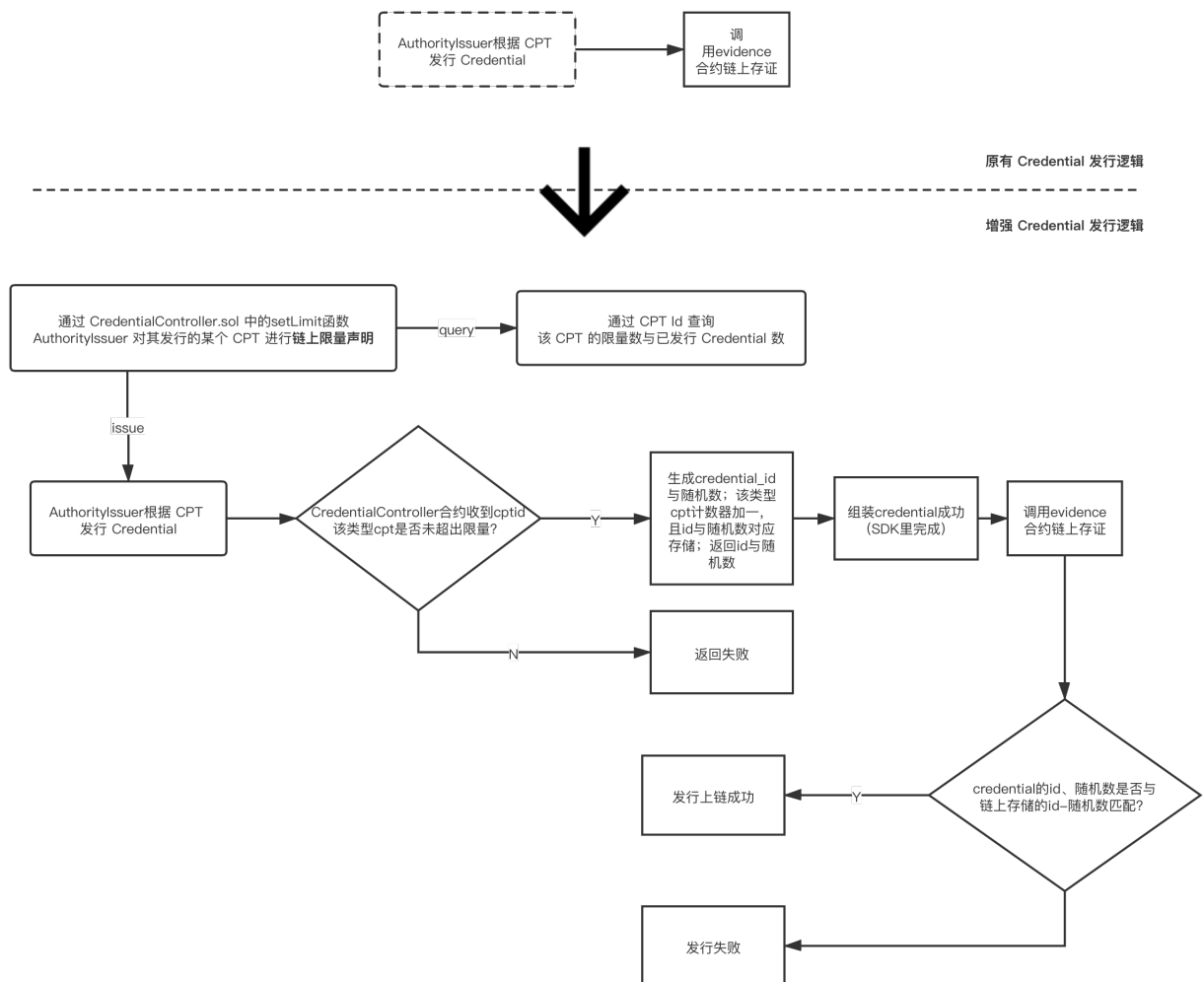
1) `CredentialController` 为全新合约，主要功能为：

- 声明某 Id 的 Cpt 的 Credential 的发行数量，并存储。
- 发行 Credential 时生成需要的 ID 号与随机数，并存储。
- 提供如下查询：
  - 通过 CPT Id 查询某 CPT 类型的总数及已发行数量。
  - 通过 Credential Id 查询某 Credential 的相应随机数。

2) `EvidenceContract` 为旧合约的升级，主要功能是凭证上链，目前对 `createEvidence` 函数进行增强，加入 `creid` 与 `randNum` 两个参数，以实现链上验证的目的。

流程从链下发行 Credential，然后再通过 evidence 进行链上存证，转变为更复杂的「链上限量声明 → 带验证的发行 → **Credential** 的合法性与限量 CPT 的总量 / 已发行量可链上查询」逻辑，以实现某一类型的 CPT 限量发行。





## 5.1 合约接口描述

- **generateIdAndRandNum** 功能：生成 credentialID 号与随机数

类型：新增函数

参数：

变量名	类型	含义
cptid	uint	要使用的 cpt 的编号

返回值：

变量名	类型	含义
creid	uint	生成的 credential 编号
randNum	uint	生成的随机数

- **setLimit** 功能：设置某类型cpt的可发行的限量数额

类型：新增函数

参数：

变量名	类型	含义
cptid	uint	要使用的 cpt 的编号
limitNum	uint	该类型cpt限量发行的额数

返回值：无

- **getCptLimitAndIssuedNum** 功能：获取某类型 cpt 的可发行的限量数额与已发行 credential 的数量

类型：新增函数

参数：

变量名	类型	含义
cptid	uint	要查询的 cpt 的编号

返回值：

变量名	类型	含义
limitNumber	uint	查询的 cpt 类型的可发行限量数额
issuedNumber	uint	查询的 cpt 类型的已发行 credential 的数量

- **getRandNumOfCreid** 功能：获得某 credential 中的随机数

类型：新增函数

参数：

变量名	类型	含义
creid	uint	要查询的 credential 编号

返回值：

变量名	类型	含义
randNum	uint	查询的credential的随机数

- **createEvidence** 功能：存证上链

类型：原函数修改

参数：

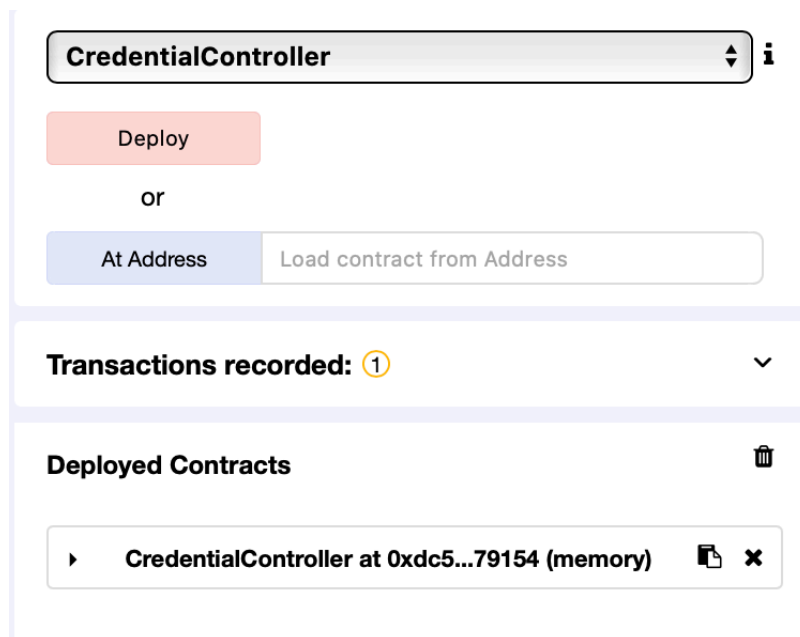
变量名	类型	含义
hash	string	查询存证的key
sig	string	发布存证的人的签名
extra	string	任意额外数据
updated	uint256	当前event的更新时间
creid	uint	某类型的cpt编号（新增变量）
randNum	uint	随机数（新增变量）

返回值：

变量名	类型	含义
result	uint	上链成功与否的反馈值

## 5.2 合约使用示范

### 1. 部署CredentialController合约



### 2. 为某类型cpt设置限量数额

CPT 限量数额默认为零，也就是说如果 CPT 遵循限量发行流程（自然也可以选择传统流程），该 CPT 无法「发行」Credential，必须在设置后，方可发行。

setLimit
^

cptid:

limitNum:


transact

设置成功


decoded input	<pre>{   "uint256 cptid": "1",   "uint256 limitNum": "1" }</pre>
decoded output	<pre>{}</pre>
logs	<pre>[]</pre>
value	<pre>0 wei</pre>

### 3. 查看某类型cpt限额

这里我们查询刚刚设置的cptid为1的类型的限额

getCptLimitAndIssuedNum
^

cptid:


transact

结果为我们刚刚设置的限额1，以及当前已发行的credential数量为0

logs	<pre>[   {     "from": "0xeall1a2d0f914282b07e449f09b4eaddc1ccae5be",     "topic": "0x7f9ea0f7fac995b277ab208233052e72fd544c86cd72f3186e180f5cc86f88c7",     "event": "limitNumberLog",     "args": {       "0": "1",       "1": "1",       "2": "0",       "cptid": "1",       "limitNum": "1",       "IssueNumber": "0",       "length": 3     }   } ]</pre>
------	--

### 4. 生成 Credential Id与随机数

填写cptid，基于该类型生成一对 credential Id 与随机数

**generateIdAndRandNum**

cptid:

1



transact

生成成功，得到 credential Id 与随机数，该id与随机数一一匹配，用于后续 Evidence 上链时的合法检验。

decoded output	<pre>{   "0": "uint256: 1",   "1": "uint256: 7334" }</pre>
----------------	--

## 5. 部署 EvidenceContract 合约

将此前部署的CredentialController合约地址填入，进行部署

**Deploy**

CredentialCon  
trollerAddress:

0xcca417069d356fcf870f4fe05b7d95bf700de960



transact

## 6. 存证上链

其中creid为要上链的credential的id号，randNum 为此前发行 credential 时得到的与id号一一配对的随机数。

**createEvidence**

hash:

string

sig:

string

extra:

string

updated:


uint256

creid:

uint256

randNum:

uint256



transact

填写参数，randNum处先随意填一个randNum，由于不配对，因此结果返回为0，上链失败。

decoded output	<pre>{   "0": "uint256: 0" }</pre>
----------------	------------------------------------

再填写配对的随机数，进行测试，结果返回为1，上链成功。

decoded output	<pre>{   "0": "uint256: 1" }</pre>
----------------	--

## 6. 总结与未来计划

---

- **对 Credential 限量功能进行进一步的完善**

目前限量 Credential 的逻辑是 AuthorityIssuer 在发行 CPT 时进行限量声明。如果进一步，我们可以做到多个 Issuer 协商 Credential 总量，之后每个特定的 Issuer 有特定的份额。

- **对 Credential 功能进行进一步的增强**

例如 Credential 间的联动。

- **更全面更细化的权限控制**

- 可以在更多方面引入多签。
- 引入更细化的权限控制，以应对实际生产环境中的复杂场景。