DIM Protocol

Release latest

Feb 19, 2020

Contents

This document introduces a new protocol designed for instant messaging (IM) and an architecture for developing decentralized IM applications. The software provides accounts (user identity recognition) and communications (IM) between accounts safely by end-to-end encryption.

CHAPTER 1

Translations

- Simplified Chinese

CHAPTER 2

Contents

2.1 DIMP Technical White Paper (V0.1)

November 11 2018

Abstract: This document introduces a new protocol designed for instant messaging (IM) and an architecture for developing decentralized IM applications. The software provides accounts (user identity recognition) and communications (IM) between accounts safely by end-to-end encryption.

2.2 Ming Ke Ming () – Identity Module

license Version

This document introduces a common **Identity Module** for decentralized user identity authentication.

  - *Number*

- *Samples*

2.2.1 0. Meta

The **Meta** was generated by your **private key**, it can be used to build a new ID for entity, or verify the ID/PK pair.

It consists of 4 fields:

| Field | Description | | ———— | ———————— | | version | Meta Algorithm Version | | seed | Entity Name | | key | Public Key | | fingerprint | Signature to generate address |

0.0. Version

Now it's value always equal to `0x01`.

0.1. Seed

A string as same as **ID.name** for generate the fingerprint.

0.2. Key

A **public key** (PK) was binded to an ID by the **Meta Algorithm**.

0.3. Fingerprint

THe **fingerprint** field was generated by your **private key** and **seed**:

```
fingerprint = sign(seed, SK);
```

2.2.2 1. ID

The **ID** is used to identify an **entity**(account/group). It consists of 3 fields and 2 extended properties:

| Field | Description | | ———— | ———————— | | name | Same with meta.seed | | address | Unique Identification | | terminal | Login point, it's optional. | | type | Network type | | number | Search Number |

The ID format is `name@address[/terminal]`.

1.0. Type

The **network type** of a person is `8`, and group is `16`:

```
// Network ID
enum {
    MKMNetwork_Main  = 0x08, // (Person)
    MKMNetwork_Group = 0x10, // (Multi-Persons)
};
```

1.1. Name

The **Name** field is a username, or just a random string for group:

1. The length of name must more than 1 byte, less than 32 bytes;
2. It should be composed by a-z, A-Z, 0-9, or charactors '_', '-', '.';
3. It cannot contain key charactors('@', '/').

```
// Name examples
userName  = "Albert.Moky";
groupName = "Group-1234567890";
```

1.2. Address

The **Address** field was created with the **Fingerprint** in Meta and a **Network ID**:

```
// Address algorithm
function btcBuildAddress(fingerprint, network) {
    hash       = ripemd160(sha256(fingerprint));
    check_code = sha256(sha256(network + hash)).prefix(4);
    address    = base58(network + hash + code);
    return address;
}
```

When you get a meta for the entity ID from the network, you must verify it with the consensus algorithm before accept its **key**.

```
// Meta algorithm
function isMatch(ID, meta) {
    // 1. check 'seed', 'key' & 'fingerprint' in meta with ID.name
    if (meta.seed != ID.name) {
        return false;
    }
    if (!verify(meta.seed, meta.fingerprint, meta.key)) {
        return false;
    }

    // 2. build address with meta, compare it with ID.address
    address = btcBuildAddress(meta.fingerprint, ID.address.network);
    if (address != ID.address) {
        return false;
    }

    // 3. if all of the above matches, get public key from meta
    ID.publicKey = meta.key;
    return true;
}
```

1.3. Terminal

A resource identifier as **Login Point**.

1.4. Number

A **Search Number** is defined for easy remember. Its value is converted from the **check code** of the address. It's greater than **0** and smaller than **232 (4,294,967,296)**.

2.2.3 2. Samples

ID

```
/* ID examples */
ID1 = "hulk@4YeVEN3aUnvC1DNUufCq1bs9zoBSJTzVEj"; // Immortal Hulk
ID2 = "moki@4WDfe3zZ4T7opFSi3iDAKiuTnUHjxmXekk"; // Monkey King
```

Meta

```
/* Meta example: hulk@4YeVEN3aUnvC1DNUufCq1bs9zoBSJTzVEj */
{
    version     : 0x01,
    seed        : "hulk",
    key         : {
        algorithm  : "RSA",
        data       : "-----BEGIN PUBLIC KEY-----
↪\nMIGJAoGBALB+vbUK48UU9rjlgnohQowME+3JtTb2hLPqtatVOW364/EKFq0/
↪PSdnZVE9V2Zq+pbX7dj3nCS4pWnYf40ELH8wuDm0Tc4jQ70v4LgAcdy3JGTnWUGiCsY+0Z8kNzRkm3FJid592FL7ryzfvIzB9bj
↪----END PUBLIC KEY-----",
        // other parameters
        keySize    : 1024,
        encryption : "PKCS1",
        signature  : "PKCS1v15SHA256"
    },
    fingerprint : "jIPGWpWSbR/
↪DQH6ol3t9DSFkYroVHQDvtbJErmFztMUP2DgRrRSNWuoKY5Y26qL38wfXJQXjYiWqNWKQmQe/
↪gK8M8NkU7lRwm+2nh9wSBYV6Q4WXsCboKbnM0+HVn9Vdfp21hMMGrxTX1pBPRbi0567ZjNQC8ffdW2WvQSoec2I=
↪"
}
```

(All data encode with **BASE64** algorithm as default, excepts the **address**)

2.3 Dao Ke Dao () – Message Module

license Version

This document introduces a common **Message Module** for decentralized instant messaging.

- *Envelope*
  - Sender
  - Receiver
  - Time
- *Content*

  - *Type*
  - Serial Number
- *Message*
  - *Instant Message*
  - *Secure Message*
  - *Reliable Message*

2.3.1 0. Envelope

Message Envelope

```
/* example */
{
    sender   : "moki@4WDfe3zZ4T7opFSi3iDAKiuTnUHjxmXekk",
    receiver : "hulk@4YeVEN3aUnvC1DNUufCq1bs9zoBSJTzVEj",
    time     : 1545405083
}
```

2.3.2 1. Content

```
/* example */
{
    type     : 0x01,      // message type
    sn       : 412968873, // serial number (message ID in conversation)

    text     : "Hey guy!"
}
```

Message Content Type

```
enum {
    DIMMessageType_Unknown = 0x00,

    DIMMessageType_Text    = 0x01,

    DIMMessageType_File    = 0x10,
    DIMMessageType_Image   = 0x12, // photo
    DIMMessageType_Audio   = 0x14, // voice
    DIMMessageType_Video   = 0x16,

    DIMMessageType_Page    = 0x20, // web page

    // quote an exists message and reply it with text
    DIMMessageType_Quote   = 0x37,

    // system command
    DIMMessageType_Command = 0x88,

    // top-secret message forwarded by proxy(account or station)
```

(continues on next page)

(continued from previous page)

```
    DIMMessageType_Forward = 0xFF
};
```

2.3.3 2. Message

When the user want to send out a message, the client needs TWO steps before sending it:

1. Encrypt the **Instant Message** to **Secure Message**;
2. Sign the **Secure Message** to **Reliable Message**.

Accordingly, when the client received a message, it needs TWO steps to extract the content:

1. Verify the **Reliable Message** to **Secure Message**;
2. Decrypt the **Secure Message** to **Instant Message**.

Instant Message

```
/* example */
{
    //-------- head (envelope) --------
    sender   : "moki@4WDfe3zZ4T7opFSi3iDAKiuTnUHjxmXekk",
    receiver : "hulk@4YeVEN3aUnvC1DNUufCq1bs9zoBSJTzVEj",
    time     : 1545405083,

    //-------- body (content) ---------
    content  : {
        type : 0x01,      // message type
        sn   : 412968873, // serial number (ID)
        text : "Hey guy!"
    }
}
```

content -> JsON string: `{"sn":412968873,"text":"Hey guy!","type":1}`

Secure Message

```
/**
 *  Algorithm:
 *      string = json(content);
 *      PW     = random();
 *      data   = encrpyt(string, PW);      // Symmetric
 *      key    = encrypt(PW, receiver.PK); // Asymmetric
 */
{
    //-------- head (envelope) --------
    sender   : "moki@4WDfe3zZ4T7opFSi3iDAKiuTnUHjxmXekk",
    receiver : "hulk@4YeVEN3aUnvC1DNUufCq1bs9zoBSJTzVEj",
    time     : 1545405083,

    //-------- body (content) ---------
    data     : "9cjCKG99ULCCxbL2mkc/MgF1saeRqJaCc+S12+HCqmsuF7TWK61EwTQWZSKskUeF",
```

(continues on next page)

(continued from previous page)

```
    key      : "WH/wAcu+HfpaLq+vRblNnYufkyjTm4FgYyzW3wBDeRtXs1TeDmRxKVu7nQI/
↪sdIALGLXrY+O5mlRfhU8f8TuIBilZUlX/eIUpL4uSDYKVLaRG9pOcrCHKevjUpId9x/
↪8KBEiMIL5LB0Vo7sKrvrqosCnIgNfHbXMKvMzwcqZEU8="
}
```

Reliable Message

```
/**
 *  Algorithm:
 *      signature = sign(data, sender.SK);
 */
{
    //-------- head (envelope) --------
    sender   : "moki@4WDfe3zZ4T7opFSi3iDAKiuTnUHjxmXekk",
    receiver : "hulk@4YeVEN3aUnvC1DNUufCq1bs9zoBSJTzVEj",
    time     : 1545405083,

    //-------- body (content) ---------
    data      : "9cjCKG99ULCCxbL2mkc/MgF1saeRqJaCc+S12+HCqmsuF7TWK61EwTQWZSKskUeF",
    key       : "WH/wAcu+HfpaLq+vRblNnYufkyjTm4FgYyzW3wBDeRtXs1TeDmRxKVu7nQI/
↪sdIALGLXrY+O5mlRfhU8f8TuIBilZUlX/eIUpL4uSDYKVLaRG9pOcrCHKevjUpId9x/
↪8KBEiMIL5LB0Vo7sKrvrqosCnIgNfHbXMKvMzwcqZEU8=",
    signature : "Yo+hchWsQlWHtc8iMGS7jpn/i9pOLNq0E3dTNsx80QdBboTLeKoJYAg/
↪lI+kZL+g7oWJYpD4qKemOwzI+9pxdMuZmPycG+0/VM3HVSMcguEOqOH9SElp/
↪fYVnm4aSjAJk2vBpARzMT0aRNp/jTFLawmMDuIlgWhBfXvH7bT7rDI="
}
```

(All data encode with **BASE64** algorithm as default)