私密合约部署 选择性部署

客户端发送一个交易,在”选择性部署”内置合约中创建一个合约空间,声明节点列表,用于节点验证
（一）jwt token
token := jwt.NewWithClaims(jwt.SigningMethodES256, jwt.MapClaims{
"ak": "03a643cf8b02b5005b4e48acdc94a8f693355a4eb51292d5508b97f3e6159e3de6",//auth pubKey 部署方的公钥
// "nk": "02595d553697305c7670dfd92628e5ff68080335265edf804aea4e6e8df5112464",//node pubKey
"h":"0x48b2f8d5f59deac09220df20c82d0aab69d51c42fd42ca6bfe878a6c642416f3",//chaincode hash
"t": 17348454325,//timestamp
"n": "eefffefreredffdsuuf2rrfdsmfljljrra",//nonce 随机盐
})
部署方用私钥签名生成token string。

（二）deploy tx(dtx1) 的 payload
payload := &protos.Transaction{
Type: 3, //1-invoke 2-deploy 3-selective-deploy 4-selective-invoke
Payload: deploy,
}

deploy := &protos.Deployment{
Owner: "8000d109DAef5C81799bC01D4d82B0589dEEDb33",
Name: "testcc02",
Version: "1.0.5",
Payload: invocation,
}

invocation := &protos.Invocation{
Fcn: "deploy",
Args: deployInfo,
Meta: map[string][]byte{
"baap-tx-type": []byte("exec"),
},
}

deployInfo := struct {
Jwt string `json:"jwt"` // sk加密后的token
ChaincodeHash string `json:"chaincode_hash"`
Nodes [][]string `json:"nodes"` // 部署策略：节点公钥和encrypted-sk列表
Consensus string `json:"consensus"` // 共识策略 all-全共识 2/3-多数共识
Link string `json:"link"` // 下载地址
AuthPublickey string `json:"auth_public_key"` // 授权方公钥
}{
"ljflasdjflasdjfldasjfadlsfjdsalfjdslflfjasdlfjkdsf",
[][]string{{"node1 publicKey", "encrypted-sk"}, {"node2 publicKey", "encrypted-sk"}, {"node3 publicKey", "encrypted-sk"}},
"all",
"http://pdx.ltd/download",
"dsflsdjfldsjfldsajfdlaksjgklsdfjgdfalsgjdslksadfds"
}

（三）节点安装成功后的deploy-dtx2(dtx2)
选择性存在智能合约名字：selective-deploy-service

tx的to: selective-deploy-service contract address

tx的payload：
type Payload struct {
Fcn string `json:"fcn"` // deploy or exec
Node string `json:"node"` // node public key
CCHash string `json:"cc-hash"` // chaincode hash
TxHash string `json:"tx-hash"` // dtx1-hash
Owner string `json:"owner"` // chaincode owner public key
Name string `json:"name"` // chaincode name
Version string `json:"version"` // chaincode version
Status string `json:"status"` // 0 : deployed, else error code: HTTP code as returned by repo, or 1: failed to connect 2: failed to install
Reason string `json:"reason"` // desc
}

（四）selective-deploy-service合约的存储结构体
合约地址对应的stateObject中存kv：
每次安装成功tx执行该合约的deploy方法时，合约验证节点成功后,向value中追加相应信息。

每次部署tx （dtx1）时，需要保存：

key：chaincode hash + owner + name + version
value: [{"node1 pbulic key","node2 pbulic key","node3 pbulic key"}

存储合约描述
key: chaincode hash + owner + name + version + "desc"
value: {
"owner":"chaincode owner public key",
"name":"chaincode name",
"version":"chaincode version",
"consensus":"all"
}

每次安装成功tx (dtx2) 执行该合约的deploy方法时，合约验证节点成功后,向value中追加相应信息。
key: chaincode hash + owner + name + version + "detail"
value: [{tx2, node, status}]

key: chaincode hash + owner + name + version + "status"
这个不知道有什么用。

（五）调用cc的tx1(etx1)
tx1的to：已经部署成功的chaincode address

tx1的payload：
payload := &protos.Transaction{
Type: 4, //1-invoke 2-deploy 3-selective-deploy 4-selective-invoke
Payload: invocation,
}
invocation := &protos.Invocation{
Fcn: "put",
Args: [][]byte{[]byte("name"), []byte("tonysu")},
Meta: map[string][]byte{
"baap-tx-type": []byte("exec"),
"url-jwt": sk-encrypted jwt token //OPTIONAL
"url-args:" // url to download, OPTIONAL
"sk-node_x: encrypted sk for node-x, used to get url-jwt or get clear-text args
},
}
invocation中的args用sk进行加密，

（六）cc交易完成后发送的tx2 (etx2)

tx2的to: selective-invoke-service contract address

tx2的payload：
type payload struct {
Fcn string `json:"fcn"` // deploy or exec
CCHash string `json:"cc_hash"` // chaincode hash
Tx1Hash string `json:"tx_hash"` // tx1的hash
Root string `json:"root"` // state root hash
}

(七)selective-invoke-service 合约的存储结构体
etx1执行成功之后发送etx2到内置合约，etx2会执行合约的exec方法，该方法会判断该chaincode的执行是否满足共识策略。

例: A(etx2) B(etx2) C(etx2) 共识策略2/3共识
执行A(tx2),不满足策略,验证A节点后是该CC的选择节点后,保存A(tx2)的数据
执行B(tx2),满足策略,验证B节点后是该CC的选择节点后,保存B(tx2)的数据,并且公布共识结果,所有执行B(tx2)选择节点和自己本地CC状态库进行结果比较,相同成功,不同回滚
执行C(tx2),满足策略,不影响最终共识,直接完成交易
超时:如果x块内没有满足共识策略,共识失败,再来tx2不予执行

存储结构体
key: hash(etx1Hash+"exec")
value:[]{"Node1 root Hash","Node2 root Hash","Node3 root Hash"}

如果满足共识策略，存状态数据：
key: tx1 hash
value: state root hash

如果不满足共识策略，存状态数据：
key: tx1 hash
value：fail 或 0x0

流程

部署交易：1/ 选定节点列表；2/ 然后对每一个节点：生成一个JWT，里边有节点的地址和合约哈希，时间戳，随机盐，部署方的公钥地址等，然后部署方签名，然后用节点的公钥和部署方的私钥基于ecdh生成一个密钥（SK），对JWT用SK加密；3/ 这些加密的JWT和部署策略、共识策略，下载地址等组成部署交易，发送给所有节点；4/ 节点收到之后，解密自己的JWT，用自己的

私钥签名JWT，然后从repo里下载安装。5/ 安装成功之后，每个节点发送部署成功交易给”选择性部署”内置合约。6/ 部署方是可以通过查询这个选择性部署内置合约来知道自己的合约是否部署成功。

执行交易：1/ 用户发送交易TX1给列表中的节点，请求执行私密合约。如果交易中有少量私密数据，可以通过生成一个私秘密钥（SK），然后用SK加密数据本身。，再用在用ecdh生成和每个节点的私秘密钥来分别加密SK。如果大量数据，或者不希望链内传输可以像部署时一样处理。2/ 任何节点都可以执行tx1但是不需要修改状态。部署节点针对每一个选择性存在的合约有一个线程顺序执行交易。3/ 部署节点执行成功之后，发送tx2，关联tx1，加上状态哈希，给所有节点上的选择性存在智能合约。4/ 内置的选择性存在智能合约，根据该私密合约的共识策略，决定TX1是否成功。