FUSION FOUNDATION
DCRM YELLOW PAPER

GUOCHANG XU
ZHAOJUN HE
XING CHANG

DCRM yellow paper is a development and use manual published on the basis of the design and implementation of DCRM at present stage. Anyone can participate in the trial, recommendation, development and application development based on DCRM according to the yellow paper and the corresponding current code. The FUSION Foundation and the DCRM R&D team will update the relevant codes and the corresponding contents of the yellow paper in time for better design, implementation schemes and functions in the follow-up development process.

I. Algorithm design of DCRM

Distributed Private Key Control realizes the generation of public-private key pairs and addresses and the transaction signatures on the target blockchain in a distributed manner through several nodes and according to digital signature algorithms adopted by the target blockchain, thus realizing the control and management of accounts and assets on the target blockchain in a distributed manner.

Such a technical route enables DCRM to be compatible with as many digital assets controlled by encryption algorithms as possible, whether these digital assets are generated on a centralized or decentralized basis. Through supporting a signature algorithm with DCRM, a series of encrypted digital assets with the same signature algorithm can be controlled and managed.

At present, most (over 80%) of encrypted digital currencies adopt the same ECDSA [JMV01] signature algorithm as Bitcoin and Ethereum, so DCRM first chooses to implement support for ECDSA signature algorithm. In addition, some encryption currencies use different signature algorithms, such as Stellar's Ed25519 signature algorithm [JL17], and Schnorr's signature algorithm

[S89]. The new signature algorithms will be studied and supported in the next phase of DCRM's development plan.

Next, we will introduce the threshold DSA signature scheme supported by DCRM in a distributed manner and its implementation scheme.

1.1 Overview of threshold DSA signature schemes

This scheme describes how the k nodes can safely cooperate to generate DSA signatures without a trusted third party. The parameter t in the scheme is the privacy threshold, even if the attacker controls t nodes, he can not forge a legitimate signature; if he wants to restore a legitimate signature, at least $t+1$ nodes need to cooperate.

To facilitate understanding and modular programming, several basic cryptographic schemes, such as Cramer-Shoup encryption [CS98], ElGamal encryption [G84] and Paillier encryption [P99], and a series of zero-knowledge proof protocols [BFM88] are at the end of paper. These zero-knowledge proof protocols ensure the security and reliability of the scheme.

In order to ensure the high security of the communication process, the scheme suggests adopting data encapsulation technology to encrypt the communication data in a mixed way to protect the privacy of the data. In the one-to-one communication scenario, AES128 [DR02] symmetric encryption algorithm is proposed to implement data encapsulation mechanism (DEM), and Cramer-Shoup (CS) asymmetric encryption algorithm is proposed to implement key encapsulation mechanism (KEM). In the one-to-many communication scenario (that is, broadcast form), we adopt random multiplexing encryption scheme (RR-MRES) , and use CS encryption algorithm to instantiate it into an efficient KEM method. Because many times of communications are involved in the scheme, we will not elaborate on when to adopt encapsulation technology in the following procedures. It is tacitly approved that the one-to-one communication adopts AES128 + CS for encapsulation, and the broadcast communication adopts AES128 + CS-RR-MRES for encapsulation.

The threshold digital signature scheme has three stages: initialization, distributed key generation and threshold signature generation. The specific protocols and algorithms will be described in detail in sections.

1.2 Scheme design in the initialization stage

The public parameters of DSA signature scheme and the public information contained in the trapdoor commitment scheme [CommitmentWiki] are mainly produced in this stage. The specific steps of the work in the initialization phase are:

1) generate a multiplicative cyclic group G generated by generator g with the order of q, which regards (G, g, q) as a common parameter.

a) randomly select the primary public key $h_c \in G$, and in the specific implementation, choose the hash value mapping on specific public information to avoid questioning on the integrity.

b) randomly select $e \in Z_q$ as the public information for an independent commitment scheme (the hash value for specific public information may be selected for implementation), and publicize (G, g, q) and (h_c, e).

1.3 Scheme design for generation of distributed private key

The scheme of generation of distributed private key describes how to generate a signature key pair in a distributed manner among all nodes, wherein y is public, but x is distributed in each node P_i in the form of shared value x_i, and broadcast $\alpha_i = E(x_i)$ in the network during the process. The generation scheme of distributed key is divided into two stages. First, generate the public-private key pairs (N, d) of the threshold Paillier encryption scheme [HMRT12] in a decentralized and distributed manner, and then generate the ECDSA private key pairs in a distributed manner.

1.4 Scheme design of threshold digital signature

The signature generation scheme is implemented in the case of input of m and success of generating the distributed Paillier private key [GGN16]. Here, we assume that if $\perp$ appears in any commitment or any zero-knowledge proof fails, then the protocol terminates without output.

A. Solution description

Specifically, the threshold digital signature is generated by the following steps:

(a) Randomly select $\rho_i \in Z_q$ for each node P_i, calculate $u_i = E(\rho_i)$, $v_i = \rho_i \times_E \alpha = E(\rho_i x_i)$ and $[C_{1,i}, D_{1,i}] = Com([u_i, v_i])$, and broadcast the value C_i.

(b) Note: $\alpha = E(x) = \prod_{i=0}^{n} \alpha_i = \prod_{i=0}^{n} E(x_i)$, it is given in the former section.

(c) Broadcast the value $D_{1,i}$ for each node P_i , then everyone calculate the $[u_i, v_i] = Ver(C_{1,i}, D_{1,i})$ through validating the algorithm; it is stated in the zero-knowledge proof $\prod_{(1,i)}$ that: there exists $\eta \in [-q^3, q^3]$, and $D(u_i) = \eta$, $D(v_i) = \eta D(E(x))$; the node works out $u = \oplus_{i=1}^{t+1} u_i = E(\rho)$ and $v = \oplus_{i=1}^{t+1} v_i = E(\rho x)$, wherein $\rho = \sum_{i=1}^{t+1} \rho_i$.

(d) Randomly select $k_i \in Z_q$ and $c_i \in [-q^3, q^3]$ for each node, calculate $r_i = \mathrm{g}^{k_i}, w_i = (k_i \times_E u) +_E E(c_i q) = E(k_i \rho + c_i q)$ and $[C_{2,i}, D_{2,i}] = Com(r_i, w_i)$, and broadcast the value $C_{2,i}$.

(e) Broadcast value $D_{2,i}$ for each node P_i , then everyone can calculate $[r_i, w_i] = Ver(C_{2,i}, D_{2,i})$ through validation algorithm; it is stated in the zero-knowledge proof $\prod_{(2,i)}$ that: there exists $\eta \in [-q^3, q^3]$, and $\mathrm{g}^{\eta} = r_i, D(w_i) = \eta D(u) \bmod q$; the node works out $w = \oplus_1^{t+1} w_i = E(k\rho + cq)$, wherein $k = \sum_{i=1}^{t+1} k_i$ and $c = \sum_{i=1}^{t+1} c_i$, and calculate $R = \prod_1^{t+1} r_i = \mathrm{g}^k$ and $r = H'(R) \in Z_q$.

(f) First, all nodes work together to decrypt w through the the distributed decryption protocol $\prod_{TDEC}$ in the threshold Paillier encryption scheme.
The specific implementation process of the agreement $\prod_{TDEC}$ is as follows:

- For $1 \le j \le k$, the node P_j works out $\prod_{i=1}^{k} c_{\sigma_{1,j}}$ and obtains g^{σ_j} through decrypting ElGamal, and adopt $\prod_{\mathrm{EQ}}$ for the zero-knowledge proof.
- For $1 \le j \le k$,the node P_j works out $c_j = c^{2\triangle si}$ on ciphertext c and broadcasts.
- Calculate $c' = \prod_{i \in S} c_i^{2\lambda_{0,i}^{S}} \bmod n^{s+1}$ for each node, wherein $\lambda_{0,i}^{S} = \Delta \prod_{i' \in S \setminus i} \frac{-i}{i-i'} \in Z.$
- Calculate the final plaintext $M = \left(c' \left(4\triangle^2\right)^{-1}\right) \bmod N$ for each node.

And then, calculate for each node:

$$\begin{aligned}\sigma &= \psi \times_E [(m \times_E u) +_E (r \times_E v)] \\ &= \psi \times_E [E(mp) +_E E(r\rho x)] \\ &= (k^{-1}\rho^{-1}) \times_E [E(\rho(m + xr))] \\ &= E(k^{-1}(m + xr)) \\ &= E(s)\end{aligned}$$

Nodes call the distributed decryption protocol $\prod_{TDEC}$ to decrypt ciphertext σ, then make $s = D(\sigma)\ mod\ q$, and finally the node outputs (r, s) as the signature of the message m.

B. Correctness and security

In terms of correctness, under normal circumstances, there are:

① From the step （a） and （b）, it can be known that:

for $i \in \{1, \dots, t\}, p_i \in_\$ Z_q$， $u_i = E(p_i)$， $v_i = \rho_i \times_E \alpha = E(\rho_i x_i)$；

ad $u = \oplus_{i=1}^{t+1} u_i = E(\rho)$； $v = \oplus_{i=1}^{t+1} v_i = E(\rho x)$， wherein $\rho = \sum_{i=1}^{t+1} \rho_i$ 。

② From the step （c） and（d）, it can be known that:

For $i \in \{1, \dots, t\}$， $k_i \in_\$ Z_q$， $c_i \in_\$ [-q^3, q^3]$， $r_i = \mathrm{g}^{k_i}$ ， $w_i = (k_i \times_E u) +_E E(c_i q) = E(k_i \rho + c_i q)$；

And ， $R = \prod_1^{t+1} r_i = \mathrm{g}^k$, wherein $k = \sum_{i=1}^{t+1} k_i$ ， $c = \sum_{i=1}^{t+1} c_i$， $r = H'(R) \in Z_q$ 。

③ Decrypt w, obtain $kp + cq$, and proceed modulo division on q, obtain the $\eta = k\rho\ mod\ q$ and the inverse element $\psi = \eta^{-1}\ mod\ q$. Thus, for the numerical summary of the message, calculate $\sigma = \psi \times_E [(m \times_E u) +_E (r \times_E v)]$ through u and v .That is, Paillier encrypted ciphertext for DSA signature (second component) s . The distributed decryption protocol $\prod_{TDEC}$ decrypts the ciphertext σ （$s = D(\sigma)\ mod\ q$）, and the standard DSA signature can be obtained.

The security of signature schemes is obvious and depends on:

① IND-CPA security of Paillier encryption scheme;

Note: Assuming that $X = \{X_n\}_{n \in \mathbb{N}}$ and $Y = \{Y_n\}_{n \in \mathbb{N}}$ are two collectivities, for any probabilistic polynomial time algorithm D and all positive polynomials, the following formula is tenable for all n which are large enough, then they are called indistinguishable in polynomial time. CPA (Chosen Plaintext Attack) is an attack mode against encryption scheme: the attacker chooses plaintext and obtains corresponding ciphertext produced by encryption service, expecting to use the obtained plaintext-ciphertext to threaten the security of cryptographic scheme. If an encryption scheme does not have an efficient algorithm to generate distinguishable ciphertext even under the chosen plaintext attack, then the signature scheme is called IND-CPA secure.

② EF-ACMA security of DSA signature scheme;

Note: ACMA refers to Adaptively Chosen-Message Attack: An attacker can request the signer to sign any message the attacker needs at any time after he has obtained the signer's public key. EF(Existential Forgery) means giving a verifiable but not necessarily meaningful message

- signature pair.If a signature scheme does not have an efficient algorithm to generate "existential forgery" even under adaptively chosen-message attack, then it is called EF-ACMA secure.

③ The security of the used zero-knowledge proof protocol.

1.5 Encryption scheme and algorithm for implementation

A. Commitment Scheme

Commitment scheme is a cryptographic primitive that allows the user to make a commitment to a selected value and keep it secret from others, and reveal the commitment value at a later time. The design of commitment schemes makes the promisor unable to change after the commitment, that is, the commitment scheme is binding, which have important applications in many cryptographic protocols, including the generation of security random number, zero-knowledge proof and security computing.

In the visual description of the commitment scheme, the sender puts the email containing the promise to the recipient in the password box and gives it to the recipient for safekeeping. The recipient can not open it because he has no unlock key. At the same time, because the password box is kept in the recipient's hands, the contents of the email can not be changed again by the sender. Only after a period of time, the sender chooses to release the unlock key, the promised contents of the e-mail will be displayed. Commitment schemes are conducted in two stages of interaction: the submission stage of choosing and formulating commitment values, and the disclosure stage of displaying and checking commitment values. According to the different cryptographic algorithms chosen to implement commitment scheme, it can be divided into:

- Use the symmetric cryptographic system's commitment scheme;
- Use the one-way function's commitment scheme;
- Use the commitment scheme of pseudo-random sequence generators.

Here, we apply the commitment scheme in order to prevent malicious behavior of nodes in a network of mutual distrust.

Commitment schemes are needed whenever the nodes need to share their intermediate computational values with the outside world. The specific operation is that the nodes first publicly send a commitment Com related to the secret value they master. When the secret value is finally disclosed, other nodes can verify whether the secret value is consistent with the commitment Com. Commitment schemes can avoid the interference of the secret value to the outside world before it is disclosed, and the promisor can not change the secret value.

In the DCRM scheme, through introducing the commitment scheme, we can detect the possible malicious behavior of the node, such as submitting the wrong secret value, or latently collecting the results of other nodes and then carrying out targeted cryptographic attacks. Commitment schemes will be able to detect and restart the agreement before the loss is generated.

B. Paillier encryption scheme

Paillier encryption algorithm, named after Pascal Paillier in 1999, is a probabilistically asymmetric public key encryption algorithm. The difficult problem of its construction lies in the difficulty of calculating the n residue. A prominent feature of Paillier encryption algorithm is the additive homomorphism of its probabilistic encryption, which means that given the ciphertext m_1 and m_2, the ciphertext of $m_1 + m_2$ can be calculated. This plays an important role in electronic voting and other applications. Paillier encryption algorithm has the following characteristics:

- Homomorphism addition in plaintext:

The decryption of the product of two ciphertexts is the sum of their corresponding plaintexts.

$$D\left(E\left(m_1, r_1\right) * E\left(m_2, r_2\right)\ mod\ n^2\right) = m_1 + m_2\ mod\ n$$

- Homomorphism addition in ciphertexts:

A plaintext Paillier becomes the power of another ciphertext after encryption, and the product of two plaintexts is obtained after decryption.

$$D\left(E\left(m_1, r_1\right)^{m_2}\ mod\ n^2\right) = m_1 * m_2\ mod\ n$$

$$D\left(E\left(m_2, r_1\right)^{m_1}\ mod\ n^2\right) = m_1 * m_2\ mod\ n$$

More generally, the constant power of the ciphertext encrypted by a plaintext Paillier is the product of the plaintext and the constant after decryption.

$$D\left(E\left(m_1, r_1\right)^{k}\ mod\ n^2\right) = m_1 * k\ mod\ n$$

Given the ciphertext of two plaintext Paillier after encryption, there is no known method to calculate the corresponding ciphertext to the product of two plaintext without knowing the private key.

Here, we use Paillier encryption scheme to ensure that the values that DCRM completes the distributed ECDSA signature process are calculated based on the encryption state, and it needs to rely on the participation of all nodes to complete the final signature decryption.

C. ElGamal Encryption scheme

In cryptography, ElGamal encryption algorithm is an asymmetric encryption algorithm based on Diffie-Hellman key exchange. It can be defined on any cyclic group G, and its security depends on the discrete logarithm difficult problem of the group G.

ElGamal encryption system is usually used in hybrid encryption system. For example, we use symmetric encryption system to encrypt messages, and then use ElGamal encryption algorithm to pass private key. This is because, at the same level of security, ElGamal encryption, as an asymmetric cryptographic system, is usually slower than symmetric encryption. The private key and the messages to be delivered of symmetric encryption algorithms are usually much shorter, so it's faster to use ElGamal encryption keys and then use symmetric encryption to encrypt messages of any length.

Here, we mainly apply ElGamal encryption scheme to complete the generation steps of Paillier distributed private key.

D. Zero-Knowledge proof

In cryptography, zero-knowledge proof is a practical and widely used protocol. It refers to the ability of the prover to convince the verifier that an assertion is correct without providing any useful information to the verifier.

Zero-knowledge proof is essentially an agreement involving two or more parties, that is, a series of steps that two or more parties need to take to complete a task. The certifier proves to the verifier and makes him believe that he knows owns a certain message, but he can not disclose any information about the proven message to the verifier during the proof process.

Zero-knowledge proof can be divided into interactive zero-knowledge proof and non-interactive zero-knowledge proof, and zero-knowledge proof has different forms according to the content of proof. In this paper, we mainly apply non-interactive zero-knowledge to detect the possible malicious behavior of nodes in an unreliable network. For example, the intermediate results submitted by nodes are not based on the correct private key fragmentation. Before the nodes do evil, the agreement is restarted to ensure the safety of the entire DCRM link and avoid property loss.

1.6 Zero-knowledge proof for implementation

Zero-knowledge proof is the most important part of threshold DSA signature scheme, which is introduced later to facilitate readers to understand the scheme as a whole. Generally, zero-knowledge proof includes two roles: Prover P and Verifier V.

A. Zero-knowledge proof $\prod_{1,i}$

Based on public parameters $c_1, c_2, c_3,$, we construct zero-knowledge proof $\prod_{1,i}$ and prove that if $\eta \in \left[-q^3, q^3\right]$,then $D(c_1) = \eta D(c_2)$ and $D(c_3) = \eta$. Wherein, D is the Paillier decryption algorithm, $\Gamma \in Z^*_{N^2}$ is the public key for Paillier encryption scheme, $\widetilde{N}$ is the distributed generated RSA composite number in this scheme (the wave line is added here to make a symbolic difference).

B. Zero-knowledge proof $\prod_{2,i}$

Based on public parameters $g, r, w,$, we construct zero-knowledge proof $\prod_{2,i}$ and prove that if $\eta_1 \in \left[-q^3, q^3\right], \eta_2 \in \left[-q^8, q^8\right]$, then $g^{n_1} = r, D(w) = \eta_1 D(u) + q\eta_2$. Wherein, D is the Paillier decryption algorithm, $\Gamma \in Z^*_{N^2}$ is the public key for Paillier encryption scheme, $\widetilde{N}$ is the distributed generated RSA composite number in this scheme (the wave line is added here to make a symbolic difference).

C. Zero-knowledge proof $\prod_{EQ}$

Based on the public parameters $(c,\ pk, G, \hbar, \hbar')$ and the private parameters (α, r) of the certifier, for Elgamal encryption ENC, prove that there is the following relationship, $\exists\alpha\ \exists r,\ \alpha \in Z_q \wedge c = ENC_{pk}(\alpha; r) \wedge \hbar,\ \hbar' \in G \wedge \hbar' = \hbar\hat{}\alpha$, namely, $ENC(log_{\hbar}\hbar', r) = c$. We have constructed $\prod_{EQ}$, and the concrete process is to repeat the following steps for l times:

- The certifier randomly chooses $s \in Z_q, r_s \in Z_q$ (the digit of s should be greater than that of α to ensure privacy of α).
- Send $c_s = ENC_{pk}(s; r_s), h_s = h^s$ to the verifier;
- The verifier randomly selects the bit b and sends to the certifier.
- The certifier sends $x = s + b\alpha, y = r_s + br$ to the verifier.
- The verifier verifies the following equations:

$$ENC_{pk}(x; y) = c^b c_s \qquad h^x = h'^b h_s$$

- If they are valid for all l times, the verification is passed (error probability is 2^{-l}), otherwise the verification will fail.

II. Compatibility of coin type

The goal of DCRM compatibility design is to be compatible with as many digital assets as possible controlled by encryption algorithms. At present, most of the encrypted currencies use ECDSA signature algorithms which are consistent with Bitcoin [N08] and Ethereum [V14], so supporting ECDSA signature algorithms is the primary goal.

The control right of encrypted digital assets is embodied in the control right of private key. Take Bitcoin as an example, the essence of private key is a random number. Add the version number at the front of private key, and add the compression flag and additional check code at the back (after two SHA-256 [SHA256Wiki] operations, take the first four bytes of the hash result twice), and then encode it with Base58 to get the private key in WIF (Wallet import Format) format. The public key is generated by the private key through the elliptic curve encryption algorithm, and the bitcoin address is generated by the public key through the hash functions RIPEMD [RIPEMDWiki] and SHA. The parameter of ECDSA curve used in Bitcoin is Secp256k1 [Secp256k1Wiki] and is defined in the High Efficiency Cryptography Standard.

2.1 Ethereum Support

The ECDSA elliptic curve digital signature algorithm and the Secp256k1 curve parameters are also used in the digital signature algorithm used in Ethereum. The public key addresses and transaction signatures generated by DCRM algorithm can meet the requirements of Ethereum. These specific processes include creating digital signatures, verifying signatures, transaction signatures, public key recovery and so on.

A. Create digital signature

In the ECDSA implementation of Ethereum, the signed "message" is a transaction, or more accurately, a Keccak256 hash of RLP-encoded data from the transaction. The signature private key is the private key. The result is signature:

$$Sig = Fsig\,(Fkeccak256(m), k)$$

Wherein:

- k is the signature private key.
- m is a RLP encoded transaction.
- Fkeccak256 is Keccak256 hash function.
- Fsig is a signature algorithm.
- Sig is the resulting signature.

The function Fsig generates a signature Sig consisting of three values, commonly called r, s and v:

$$Sig = (r, s, v)$$

The core of DCRM algorithm is that the transaction hash is signed by the decentralized multiple private keys. The signature results r, s and v can meet the signature criteria of Ethereum, and are verified by miners and then packaged for transaction.

B. Signature verification

The signature verification algorithm receives the message (that is, the hash of the transaction we use), the signer's public key and signature (r and s values), and returns true if the signature is valid for this message and the public key.

C. Transaction signature

To generate a valid transaction, the initiator shall use the elliptic curve digital signature algorithm to digitally sign the transaction content. When we say "signing a transaction," we actually mean "signing the Keccak256 hash of RLP serialized transaction data". The signature is applied to the hash of transaction data instead of the transaction itself.

To sign a transaction in the Ethereum, the initiator shall:

a) create a transaction data structure with nine fields: nonce, gasPrice, gasLimit, to, value, data, chainID, 0, 0

b) generate RLP encoded serialized messages for transaction data structures

c) calculate the Keccak256 hash value of this serialized message

d) compute the ECDSA signature and use the private key to sign the hash

e) append the v, r and s values of the ECDSA signature to the transaction

A. Public key recovery

The transaction structure of Ethereum does not contain any "coming from" fields. This is because the public key of the initiator can be calculated directly from the ECDSA signature. After obtaining the public key, the address can be calculated. The process of restoring the public key of the signer is called public key recovery.

Given the values r and s calculated in ECDSA, we can calculate two possible public keys. First, we calculate the two elliptic curve points R and R' from the r value of X coordinate in the signature. There are two points, because the elliptic curve is symmetrical on the X axis, for any value x, there are two possible points on the curve on both sides of the X axis. From r, we also calculate r^{-1}, which is a multiplication inverse of r. Finally, we calculate z, which means "Keccak256 hash that signs RLP serialized transaction data".

So the two possible public keys are:

$$K_1 = r^{-1}(sR - zG)$$

and

$$K_2 = r^{-1}(sR' - zG)$$

Wherein:

* K_1 and K_2 are two possibilities for the public key of signer.

* r^{-1} is a multiplication inverse of the signed r value.

* s is the value of the signature.

* R and R' are two possibilities for the temporary public key Q.

* z refers to "Keccak256 hash that signs RLP serialized transaction data".

* G is the generating point of elliptic curve.

To improve efficiency, the transaction signature includes the prefix v, which tells us which of the two possible R values is the temporary public key.

The signature result of the DCRM algorithm includes the v value needed to restore the public key.

2.2 Bitcoin support

The signature algorithm of Bitcoin is the same as that of Ethereum. In the implementation of the ECDSA algorithm of Bitcoin, the signed "message" is the transaction, specifically the SIGHASH of a specific subset of data in the transaction. The signature private key is the user's private key and the result is signature:

$$Sig = Fsig(Fhash(m), dA)$$

Wherein:

- ♦ dA is the signature private key.
- ♦ m is transaction (or part of it).
- ♦ Fhash is a hash function.
- ♦ Fsig is a signature algorithm.
- ♦ Sig is a result signature.

The function Fsig generates a signature Sig consisting of two values, commonly called r and s:

$$Sig = (r, s)$$

The signature result is a serialized byte stream, which adopts the international standard coding scheme of Distinguished Encoding Rules.

The difference between the signature process of Bitcoin and Ethereum is that the hash algorithm adopted by Bitcoin is sha256, and the Bitcoin does not need to use the v value used to recover the public key in the signature result.

2.3 Other encryption currency support

Standardized signature algorithms facilitate DCRM compatibility. The current version of DCRM supports all currencies that use the same ECDSA signature algorithm as Bitcoin, which contains almost 80% of the current digital currency. Ed25519, Schnorr and other new signature algorithms are also being adopted, and DCRM will research and support in the next stage.

III. Protocol implementation

3.1 DCRM node grouping protocol implementation

DCRM algorithm runs in the P2P network [P2PWiki] environment of FUSION through the network protocol. Specifically, m nodes are randomly selected in the super-node network composed of n nodes to form a fixed group of m nodes to complete the DCRM algorithm. Supernodes running the DCRM protocol will benefit, but only if a certain number of tokens are mortgaged, with a fixed IP address and the stable 7x24 operation.

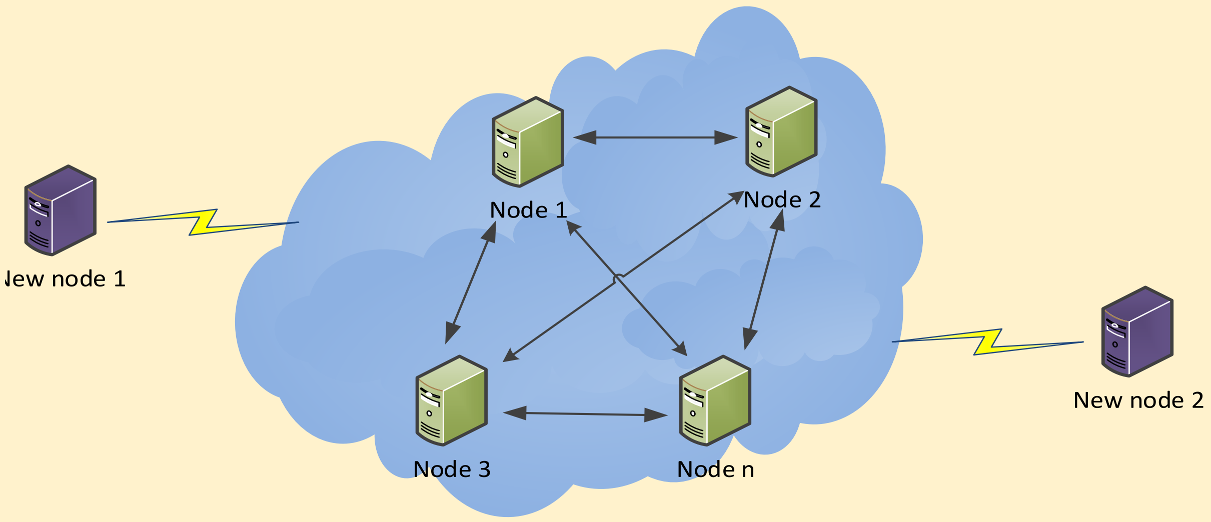


Fig. 1 Network topological graph

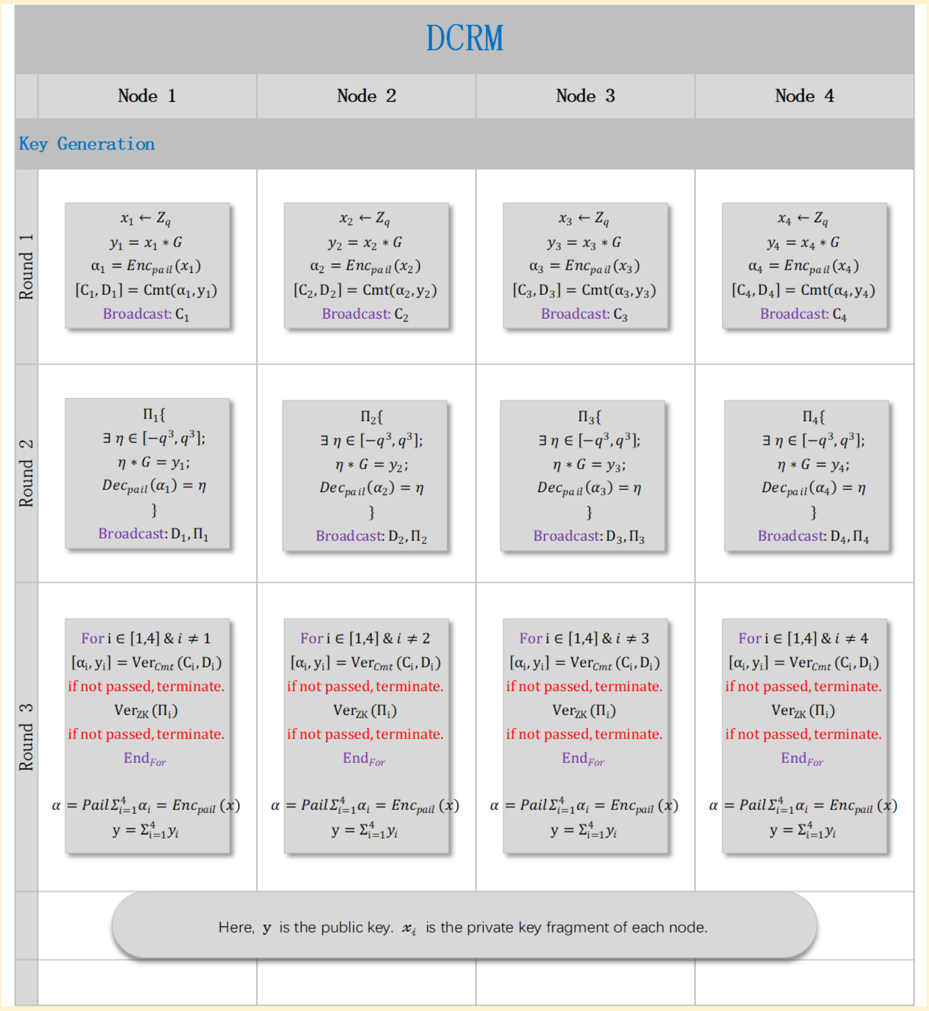
After the DCRM protocol group completes the construction, it can carry out the DCRM protocol initialization and distributed signature and other major functions.

3.2 DCRM initialization protocol implementation

In the initialization stage, the private key fragmentation and the generation of public key and address are completed.

First, DCRM initiates distributed private key generation according to user request, and each node independently generates private key fragments. Then, according to the specifications of the target blockchain, each node carries out distributed calculation based on private key fragments, and generates a valid public key and address on the target blockchain. This process is called initialization stage.

In the whole process, there is no transfer of private key fragments and splicing of private key. The specific implementation process is as follows:



DCRM

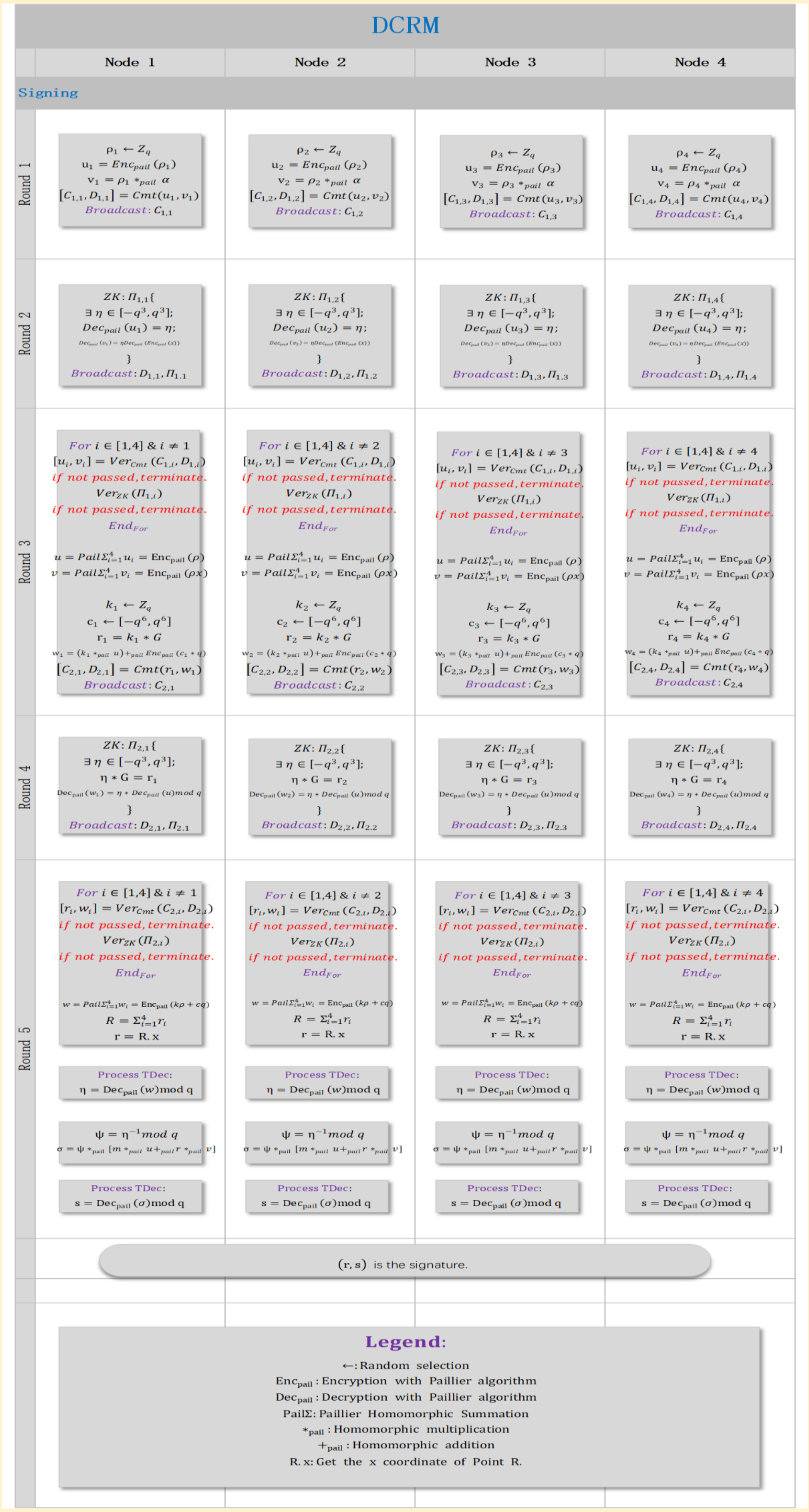
	Node 1	Node 2	Node 3	Node 4
Key Generation				
Round 1	$x_1 \leftarrow Z_q$ $y_1 = x_1 * G$ $\alpha_1 = Enc_{pail}(x_1)$ $[C_1, D_1] = Cmt(\alpha_1, y_1)$ Broadcast: C_1	$x_2 \leftarrow Z_q$ $y_2 = x_2 * G$ $\alpha_2 = Enc_{pail}(x_2)$ $[C_2, D_2] = Cmt(\alpha_2, y_2)$ Broadcast: C_2	$x_3 \leftarrow Z_q$ $y_3 = x_3 * G$ $\alpha_3 = Enc_{pail}(x_3)$ $[C_3, D_3] = Cmt(\alpha_3, y_3)$ Broadcast: C_3	$x_4 \leftarrow Z_q$ $y_4 = x_4 * G$ $\alpha_4 = Enc_{pail}(x_4)$ $[C_4, D_4] = Cmt(\alpha_4, y_4)$ Broadcast: C_4
Round 2	$\Pi_1\{$ $\exists\, \eta \in [-q^3, q^3];$ $\eta * G = y_1;$ $Dec_{pail}(\alpha_1) = \eta$ $\}$ Broadcast: D_1, Π_1	$\Pi_2\{$ $\exists\, \eta \in [-q^3, q^3];$ $\eta * G = y_2;$ $Dec_{pail}(\alpha_2) = \eta$ $\}$ Broadcast: D_2, Π_2	$\Pi_3\{$ $\exists\, \eta \in [-q^3, q^3];$ $\eta * G = y_3;$ $Dec_{pail}(\alpha_3) = \eta$ $\}$ Broadcast: D_3, Π_3	$\Pi_4\{$ $\exists\, \eta \in [-q^3, q^3];$ $\eta * G = y_4;$ $Dec_{pail}(\alpha_4) = \eta$ $\}$ Broadcast: D_4, Π_4
Round 3	For $i \in [1,4]\ \&\ i \neq 1$ $[\alpha_i, y_i] = Ver_{Cmt}(C_i, D_i)$ if not passed, terminate. $Ver_{ZK}(\Pi_i)$ if not passed, terminate. End_{For} $\alpha = Pail\Sigma_{i=1}^{4}\alpha_i = Enc_{pail}(x)$ $y = \Sigma_{i=1}^{4} y_i$	For $i \in [1,4]\ \&\ i \neq 2$ $[\alpha_i, y_i] = Ver_{Cmt}(C_i, D_i)$ if not passed, terminate. $Ver_{ZK}(\Pi_i)$ if not passed, terminate. End_{For} $\alpha = Pail\Sigma_{i=1}^{4}\alpha_i = Enc_{pail}(x)$ $y = \Sigma_{i=1}^{4} y_i$	For $i \in [1,4]\ \&\ i \neq 3$ $[\alpha_i, y_i] = Ver_{Cmt}(C_i, D_i)$ if not passed, terminate. $Ver_{ZK}(\Pi_i)$ if not passed, terminate. End_{For} $\alpha = Pail\Sigma_{i=1}^{4}\alpha_i = Enc_{pail}(x)$ $y = \Sigma_{i=1}^{4} y_i$	For $i \in [1,4]\ \&\ i \neq 4$ $[\alpha_i, y_i] = Ver_{Cmt}(C_i, D_i)$ if not passed, terminate. $Ver_{ZK}(\Pi_i)$ if not passed, terminate. End_{For} $\alpha = Pail\Sigma_{i=1}^{4}\alpha_i = Enc_{pail}(x)$ $y = \Sigma_{i=1}^{4} y_i$

Here, y is the public key. x_i is the private key fragment of each node.

3.3 DCRM distributed signature protocol implementation

At the signature stage, complete the effective signature of assets transactions on the relevant accounts.

ECDSA is a complex signature process where nonlinear computation is introduced. Therefore, the distributed ECDSA signature is implemented in DCRM through adopting complex and secure group signature technology, and cryptographic algorithms such as ElGamal, Paillier homomorphism algorithm, commitment algorithm, zero knowledge verification are used to ensure the security and effectiveness of the whole process. The specific implementation process is as follows:



DCRM

	Node 1	Node 2	Node 3	Node 4
Signing				
Round 1	$\rho_1 \leftarrow Z_q$ $u_1 = Enc_{pail}(\rho_1)$ $v_1 = \rho_1 *_{pail} \alpha$ $[C_{1,1}, D_{1,1}] = Cmt(u_1, v_1)$ $Broadcast: C_{1,1}$	$\rho_2 \leftarrow Z_q$ $u_2 = Enc_{pail}(\rho_2)$ $v_2 = \rho_2 *_{pail} \alpha$ $[C_{1,2}, D_{1,2}] = Cmt(u_2, v_2)$ $Broadcast: C_{1,2}$	$\rho_3 \leftarrow Z_q$ $u_3 = Enc_{pail}(\rho_3)$ $v_3 = \rho_3 *_{pail} \alpha$ $[C_{1,3}, D_{1,3}] = Cmt(u_3, v_3)$ $Broadcast: C_{1,3}$	$\rho_4 \leftarrow Z_q$ $u_4 = Enc_{pail}(\rho_4)$ $v_4 = \rho_4 *_{pail} \alpha$ $[C_{1,4}, D_{1,4}] = Cmt(u_4, v_4)$ $Broadcast: C_{1,4}$
Round 2	$ZK: \Pi_{1,1}\{$ $\exists \eta \in [-q^3, q^3];$ $Dec_{pail}(u_1) = \eta;$ $Dec_{pail}(v_1) = \eta Dec_{pail}(Enc_{pail}(x))$ $\}$ $Broadcast: D_{1,1}, \Pi_{1.1}$	$ZK: \Pi_{1,2}\{$ $\exists \eta \in [-q^3, q^3];$ $Dec_{pail}(u_2) = \eta;$ $Dec_{pail}(v_2) = \eta Dec_{pail}(Enc_{pail}(x))$ $\}$ $Broadcast: D_{1,2}, \Pi_{1.2}$	$ZK: \Pi_{1,3}\{$ $\exists \eta \in [-q^3, q^3];$ $Dec_{pail}(u_3) = \eta;$ $Dec_{pail}(v_3) = \eta Dec_{pail}(Enc_{pail}(x))$ $\}$ $Broadcast: D_{1,3}, \Pi_{1.3}$	$ZK: \Pi_{1,4}\{$ $\exists \eta \in [-q^3, q^3];$ $Dec_{pail}(u_4) = \eta;$ $Dec_{pail}(v_4) = \eta Dec_{pail}(Enc_{pail}(x))$ $\}$ $Broadcast: D_{1,4}, \Pi_{1.4}$
Round 3	$For\ i \in [1,4]\ \&\ i \neq 1$ $[u_i, v_i] = Ver_{Cmt}(C_{1,i}, D_{1,i})$ $if\ not\ passed, terminate.$ $Ver_{ZK}(\Pi_{1,i})$ $if\ not\ passed, terminate.$ End_{For} $u = Pail\Sigma_{i=1}^{4} u_i = Enc_{pail}(\rho)$ $v = Pail\Sigma_{i=1}^{4} v_i = Enc_{pail}(\rho x)$ $k_1 \leftarrow Z_q$ $c_1 \leftarrow [-q^6, q^6]$ $r_1 = k_1 * G$ $w_1 = (k_1 *_{pail} u) +_{pail} Enc_{pail}(c_1 * q)$ $[C_{2,1}, D_{2,1}] = Cmt(r_1, w_1)$ $Broadcast: C_{2,1}$	$For\ i \in [1,4]\ \&\ i \neq 2$ $[u_i, v_i] = Ver_{Cmt}(C_{1,i}, D_{1,i})$ $if\ not\ passed, terminate.$ $Ver_{ZK}(\Pi_{1,i})$ $if\ not\ passed, terminate.$ End_{For} $u = Pail\Sigma_{i=1}^{4} u_i = Enc_{pail}(\rho)$ $v = Pail\Sigma_{i=1}^{4} v_i = Enc_{pail}(\rho x)$ $k_2 \leftarrow Z_q$ $c_2 \leftarrow [-q^6, q^6]$ $r_2 = k_2 * G$ $w_2 = (k_2 *_{pail} u) +_{pail} Enc_{pail}(c_2 * q)$ $[C_{2,2}, D_{2,2}] = Cmt(r_2, w_2)$ $Broadcast: C_{2,2}$	$For\ i \in [1,4]\ \&\ i \neq 3$ $[u_i, v_i] = Ver_{Cmt}(C_{1,i}, D_{1,i})$ $if\ not\ passed, terminate.$ $Ver_{ZK}(\Pi_{1,i})$ $if\ not\ passed, terminate.$ End_{For} $u = Pail\Sigma_{i=1}^{4} u_i = Enc_{pail}(\rho)$ $v = Pail\Sigma_{i=1}^{4} v_i = Enc_{pail}(\rho x)$ $k_3 \leftarrow Z_q$ $c_3 \leftarrow [-q^6, q^6]$ $r_3 = k_3 * G$ $w_3 = (k_3 *_{pail} u) +_{pail} Enc_{pail}(c_3 * q)$ $[C_{2,3}, D_{2,3}] = Cmt(r_3, w_3)$ $Broadcast: C_{2,3}$	$For\ i \in [1,4]\ \&\ i \neq 4$ $[u_i, v_i] = Ver_{Cmt}(C_{1,i}, D_{1,i})$ $if\ not\ passed, terminate.$ $Ver_{ZK}(\Pi_{1,i})$ $if\ not\ passed, terminate.$ End_{For} $u = Pail\Sigma_{i=1}^{4} u_i = Enc_{pail}(\rho)$ $v = Pail\Sigma_{i=1}^{4} v_i = Enc_{pail}(\rho x)$ $k_4 \leftarrow Z_q$ $c_4 \leftarrow [-q^6, q^6]$ $r_4 = k_4 * G$ $w_4 = (k_4 *_{pail} u) +_{pail} Enc_{pail}(c_4 * q)$ $[C_{2,4}, D_{2,4}] = Cmt(r_4, w_4)$ $Broadcast: C_{2,4}$
Round 4	$ZK: \Pi_{2,1}\{$ $\exists \eta \in [-q^3, q^3];$ $\eta * G = r_1$ $Dec_{pail}(w_1) = \eta * Dec_{pail}(u) mod\ q$ $\}$ $Broadcast: D_{2,1}, \Pi_{2.1}$	$ZK: \Pi_{2,2}\{$ $\exists \eta \in [-q^3, q^3];$ $\eta * G = r_2$ $Dec_{pail}(w_2) = \eta * Dec_{pail}(u) mod\ q$ $\}$ $Broadcast: D_{2,2}, \Pi_{2.2}$	$ZK: \Pi_{2,3}\{$ $\exists \eta \in [-q^3, q^3];$ $\eta * G = r_3$ $Dec_{pail}(w_3) = \eta * Dec_{pail}(u) mod\ q$ $\}$ $Broadcast: D_{2,3}, \Pi_{2.3}$	$ZK: \Pi_{2,4}\{$ $\exists \eta \in [-q^3, q^3];$ $\eta * G = r_4$ $Dec_{pail}(w_4) = \eta * Dec_{pail}(u) mod\ q$ $\}$ $Broadcast: D_{2,4}, \Pi_{2.4}$
Round 5	$For\ i \in [1,4]\ \&\ i \neq 1$ $[r_i, w_i] = Ver_{Cmt}(C_{2,i}, D_{2,i})$ $if\ not\ passed, terminate.$ $Ver_{ZK}(\Pi_{2,i})$ $if\ not\ passed, terminate.$ End_{For} $w = Pail\Sigma_{i=1}^{4} w_i = Enc_{pail}(k\rho + cq)$ $R = \Sigma_{i=1}^{4} r_i$ $r = R.x$ Process TDec: $\eta = Dec_{pail}(w) mod\ q$ $\psi = \eta^{-1} mod\ q$ $\sigma = \psi *_{pail} [m *_{pail} u +_{pail} r *_{pail} v]$ Process TDec: $s = Dec_{pail}(\sigma) mod\ q$	$For\ i \in [1,4]\ \&\ i \neq 2$ $[r_i, w_i] = Ver_{Cmt}(C_{2,i}, D_{2,i})$ $if\ not\ passed, terminate.$ $Ver_{ZK}(\Pi_{2,i})$ $if\ not\ passed, terminate.$ End_{For} $w = Pail\Sigma_{i=1}^{4} w_i = Enc_{pail}(k\rho + cq)$ $R = \Sigma_{i=1}^{4} r_i$ $r = R.x$ Process TDec: $\eta = Dec_{pail}(w) mod\ q$ $\psi = \eta^{-1} mod\ q$ $\sigma = \psi *_{pail} [m *_{pail} u +_{pail} r *_{pail} v]$ Process TDec: $s = Dec_{pail}(\sigma) mod\ q$	$For\ i \in [1,4]\ \&\ i \neq 3$ $[r_i, w_i] = Ver_{Cmt}(C_{2,i}, D_{2,i})$ $if\ not\ passed, terminate.$ $Ver_{ZK}(\Pi_{2,i})$ $if\ not\ passed, terminate.$ End_{For} $w = Pail\Sigma_{i=1}^{4} w_i = Enc_{pail}(k\rho + cq)$ $R = \Sigma_{i=1}^{4} r_i$ $r = R.x$ Process TDec: $\eta = Dec_{pail}(w) mod\ q$ $\psi = \eta^{-1} mod\ q$ $\sigma = \psi *_{pail} [m *_{pail} u +_{pail} r *_{pail} v]$ Process TDec: $s = Dec_{pail}(\sigma) mod\ q$	$For\ i \in [1,4]\ \&\ i \neq 4$ $[r_i, w_i] = Ver_{Cmt}(C_{2,i}, D_{2,i})$ $if\ not\ passed, terminate.$ $Ver_{ZK}(\Pi_{2,i})$ $if\ not\ passed, terminate.$ End_{For} $w = Pail\Sigma_{i=1}^{4} w_i = Enc_{pail}(k\rho + cq)$ $R = \Sigma_{i=1}^{4} r_i$ $r = R.x$ Process TDec: $\eta = Dec_{pail}(w) mod\ q$ $\psi = \eta^{-1} mod\ q$ $\sigma = \psi *_{pail} [m *_{pail} u +_{pail} r *_{pail} v]$ Process TDec: $s = Dec_{pail}(\sigma) mod\ q$

(r, s) is the signature.

Legend:

$\leftarrow$: Random selection
Enc_{pail}: Encryption with Paillier algorithm
Dec_{pail}: Decryption with Paillier algorithm
$Pail\Sigma$: Paillier Homomorphic Summation
$*_{pail}$: Homomorphic multiplication
$+_{pail}$: Homomorphic addition
$R.x$: Get the x coordinate of Point R.

IV. RPC interface

The core functions of DCRM include generating public keys and addresses and completing distributed signatures, which are provided by the super nodes in the chain through RPC services. Wallet and other applications can use the RPC interface to complete the Lock-in and Lock-out functions of assets.

JSON-RPC is a stateless and lightweight remote procedure call (RPC) protocol. The specification mainly defines several data structures and the treatment rules around them. It has nothing to do with transport, because these contents can be used in the same process, through sockets, through HTTP, or in many different message-conveying environments. It uses JSON (RFC 4627) as data format. JSON is a lightweight data interchange format that represents numbers, strings, ordered value sequences, and a collection of name / value pairs.

The completion of DCRM's transaction signature means asset transfer Lock-out, and the caller's ownership of the asset shall be confirmed when using RPC interface.

4.1 Interface of application for public key address

Interface of application for public key address：

fsn_dcrmReqAddress(user.publicKey,CoinType)

A. Functional description

According to the terminal request, complete the user registration and the generation of distributed private key fragments, public key and address, bind the generated address to the user's identity, and return the generated address to the user.

The address is the Lock-in address of the cross linked asset based on DCRM technology. Transfer the corresponding assets to the address, that is, perform the Lock-in operation.

B. Input parameters

***user.publicKey*:** the public key generated by the user locally in the public private key pair that represents the user identity. For the DCRM function provided by FSUION in the future, user. publicKey is the public key corresponding to the user-generated FUSION account.

The public key of the user identity completes the verification of the Lock-in account and the ownership of the asset when the user initiates Lock-out in the future.

CoinType: the type of the crypto-asset which the user select.

C. Return value

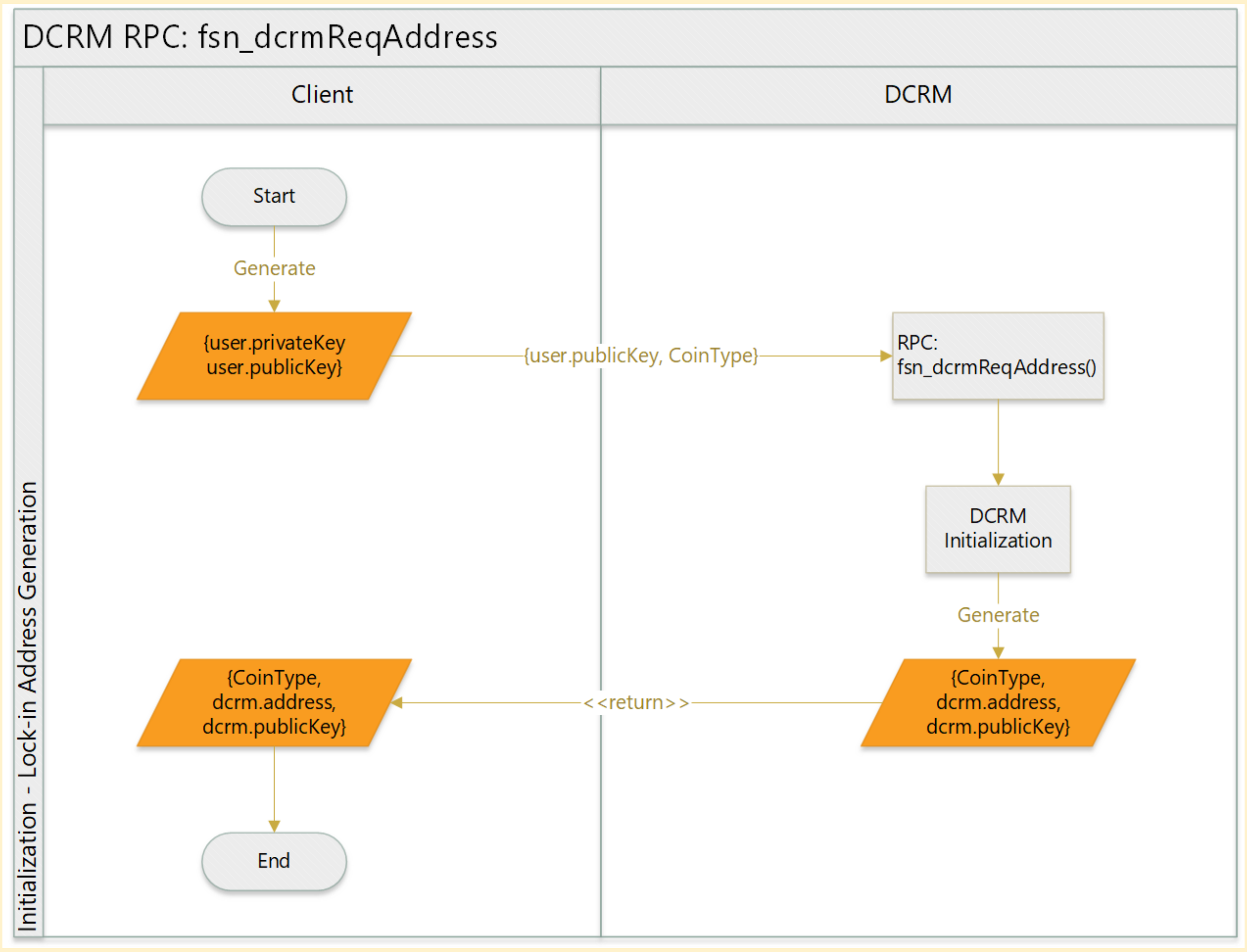
CoinType: the type of the crypto-asset.

dcrm.address: the address on the target blockchain generated by DCRM in a distributed manner.

dcrm.publicKey: the public key on the target blockchain generated by DCRM in a distributed manner.

We plan to return the public key instead of the address in the future. With this public key, the user can generate the corresponding addresses on each target blockchain supported by DCRM, such as the addresses of Bitcoin and Ethereum. This address is the user's Lock-in address.

RPC call flow chart of ***fsn_dcrmReqAddress(user.PublicKey,CoinType)*** interface


DCRM RPC: fsn_dcrmReqAddress
Client
DCRM
Initialization - Lock-in Address Generation
Start
Generate
{user.privateKey user.publicKey}
{user.publicKey, CoinType}
RPC: fsn_dcrmReqAddress()
DCRM Initialization
Generate
{CoinType, dcrm.address, dcrm.publicKey}
<<return>>
{CoinType, dcrm.address, dcrm.publicKey}
End

4.2 Interface of application for distributed signature

Interface of application for distributed signature：

fsn_dcrmSign(sig, transactionHash, dcrm.address,CoinType)

A. Functional description

According to the terminal request, the external transfer transaction initiated by cross-chain assets managed and controlled by DCRM is signed.

During the calling process of this interface, the terminal is required to submit the signature of the transaction outgoing address with the user's identity private key to verify that the request has ownership of the address.

B. Input parameters

sig: use the value of user identity private key to transactionHash signature. It is a string of signature (r, s) generated by this signature.

Verification of sig values is used to verify the ownership of addresses in transactions.

transactionHash: the Hash value of the transaction to be signed.

dcrm.address: the transaction outgoing address.

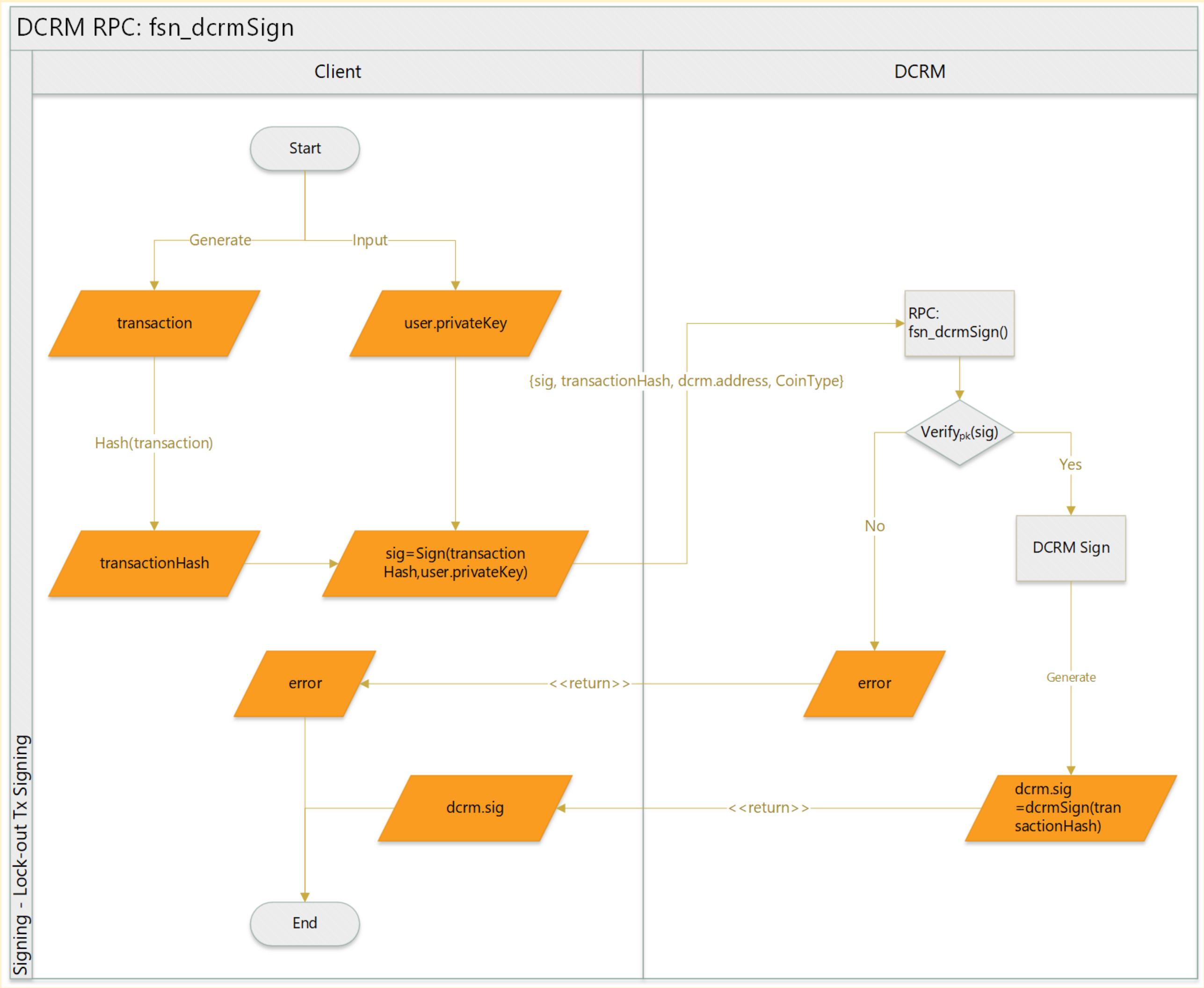
CoinType: the type of the crypto-asset which the user select.

C. Return value

dcrm.sig: Return a transaction signature generated by DCRM in a distributed manner for the transaction when the input parameter Sig passes the verification. It is a string of signature (r, s, v) generated by DCRM signature.

error: Return error messages that do not have permission to operate on the address in the event of failure to verify the input parameter Sig.

D.RPC calling flow chart of ***fsn_dcrmSign(sig, transactionHash, dcrm.address,CoinType)*** interface



References

- [BFM88] Manuel Blum, Paul Feldman, Silvio Micali: Non-Interactive Zero-Knowledge and Its Applications (Extended Abstract). STOC 1988: 103-112
- [CommitmentWiki] Commitment Scheme of Wikipedia,available at https://en.wikipedia.org/wiki/Commitment_scheme
- [CS98] Ronald Cramer, Victor Shoup: A Practical Public Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attack. CRYPTO 1998: 13-25
- [DR02] Joan Daemen, Vincent Rijmen: The Design of Rijndael: AES - The Advanced Encryption Standard. Information Security and Cryptography, Springer 2002, ISBN 3-540-42580-2
- [G84] Taher El Gamal: A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. CRYPTO 1984: 10-18
- [GGN16] Rosario Gennaro, Steven Goldfeder, Arvind Narayanan: Threshold-Optimal DSA/ECDSA Signatures and an Application to Bitcoin Wallet Security. ACNS 2016: 156-174
- [HMRT12] Carmit Hazay, Gert Læssøe Mikkelsen, Tal Rabin, Tomas Toft: Efficient RSA Key Genera-tion and Threshold Paillier in the Two-Party Setting. CT-RSA 2012: 313-331
- [JL17] Simon Josefsson, Ilari Liusvaara: Edwards-Curve Digital Signature Algorithm (EdDSA). RFC 8032: 1-60 (2017)
- [JMV01] Don Johnson, Alfred Menezes, Scott A. Vanstone: The Elliptic Curve Digital Signature Algo-rithm (ECDSA). Int. J. Inf. Sec. 1(1): 36-63 (2001)
- [N08] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 31 October 2008, available at https://bitcoin.org/bitcoin.pdf
- [P99] Pascal Paillier: Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. EUROCRYPT 1999: 223-238
- [P2PWiki] Peer-to-peer (P2P) networking is a distributed application architecture that partitions tasks or workloads between peers, available at https://en.wikipedia.org/wiki/Peer-to-peer
- [RIPEMDWiki] RIPEMD is a family of cryptographic hash functions developed in Leuven, Belgium, by Hans Dobbertin, Antoon Bosselaers and Bart Preneel at the COSIC research group at the Katholieke Universiteit Leuven, and first published in 1996, available at https://en.wikipedia.org/wiki/RIPEMD

- [S89] Claus-Peter Schnorr: Efficient Identification and Signatures for Smart Cards (Abstract). EUROCRYPT 1989: 688-689
- [Secp256k1Wiki] Parameters of the elliptic curve used in Bitcoin's public-key cryptography, available at https://en.bitcoin.it/wiki/Secp256k1, http://www.secg.org/sec2-v2.pdf
- [SHA256Wiki] SHA-256 is a set of cryptographic hash functions designed by the United States National Security Agency (NSA), available at https://en.wikipedia.org/wiki/SHA-2
- [V14] Vitalik Buterin. Ethereum: Now Going Public. Archived from the original on 2 March 2014