dapp uri中的publicKey来源以及URI拼接逻辑

react-app/node\_modules/@walletconnect/client/dist/cjs/controllers/engine.js

propose(params) {

const self = {

publicKey: yield this.sequence.client.crypto.generateKeyPair(),

metadata: *params* === null || *params* === void 0 ? void 0 : *params*.metadata,

};

react-app/node\_modules/@walletconnect/client/dist/cjs/controllers/crypto.js

const utils\_1 = require("@walletconnect/utils");

generateKeyPair() {

return tslib\_1.\_\_awaiter(this, void 0, void 0, function\* () {

const keyPair = utils\_1.generateKeyPair();

return this.setKeyPair(keyPair);

});

}

setKeyPair(*keyPair*) {

return tslib\_1.\_\_awaiter(this, void 0, void 0, function\* () {

const keys = this.concatKeys(*keyPair*.publicKey, *keyPair*.privateKey); // publicKey = publicKey + privateKey

yield this.keychain.set(*keyPair*.publicKey, keys);

return *keyPair*.publicKey;

});

}

react-app/node\_modules/@walletconnect/utils/dist/cjs/crypto.js

const ecies25519 = tslib\_1.\_\_importStar(require("@walletconnect/ecies-25519"));

const encoding = tslib\_1.\_\_importStar(require("@walletconnect/encoding"));

function generateKeyPair() {

const keyPair = ecies25519.generateKeyPair();

return {

privateKey: encoding.arrayToHex(keyPair.privateKey),

publicKey: encoding.arrayToHex(keyPair.publicKey),

};

}

const x25519 = \_\_importStar(require("@stablelib/x25519"));

function generateKeyPair(*entropy*) {

const prng = typeof *entropy* !== "undefined"

? generatePnrgFromEntropy(*entropy*)

: undefined;

const keyPair = x25519.generateKeyPair(prng);

return {

publicKey: keyPair.publicKey,

privateKey: keyPair.secretKey,

};

}

var random\_1 = require("@stablelib/random");

function generateKeyPair(*prng*) {

var seed = random\_1.randomBytes(32, *prng*);

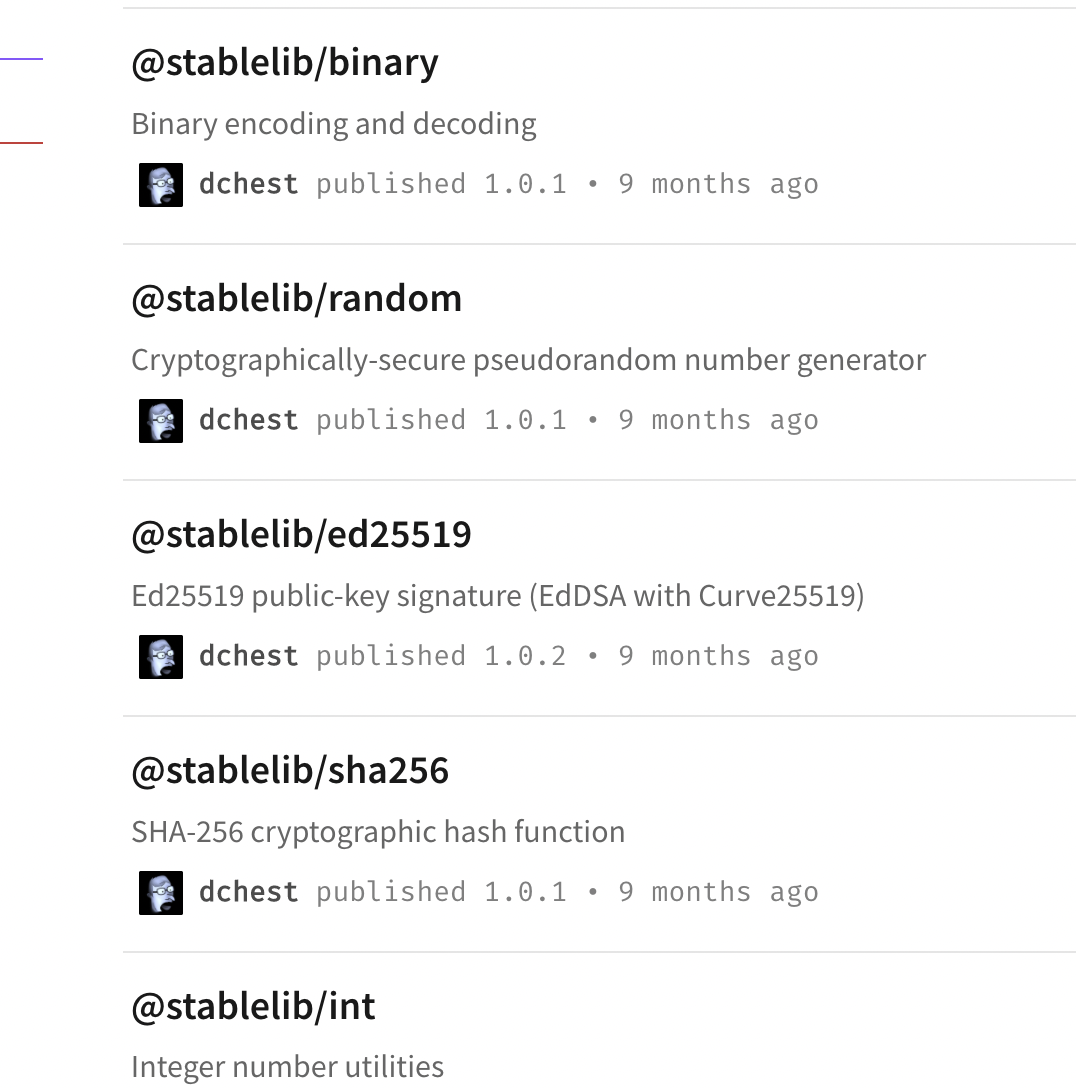
var result = generateKeyPairFromSeed(seed);

wipe\_1.wipe(seed);

return result;

}

exports.generateKeyPair = generateKeyPair;



getDefaultSignal({ *topic*, *relay*, *proposer* }) {

return tslib\_1.\_\_awaiter(this, void 0, void 0, function\* () {

const uri = utils\_1.formatUri({

protocol: this.client.protocol,

version: this.client.version,

topic: *topic*,

publicKey: *proposer*.publicKey,

controller: *proposer*.controller,

relay: *relay*,

});

const signal = {

method: constants\_1.PAIRING\_SIGNAL\_METHOD\_URI,

params: { uri },

};

return signal;

});

}

const qs = tslib\_1.\_\_importStar(require("query-string"));

const safe\_json\_1 = require("@walletconnect/safe-json");

function formatUri(*params*) {

return (`${*params*.protocol}:${*params*.topic}@${*params*.version}?` +

qs.stringify({

publicKey: *params*.publicKey,

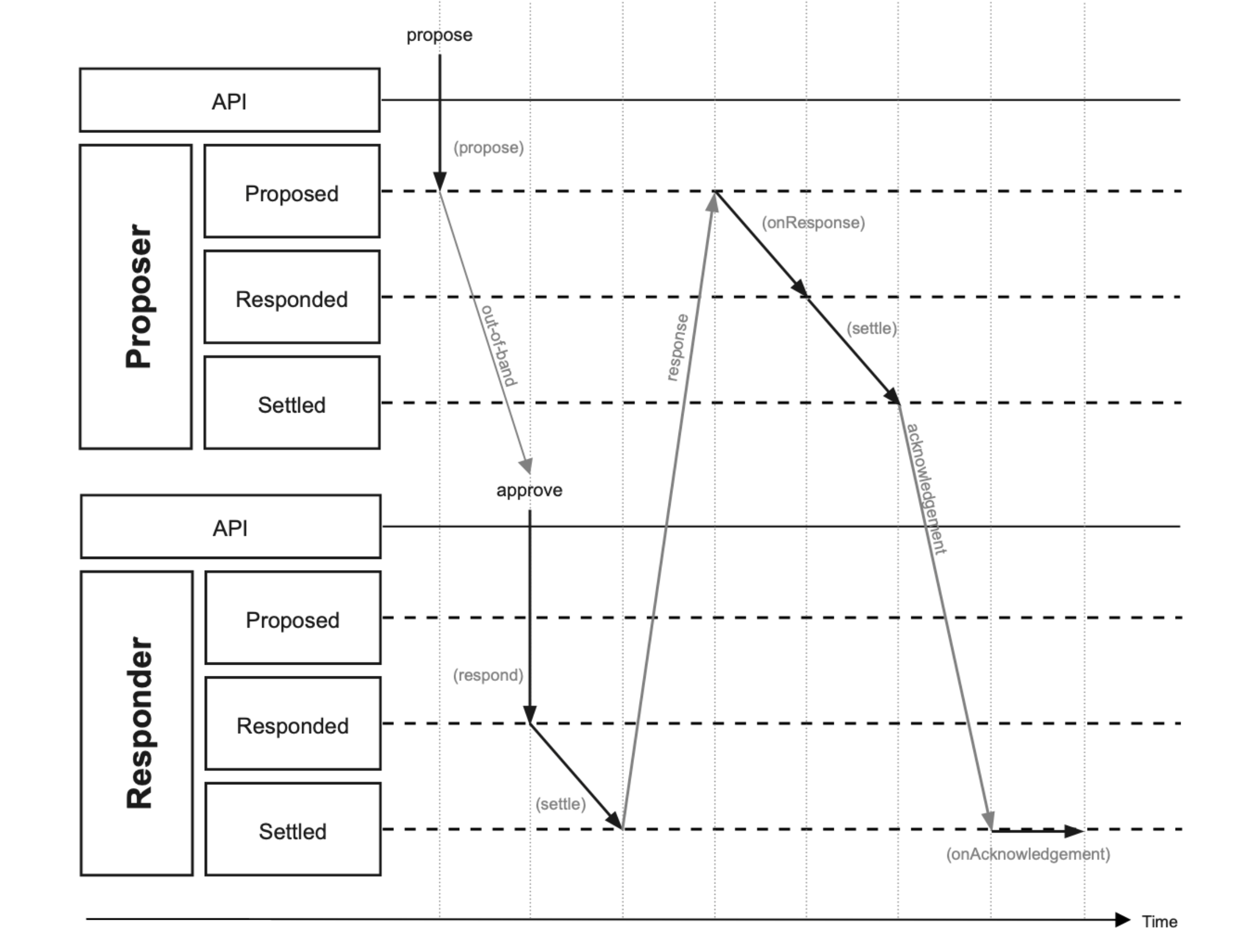
controller: *params*.controller,

relay: safe\_json\_1.safeJsonStringify(*params*.relay),

}));

}

exports.formatUri = formatUri;



let params = {

metadata: getAppMetadata() || DEFAULT\_APP\_METADATA,

pairing,

permissions: {

blockchain: {

chains,

},

jsonrpc: {

methods,

},

},

}

{

"metadata": {

"description": "React App for WalletConnect",

"url": "http://localhost:3001",

"icons": [

"http://localhost:3001/favicon.ico"

],

"name": "React App"

},

"permissions": {

"blockchain": {

"chains": [

"eip155:65534"

]

},

"jsonrpc": {

"methods": [

"eth\_sendTransaction",

"personal\_sign",

"eth\_signTypedData",

"eth\_signTransaction"

]

}

}

}

const session = await this.state.client.connect(params);

**1.propose**

create(*params*) {

let pending = yield this.propose(*params*);

……

}

propose(*params*) {

return tslib\_1.\_\_awaiter(this, void 0, void 0, function\* () {

this.sequence.logger.debug(`Propose ${this.sequence.context}`);

this.sequence.logger.trace({ type: "method", method: "propose", params });

yield this.sequence.validatePropose(*params*);

const relay = (*params* === null || *params* === void 0 ? void 0 : *params*.relay) || { protocol: constants\_1.RELAYER\_DEFAULT\_PROTOCOL };

const topic = utils\_1.generateRandomBytes32();

const self = {

publicKey: yield this.sequence.client.crypto.generateKeyPair(),

metadata: *params* === null || *params* === void 0 ? void 0 : *params*.metadata,

};

if (!self.metadata)

delete self.metadata;

const proposer = {

publicKey: self.publicKey,

controller: this.sequence.client.controller,

metadata: self.metadata,

};

if (!proposer.metadata)

delete proposer.metadata;

const signal = (*params* === null || *params* === void 0 ? void 0 : *params*.signal) || (yield this.sequence.getDefaultSignal({ topic, relay, proposer }));

const permissions = (*params* === null || *params* === void 0 ? void 0 : *params*.permissions) || (yield this.sequence.getDefaultPermissions());

const ttl = (*params* === null || *params* === void 0 ? void 0 : *params*.ttl) || (yield this.sequence.getDefaultTTL());

const proposal = {

relay,

topic,

proposer,

signal,

permissions,

ttl,

};

const pending = {

status: this.sequence.config.status.proposed,

topic: proposal.topic,

relay: proposal.relay,

self,

proposal,

};

yield this.sequence.pending.set(pending.topic, pending);

return pending;

});

}

**2.out on band**

this.state.client.on(

CLIENT\_EVENTS.pairing.proposal,

async (*proposal*: PairingTypes.Proposal) => {

*// uri 应通过 QR 码扫描或移动深度链接与钱包共享*

const { uri } = *proposal*.signal.params;

this.setState({ uri });

console.log("EVENT", "QR Code Modal open");

QRCodeModal.open(uri, () => {

console.log("EVENT", "QR Code Modal closed");

});

},

);

**uri:**

wc:f80cedcc8b7def1752467ec7de5a1964c6a1d15727fc8484c2ceb1a3be4f0483@2?controller=false&publicKey=4f9af375bf39375a99bdb5d46ec2158ca0f890eeda15cadc7c4c6d7e7969ef7b&relay=%7B%22protocol%22%3A%22waku%22%7D

往wallet输入从app生成的URI，进行配对

await this.state.client.pair({ uri });

pair(*params*) {

return tslib\_1.\_\_awaiter(this, void 0, void 0, function\* () {

this.logger.debug(`Pairing`);

this.logger.trace({ type: "method", method: "pair", params });

const proposal = formatPairingProposal(*params*.uri);

const approved = proposal.proposer.controller !== this.controller;

const reason = approved

? undefined

: utils\_1.ERROR.UNAUTHORIZED\_MATCHING\_CONTROLLER.format({ controller: this.controller });

// approved = true

const pending = yield this.pairing.respond({ approved, proposal, reason });

if (!utils\_1.isPairingResponded(pending)) {

const error = utils\_1.ERROR.NO\_MATCHING\_RESPONSE.format({ context: "pairing" });

this.logger.error(error.message);

throw new Error(error.message);

}

if (utils\_1.isPairingFailed(pending.outcome)) {

this.logger.debug(`Pairing Failure`);

this.logger.trace({ type: "method", method: "pair", outcome: pending.outcome });

throw new Error(pending.outcome.reason.message);

}

this.logger.debug(`Pairing Success`);

this.logger.trace({ type: "method", method: "pair", pending });

const pairing = yield this.pairing.get(pending.outcome.topic);

return pairing;

});

}

function formatPairingProposal(*uri*) {

const uriParams = utils\_1.parseUri(*uri*);

return {

topic: uriParams.topic,

relay: uriParams.relay,

proposer: { publicKey: uriParams.publicKey, controller: uriParams.controller },

signal: { method: constants\_1.PAIRING\_SIGNAL\_METHOD\_URI, params: { uri } },

permissions: {

jsonrpc: { methods: [constants\_1.SESSION\_JSONRPC.propose] },

notifications: { types: [] },

},

ttl: constants\_1.PAIRING\_DEFAULT\_TTL,

};

}

respond(*params*) {

return tslib\_1.\_\_awaiter(this, void 0, void 0, function\* () {

this.sequence.logger.debug(`Respond ${this.sequence.context}`);

this.sequence.logger.trace({ type: "method", method: "respond", params });

yield this.sequence.validateRespond(*params*);

const { approved, proposal, response } = *params*;

const { relay, ttl } = proposal;

const self = {

publicKey: yield this.sequence.client.crypto.generateKeyPair(),

metadata: response === null || response === void 0 ? void 0 : response.metadata,

};

if (!self.metadata)

delete self.metadata;

if (approved) {

try {

const responder = {

publicKey: self.publicKey,

metadata: response === null || response === void 0 ? void 0 : response.metadata,

};

if (!responder.metadata)

delete responder.metadata;

const expiry = utils\_1.calcExpiry(proposal.ttl);

const state = (response === null || response === void 0 ? void 0 : response.state) || {};

const peer = {

publicKey: proposal.proposer.publicKey,

metadata: proposal.proposer.metadata,

};

if (!peer.metadata)

delete peer.metadata;

const controller = proposal.proposer.controller

? { publicKey: peer.publicKey }

: { publicKey: self.publicKey };

const permissions = Object.assign(Object.assign({}, proposal.permissions), { controller });

const settled = yield this.settle({

relay,

self,

peer,

permissions,

state,

ttl,

expiry,

});

const outcome = {

topic: settled.topic,

relay,

state,

responder,

expiry,

};

const pending = {

status: this.sequence.config.status.responded,

topic: proposal.topic,

relay,

self,

proposal,

outcome,

};

yield this.sequence.pending.set(pending.topic, pending);

return pending;

}

catch (e) {

const reason = utils\_1.ERROR.GENERIC.format({ message: e.message });

const outcome = { reason };

const pending = {

status: this.sequence.config.status.responded,

topic: proposal.topic,

relay,

self,

proposal,

outcome,

};

yield this.sequence.pending.set(pending.topic, pending);

return pending;

}

}

else {

const defaultReason = utils\_1.ERROR.NOT\_APPROVED.format({ context: this.sequence.name });

const outcome = { reason: (*params* === null || *params* === void 0 ? void 0 : *params*.reason) || defaultReason };

const pending = {

status: this.sequence.config.status.responded,

topic: proposal.topic,

relay,

self,

proposal,

outcome,

};

yield this.sequence.pending.set(pending.topic, pending);

return pending;

}

});

}

proposal

{

"relay": {

"protocol": "waku"

},

"topic": "9c152ce8a17d4043c12697c7d179e98d67fc8f505147cf615e226708fe1cbf61",

"proposer": {

"publicKey": "30d98aed277ff2411afcceb5fb227d4062c723a367e94efb1419b17944024e01",

"controller": false,

"metadata": {

"description": "React App for WalletConnect",

"url": "http://localhost:3001",

"icons": [

"http://localhost:3001/favicon.ico"

],

"name": "React App"

}

},

"signal": {

"method": "pairing",

"params": {

"topic": "74a7c849cd60cfb2cfa0be6a58b34e7d6f78be6663164da0308d76226c7ab96e"

}

},

"permissions": {

"blockchain": {

"chains": [

"eip155:65534"

]

},

"jsonrpc": {

"methods": [

"eth\_sendTransaction",

"personal\_sign",

"eth\_signTypedData",

"eth\_signTransaction"

]

},

"notifications": {

"types": []

}

},

"ttl": 604800

}

response

{

"state": {

"accounts": [

"eip155:65534:0x8cb2c7Cc1a6E52Bc71651e1bC1e7354A02F4Eb07"

]

},

"metadata": {

"description": "React Wallet for WalletConnect",

"url": "http://localhost:3000",

"icons": [

"http://localhost:3000/favicon.ico"

],

"name": "React Wallet"

}

}

Wallet处理用户对提议会话的批准

3.approve

const session = await this.state.client.approve({ proposal, response });

approve(*params*) {

return tslib\_1.\_\_awaiter(this, void 0, void 0, function\* () {

this.logger.debug(`Approving Session Proposal`);

this.logger.trace({ type: "method", method: "approve", params });

if (typeof *params*.response === "undefined") {

const error = utils\_1.ERROR.MISSING\_RESPONSE.format({ context: "session" });

this.logger.error(error.message);

throw new Error(error.message);

}

const state = *params*.response.state || constants\_1.SESSION\_EMPTY\_STATE;

const metadata = *params*.response.metadata || this.metadata;

if (typeof metadata === "undefined") {

const error = utils\_1.ERROR.MISSING\_OR\_INVALID.format({ name: "app metadata" });

this.logger.error(error.message);

throw new Error(error.message);

}

const approved = *params*.proposal.proposer.controller !== this.controller;

const reason = approved

? undefined

: utils\_1.ERROR.UNAUTHORIZED\_MATCHING\_CONTROLLER.format({ controller: this.controller });

4.(respond)

const pending = yield this.session.respond({

approved,

proposal: *params*.proposal,

response: { state, metadata },

reason,

});

if (!utils\_1.isSessionResponded(pending)) {

const error = utils\_1.ERROR.NO\_MATCHING\_RESPONSE.format({ context: "session" });

this.logger.error(error.message);

throw new Error(error.message);

}

if (utils\_1.isSessionFailed(pending.outcome)) {

this.logger.debug(`Session Proposal Approval Failure`);

this.logger.trace({ type: "method", method: "approve", outcome: pending.outcome });

throw new Error(pending.outcome.reason.message);

}

this.logger.debug(`Session Proposal Approval Success`);

this.logger.trace({ type: "method", method: "approve", pending });

return this.session.get(pending.outcome.topic);

});

}

respond(*params*) {

return tslib\_1.\_\_awaiter(this, void 0, void 0, function\* () {

this.sequence.logger.debug(`Respond ${this.sequence.context}`);

this.sequence.logger.trace({ type: "method", method: "respond", params });

yield this.sequence.validateRespond(*params*);

const { approved, proposal, response } = *params*;

const { relay, ttl } = proposal;

const self = {

publicKey: yield this.sequence.client.crypto.generateKeyPair(),

metadata: response === null || response === void 0 ? void 0 : response.metadata,

};

if (!self.metadata)

delete self.metadata;

if (approved) {

try {

const responder = {

publicKey: self.publicKey,

metadata: response === null || response === void 0 ? void 0 : response.metadata,

};

if (!responder.metadata)

delete responder.metadata;

const expiry = utils\_1.calcExpiry(proposal.ttl);

const state = (response === null || response === void 0 ? void 0 : response.state) || {};

const peer = {

publicKey: proposal.proposer.publicKey,

metadata: proposal.proposer.metadata,

};

if (!peer.metadata)

delete peer.metadata;

const controller = proposal.proposer.controller

? { publicKey: peer.publicKey }

: { publicKey: self.publicKey };

const permissions = Object.assign(Object.assign({}, proposal.permissions), { controller });

5.(settle)

const settled = yield this.settle({

relay,

self,

peer,

permissions,

state,

ttl,

expiry,

});

const outcome = {

topic: settled.topic,

relay,

state,

responder,

expiry,

};

const pending = {

status: this.sequence.config.status.responded,

topic: proposal.topic,

relay,

self,

proposal,

outcome,

};

yield this.sequence.pending.set(pending.topic, pending);

return pending;

}

catch (e) {

const reason = utils\_1.ERROR.GENERIC.format({ message: e.message });

const outcome = { reason };

const pending = {

status: this.sequence.config.status.responded,

topic: proposal.topic,

relay,

self,

proposal,

outcome,

};

yield this.sequence.pending.set(pending.topic, pending);

return pending;

}

}

else {

const defaultReason = utils\_1.ERROR.NOT\_APPROVED.format({ context: this.sequence.name });

const outcome = { reason: (*params* === null || *params* === void 0 ? void 0 : *params*.reason) || defaultReason };

const pending = {

status: this.sequence.config.status.responded,

topic: proposal.topic,

relay,

self,

proposal,

outcome,

};

yield this.sequence.pending.set(pending.topic, pending);

return pending;

}

});

}

const session = await this.state.client.approve({ proposal, response });

6. wallet approve 后session 响应结果

Session = {

"topic": "368b38a3f68ff1bee64bbfe07f77f5d025dc6f383d270b51248da17b214c6141",

"relay": {

"protocol": "waku"

},

"self": {

"publicKey": "ce7c40567b20f409a510cd1b1ff11677e83b7bbe527814509c460cfa061dd212",

"metadata": {

"description": "React Wallet for WalletConnect",

"url": "http://localhost:3001",

"icons": [

"http://localhost:3001/favicon.ico"

],

"name": "React Wallet"

}

},

"peer": {

"publicKey": "e6a5bea3395324b9b8e062252ae0538340c16ceb25567f76ab382e9ba8080167",

"metadata": {

"description": "React App for WalletConnect",

"url": "http://localhost:3000",

"icons": [

"http://localhost:3000/favicon.ico"

],

"name": "React App"

}

},

"permissions": {

"blockchain": {

"chains": [

"eip155:65534"

]

},

"jsonrpc": {

"methods": [

"eth\_sendTransaction",

"personal\_sign",

"eth\_signTypedData",

"eth\_signTransaction"

]

},

"notifications": {

"types": []

},

"controller": {

"publicKey": "ce7c40567b20f409a510cd1b1ff11677e83b7bbe527814509c460cfa061dd212"

}

},

"expiry": 1647240125,

"state": {

"accounts": [

"eip155:65534:0x8cb2c7Cc1a6E52Bc71651e1bC1e7354A02F4Eb07"

]

},

"acknowledged": false

}

const session = await this.state.client.connect(params);

7.( 6. wallet approve 后session 响应结果) 6步骤响应结果后，app的session响应结果

Session = {

"topic": "368b38a3f68ff1bee64bbfe07f77f5d025dc6f383d270b51248da17b214c6141",

"relay": {

"protocol": "waku"

},

"self": {

"publicKey": "e6a5bea3395324b9b8e062252ae0538340c16ceb25567f76ab382e9ba8080167",

"metadata": {

"description": "React App for WalletConnect",

"url": "http://localhost:3000",

"icons": [

"http://localhost:3000/favicon.ico"

],

"name": "React App"

}

},

"peer": {

"publicKey": "ce7c40567b20f409a510cd1b1ff11677e83b7bbe527814509c460cfa061dd212",

"metadata": {

"description": "React Wallet for WalletConnect",

"url": "http://localhost:3001",

"icons": [

"http://localhost:3001/favicon.ico"

],

"name": "React Wallet"

}

},

"permissions": {

"blockchain": {

"chains": [

"eip155:65534"

]

},

"jsonrpc": {

"methods": [

"eth\_sendTransaction",

"personal\_sign",

"eth\_signTypedData",

"eth\_signTransaction"

]

},

"notifications": {

"types": []

},

"controller": {

"publicKey": "ce7c40567b20f409a510cd1b1ff11677e83b7bbe527814509c460cfa061dd212"

}

},

"expiry": 1647240125,

"state": {

"accounts": [

"eip155:65534:0x8cb2c7Cc1a6E52Bc71651e1bC1e7354A02F4Eb07"

]

},

"acknowledged": false

}

const session = await this.state.client.connect(params);

8.settle

this.onSessionConnected(session);

public onSessionConnected = async (*session*: SessionTypes.Settled) => {

console.log("onSessionConnected session = ", *session*)

this.setState({ session });

this.onSessionUpdate(*session*.state.accounts, *session*.permissions.blockchain.chains);

};

public onSessionUpdate = async (*accounts*: string[], *chains*: string[]) => {

console.log("onSessionUpdate")

this.setState({ chains, accounts });

await this.getAccountBalances();

};

public getAccountBalances = async () => {

console.log("getAccountBalances")

this.setState({ fetching: true });

try {

const arr = await Promise.all(

this.state.accounts.map(async *account* => {

const [namespace, reference, address] = *account*.split(":");

const chainId = `${namespace}:${reference}`;

const assets = await apiGetAccountAssets(address, chainId);

return { account, assets };

}),

);

console.log("arr = ", arr)

const balances: AccountBalances = {};

arr.forEach(({ *account*, *assets* }) => {

balances[*account*] = *assets*;

});

this.setState({ fetching: false, balances });

} catch (e) {

console.error(e);

this.setState({ fetching: false });

}

};

9.app请求签名

const result = await this.state.client.request({

topic: this.state.session.topic,

chainId,

request: {

method: "eth\_signTransaction",

params:[tx]

},

});

{

"topic": "368b38a3f68ff1bee64bbfe07f77f5d025dc6f383d270b51248da17b214c6141",

"chainId": "eip155:65534",

"request": {

"method": "eth\_signTransaction",

"params": [

{

"from": "0x8cb2c7Cc1a6E52Bc71651e1bC1e7354A02F4Eb07",

"to": "0x67e0E2EF7B56cC472B80112AC81849703766583B",

"data": "",

"gasPrice": "0x05312eff00",

"gasLimit": "0x5208",

"value": "0x01"

}

]

}

}

10.wallet 签名

this.state.client.on(

CLIENT\_EVENTS.session.request,

async (*requestEvent*: SessionTypes.RequestEvent) => {

*// tslint:disable-next-line*

const chainId = *requestEvent*.chainId || this.state.chains[0];

const [namespace] = chainId.split(":");

try {

*// TODO: needs improvement*

const requiresApproval = this.state.jsonrpc[namespace].methods.sign.includes(

*requestEvent*.request.method,

);

if (requiresApproval) {

this.setState({ requests: [...this.state.requests, *requestEvent*] });

} else {

const result = await this.state.wallet.request(*requestEvent*.request, { chainId });

const response = formatJsonRpcResult(*requestEvent*.request.id, result);

await this.respondRequest(*requestEvent*.topic, response);

}

} catch (e) {

console.error(e);

const response = formatJsonRpcError(*requestEvent*.request.id, (e as any).message);

await this.respondRequest(*requestEvent*.topic, response);

}

},

);