海联金汇 HYUNION HOLDINGS
股票代码：002537

UMF 联动优势 UNION MOBILE FINANCIAL TECHNOLOGY
海联金汇旗下企业

科技赋能普惠金融

基于 Hyperledger Indy 的分布式数字身份

王宇

MAKE FINTECH EVERYWHERE

核心问题

为什么要进行这次分享？

1、同步一个信息，利用相关的开源项目，我们已经能够支持数字身份认证相关的能力（先用起来）

2. 从实际代码的流程上更加明确数字身份认证的相关操作

遗留问题

Hyperledeger Indy 使用 RBFT 共识，交易使用 Merkle Tree 的数据结构，State 使用 Merkle Patrical Tree、RocksDB 等区块链系统常见的技术维护。

但是 Indy 没有块信息（块高度、块哈希）、交易信息（交易哈希）等常用于可视化的内容和接口。

遗留问题：

1、Indy 的节点算区块链吗？

2.、什么样的系统和软件算是区块链？分界线是什么？

相关项目


HYPERLEDGER
uport
sovrin
identity for all
Bloom
JOLOCOM
HODL HELPER

目录

启动节点

```
[mpsp@tlsca ~]$ docker ps --format="table {{.Image}}\t{{.Status}}\t{{.Ports}}\t{{.Names}}"
IMAGE               STATUS              PORTS                                       NAMES
indy_pool           Up 28 minutes       10.10.144.118:9701-9708->9701-9708/tcp      vigorous_grothendieck
[mpsp@tlsca ~]$
[mpsp@tlsca ~]$ docker exec -it vigorous_grothendieck /bin/bash
indy@6e68dcb54754:/$
indy@6e68dcb54754:/$ ps -ef
UID          PID  PPID  C STIME TTY          TIME CMD
indy           1     0  0 04:38 pts/0    00:00:00 /usr/bin/python /usr/bin/supervisord
indy           9     1 14 04:38 pts/0    00:04:04 /usr/bin/python3 /usr/local/bin/start_indy_node Node1 0.0.0.0 9701 0.0
indy          10     1 13 04:38 pts/0    00:03:57 /usr/bin/python3 /usr/local/bin/start_indy_node Node3 0.0.0.0 9705 0.0
indy          11     1 13 04:38 pts/0    00:03:59 /usr/bin/python3 /usr/local/bin/start_indy_node Node2 0.0.0.0 9703 0.0
indy          12     1 13 04:38 pts/0    00:03:59 /usr/bin/python3 /usr/local/bin/start_indy_node Node4 0.0.0.0 9707 0.0
indy        2085     0  0 05:07 pts/1    00:00:00 /bin/bash
indy        2099  2085  0 05:07 pts/1    00:00:00 ps -ef
indy@6e68dcb54754:/$
```

反正是启动起来了

节点配置（自动生成）


```
{
    "reqSignature": {},
    "txn": {
        "data": {
            "data": {
                "alias": "Node1",
                "blskey": "4N8aUNHSgjQVgkpm8nhNEfDf6txHznoYREg9kirm
                "blskey_pop": "RahHYiCvoNCtPTrVtP7nMC5eTYrsUA8WjXbd
                "client_ip": "10.10.144.118",
                "client_port": 9702,
                "node_ip": "10.10.144.118",
                "node_port": 9701,
                "services": [
                    "VALIDATOR"
                ]
            },
            "dest": "Gw6pDLhcBcoQesN72qfotTgFa7cbuqZpkX3Xo6pLhPhv"
        },
        "metadata": {
            "from": "Th7MpTaRZVRYnPiabds81Y"
        },
        "type": "0"
```


用来连接节点

客户端

SDK支持的语言：

- Java
- Python
- iOS
- NodeJS
- .Net
- Rust

后面用 Python 做示例

建立连接

```
# 创建连接池
poolName = "pool"
await pool.create_pool_ledger_config(
    poolName,
    json.dumps({"genesis_txn": "config.txn"})
)
# 获取操作句柄
poolHandler = await pool.open_pool_ledger(poolName, None)
```

config.txn 是节点配置文件的路径

目录

场景

小明是一个大学生，正在找工作，找工作需要学校的成绩单证明在校成绩

小明 - 普通用户

学校 - 发证方

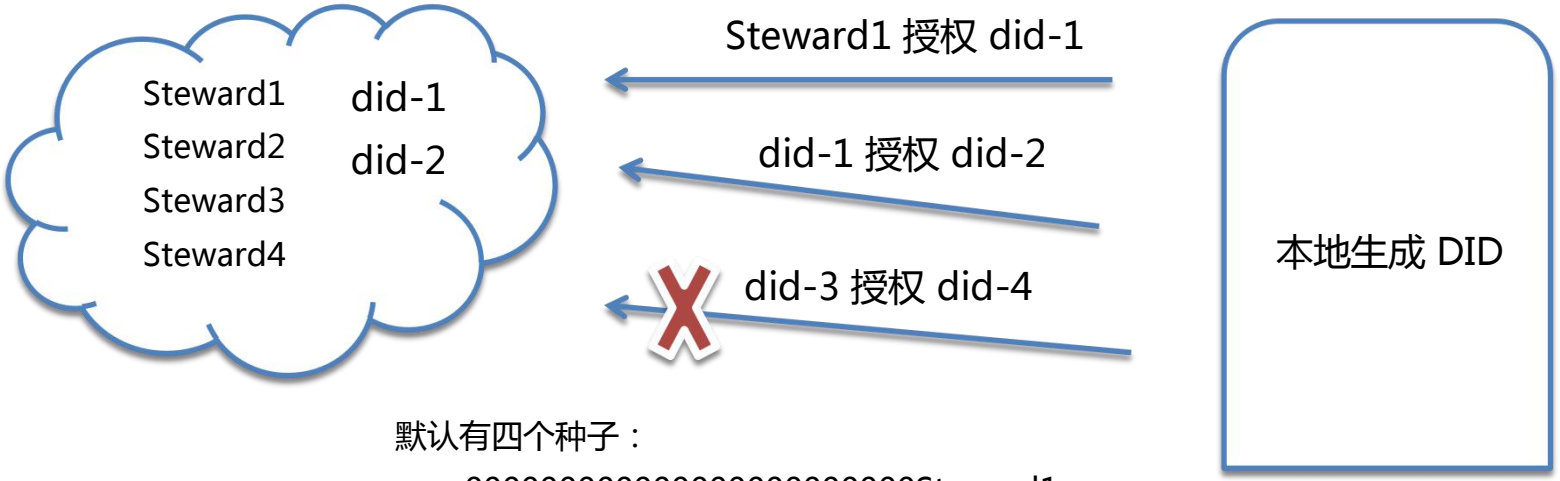
企业 - 验证方

UMF 联动优势 UNION MOBILE FINANCIAL TECHNOLOGY

区块链节点

Indy 的权限控制

新的 DID 加入需要已经在节点上的 DID 授权

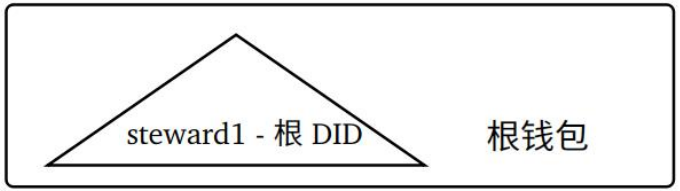


默认有四个种子：

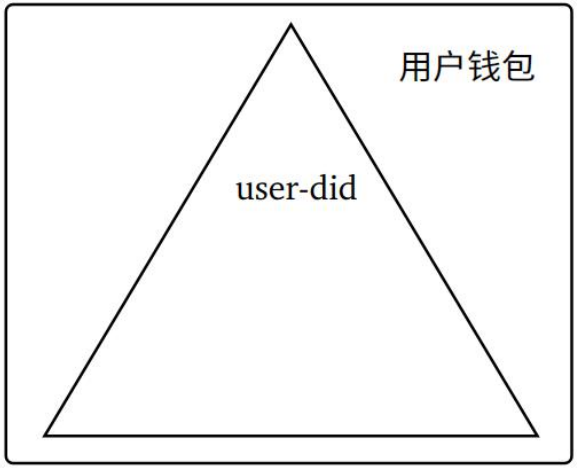
000000000000000000000000Steward1
000000000000000000000000Steward2
000000000000000000000000Steward3
000000000000000000000000Steward4

场景分析

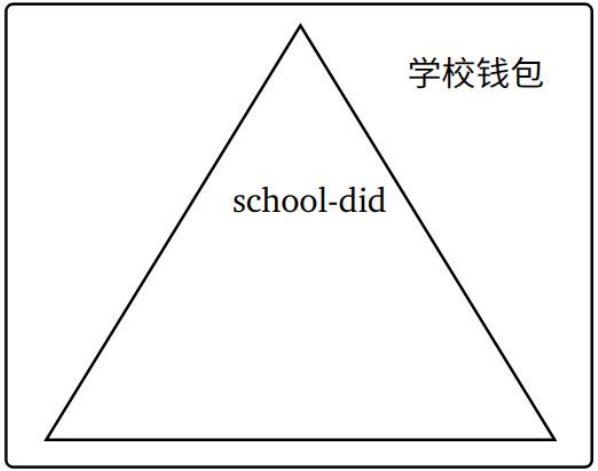
区块链节点



用户



发证方（学校）



验证方（企业）

* 验证方只需要验证
* **验证可以离线完成**

创建钱包和 DID

```
# 创建钱包
await wallet.create_wallet(
    json.dumps({"id": "wallet"}),
    json.dumps({"key": "password"})
)
# 获得操作句柄
walletHandler = await wallet.open_wallet(
    json.dumps({"id": "wallet"}),
    json.dumps({"key": "password"})
)
```

```
# 创建 DID
adminDid, adminKey = await did.create_and_store_my_did(
    walletHandler,
    json.dumps({"seed": "000000000000000000000000Steward1"})
)
```

用户和发证方同理

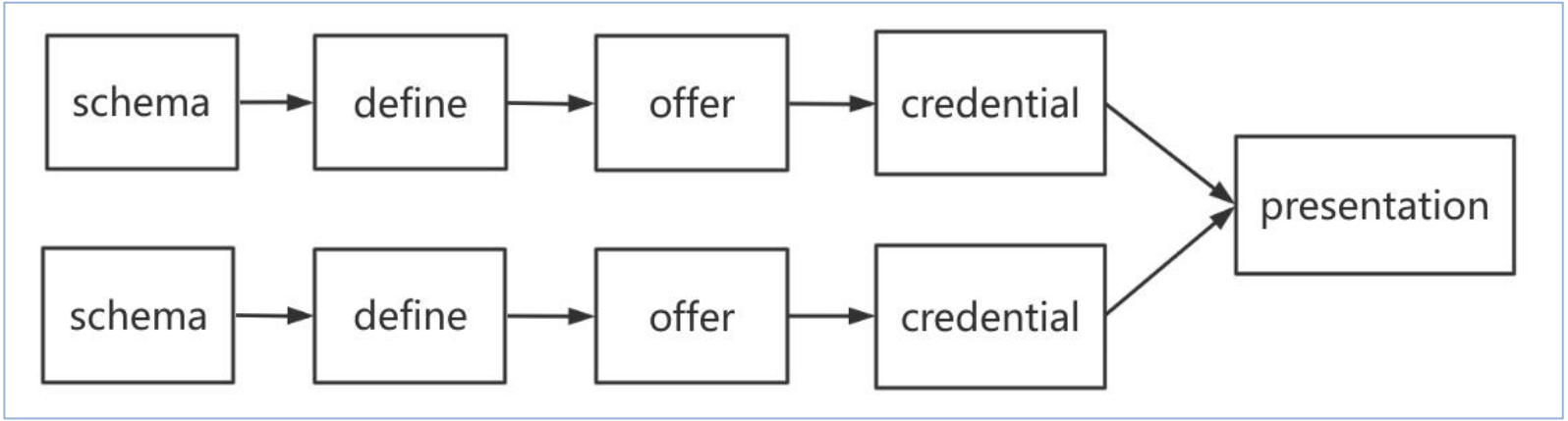
授权 DID

```
# 构建交易请求
nym_request = await ledger.build_nym_request(
    adminDid,
    userDid,
    userKey,
    None,
    "TRUST_ANCHOR"
)
# 提交交易
signResult = await ledger.sign_and_submit_request(
    poolHandler,
    walletHandler,
    adminDid,
    nym_request
)
```

发证方同理

目录

一张证明经历的变化



- schema	数据结构，定义包含哪些字段	发证方生成
- define	证明的定义，包括使用的签名算法	发证方生成
- offer	提供给用户填的	用户使用
- credential	发证方已经签发的证明	用户储存
- presentation	用户用于展示的证明	用户生成

设计字段，创建 schema

```
schemaId, schemaContent = await anoncreds.issuer_create_schema(
    schoolDid,
    "schemaName",
    "0.3",
    json.dumps(["name", "age", "score"])
)
schemaRequest = await ledger.build_schema_request(
    schoolDid,
    schemaContent
)
signResult = await ledger.sign_and_submit_request(
    poolHandler,
    schoolWalletHandler,
    schoolDid,
    schemaRequest
)
```

这里定义了三个字段：

- name
- age
-score

查询 schema

```
get_schema_request = await ledger.build_get_schema_request(
    schoolDid,
    schemaId
)
get_schema_response = await ledger.submit_request(
    poolHandler,
    get_schema_request
)
schemaId, schemaContent = await ledger.parse_get_schema_response(
    get_schema_response
)
```

过渡操作，从节点查出刚才创建的 schema

创建证明的定义（define）

```
credDefId, credDefContent = await anoncreds.issuer_create_and_store_credential_def(
    schoolWalletHandler,
    schoolDid,
    schemaContent,
    "TAG1",
    "CL",
    json.dumps({"support_revocation": False})
)
credDefRequest = await ledger.build_cred_def_request(
    schoolDid,
    credDefContent
)
signResult = await ledger.sign_and_submit_request(
    poolHandler,
    schoolWalletHandler,
    schoolDid,
    credDefRequest
)
```

证明的结构包含签名算法的定义

从 define 生成 offer

```
credOffer = await anoncreds.issuer_create_credential_offer(
    schoolWalletHandler,
    credDefId
)
```

用户将使用这里生成的 offer 进行后续操作

用户填写申请

```
userSecretId = await anoncreds.prover_create_master_secret(
    userWalletHandler,
    None
)
credRequest, credMeta = await anoncreds.prover_create_credential_req(
    userWalletHandler,
    userDid,
    credOffer,
    credDefContent,
    userSecretId
)
data = json.dumps({
    "name": {"raw": "user", "encoded": "0"},
    "age": {"raw": "22", "encoded": "22"},
    "score": {"raw": "100", "encoded": "100"}
})
```

用户密钥用来判断所有权

签发、保存证明

```
# 签发证明
credential, _, _ = await anoncreds.issuer_create_credential(
    schoolWalletHandler,
    credOffer,
    credRequest,
    data,
    None,
    None
)
# 用户将证明保存到钱包
storeResult = await anoncreds.prover_store_credential(
    userWalletHandler,
    None,
    credMeta,
    credential,
    credDefContent, None
)
```

目录

验证方发布要求

例如，企业要求：

年龄大于 18

分数高于 90

```
nonce = await anoncreds.generate_nonce()
jobCondition = json.dumps({
    'nonce': nonce,
    'name': 'Job-Application',
    'version': '0.1',
    'requested_attributes': {
        'attr1_referent': {
            'name': 'name',
            'restrictions': [{'cred_def_id': credDefId}]
        }
    },
    'requested_predicates': {
        'predicate1_referent': {
            'name': 'age',
            'p_type': '>',
            'p_value': 18,
            'restrictions': [{'cred_def_id': credDefId}]
        },
        'predicate2_referent': {
            'name': 'score',
            'p_type': '>',
            'p_value': 90,
            'restrictions': [{'cred_def_id': credDefId}]
        }
    }
})
```

用户从钱包中查找证明


```
searchHandler = await anoncreds.prover_search_credentials_for_proof_req(
    userWalletHandler,
    jobCondition,
    None
)
searchResult = await anoncreds.prover_fetch_credentials_for_proof_req(
    searchHandler,
    "attr1_referent",
    100
)
await anoncreds.prover_close_credentials_search_for_proof_req(
    searchHandler
)
credAll = json.loads(searchResult)
```

用户使用证明生成展示（presentation）

可以结合多个证明生成展示（代码下一页）

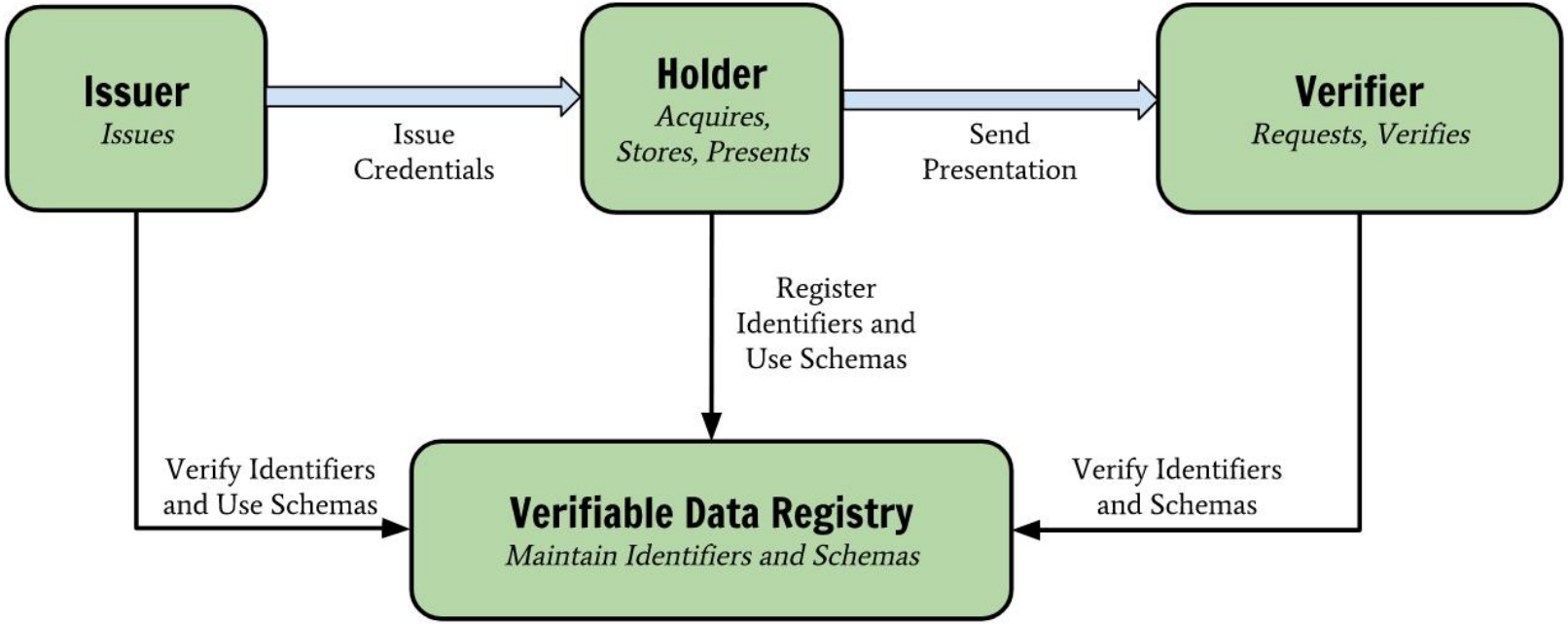
```
credUseReferent = credAll[-1]["cred_info"]['referent']
presentationSchema = json.dumps({
    'self_attested_attributes': {},
    'requested_attributes': {
        'attr1_referent': {'cred_id': credUseReferent, 'revealed': True},
    },
    'requested_predicates': {
        'predicate1_referent': {'cred_id': credUseReferent},
        'predicate2_referent': {'cred_id': credUseReferent},
    }
})
schemaObj = json.dumps({schemaId: json.loads(schemaContent)})
credDefObj = json.dumps({credDefId: json.loads(credDefContent)})
presentation = await anoncreds.prover_create_proof(
    userWalletHandler,
    jobCondition,
    presentationSchema,
    userSecretId,
    schemaObj,
    credDefObj,
    "{}"
)
```

验证方验证 presentation

```
vRes = await anoncreds.verifier_verify_proof(
    jobCondition,
    presentation,
    schemaObj,
    credDefObj,
    '{}',
    '{}'
)
```

验证是离线的

回顾



回顾一下 W3C 的规范中特别经典的这张图

感谢聆听　敬请指正