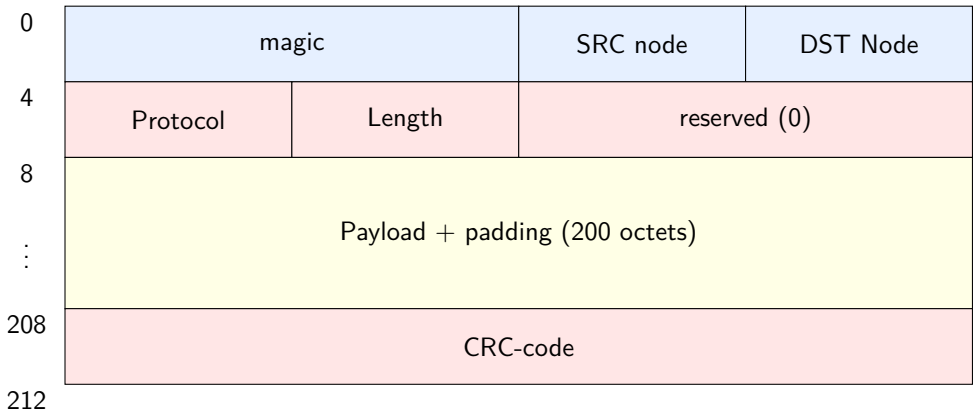
Lownet Protocol – The (In)Complete Specification

Revision 1.0 (date 2.10.2024)

1 Packet format v1

The lownet protocol supports both unencrypted and encrypted communication. The unencrypted (plaintext) packets are as follows (packet format v1):



Note that:

- Each row corresponds to 32 bits.
- Magic is a fixed two octet identifier: `0x10`, `0x4e`
- `Source` and `Destination` node id's are one byte each (`uint8_t`). For broadcast messages the destination field is set to `0xFF`.
- `Reserved` field shall be set to zero.
- `Protocol` defines the application.
- `Length` corresponds to the length of actual data in the payload.
- `CRC` computation is based on the given 24-bit CRC generator polynomial,

$$G(x) = x^{24} + x^{10} + x^6 + x^5 + x + 1,$$

  so that the 8 MSBs are zero (32 bits reserved for CRC code in the packet).

1.1 Time protocol (0x01)

The *the network time* is provided by the master node using the following frame format:

- `Destination` field is (typically) `0xFF`, i.e., this information is meant for everyone.
- `Protocol` field has value `0x01`
- `Length` has value 5 (5 octets)
- `Payload` contains the time information (time stamp):

seconds, `uint32_t` :	4 octets
partial (1/256 seconds) :	1 octet

1.2 Chat protocol (0x02)

The Chat protocol is essentially instant messaging protocol:

- `Destination` field is either `0xFF` (CHAT) or a node id (TELL).
- `Protocol` field has value `0x02`
- `Length` field defines the length of the actual message.
- `Payload` contains the text message (one line of text with newline or a null terminator):

message : 0-192 octets

1.3 Ping protocol (0x03)

Ping messages shall be responded with "Pong" messages, both sharing the same message format:

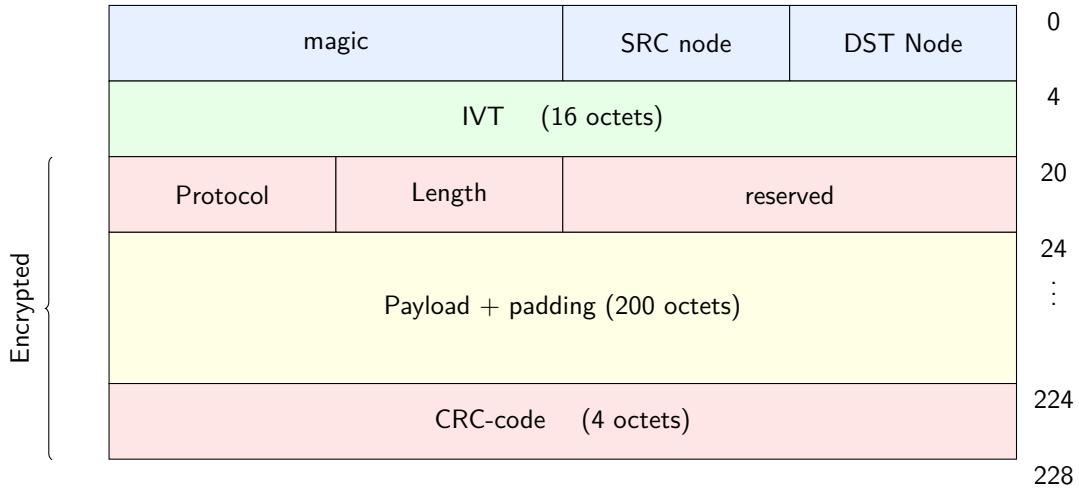
- `Destination` field defines the recipient(s). Broadcast pings are allowed, but the response shall always be an unicast.
- `Protocol` field has value `0x03`
- `Length` field has a value of 11 or more (see below).
- `Payload` is as follows:

`lownet_time_t`:	time stamp of the origin (5 bytes)
`lownet_time_t`:	time stamp of the response (5 bytes)
`uint8_t`:	node id of the origin (the node that initiated the ping)
`optional`:	any optional data

The payload may contain additional data that the node responding to the ping shall keep intact.

2 Packet format v2

The encrypted packets (packet format v2) is as follows:



Note that:

- Green block, IVT for AES encryption, should be random numbers (for each packet!).
- Once the encryption is removed (decryption + relocation), the packet is an ordinary v1 packet.

2.1 Operation when Encryption is Enabled

When encryption is enabled, the lownet device shall ignore the original (insecure) packets and processes only the encypted packets.

The encryption algorithm is the standard AES-256 in CBC mode. Each encrypted packet includes a 16-byte long IVT that is used to initilize the encryption/decryption. The default AES key is

```
uint32_t *aes_key = { 0xc0c71cc5, 0x748ce81a, 0x4b0e4aa7, 0x70c0d55e,
                      0x58957e01, 0xed51d8cc, 0x26b844c4, 0x49c50530 };
```

Upon receiving an encrypted packet, the network layer shall decode it using the AES-256 with the current key, which will recover the plaintext packet. The CRC-code shall then be verified.

Upon transmission the reverse occurs, i.e., plaintext v1 packet frames will be encrypted and encapsulated to v2 frame.

2.2 Digital Signature: RSA and SHA-256

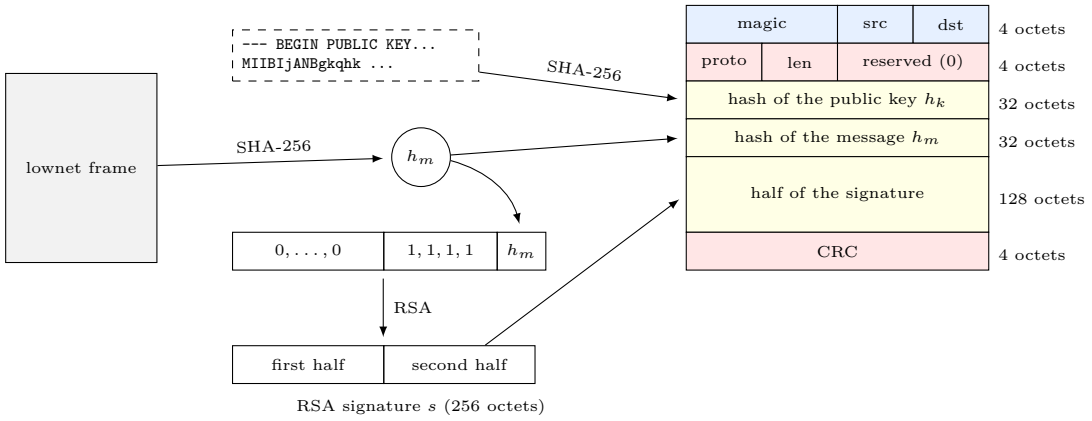
Each node shall store the master node's 2048-bit public RSA key d (available in the source files), that is used to validate digital signatures.

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAxG9VF9wuocepQnwBkxUb
4YxCo1NJ1MAKAGoaK2csfPABSRkjlESev42rFVzejGtOp2pxKcyihDXVe1BEzD0q
HXxEgtkRy0/bJNhGxoMmWTbikO3BmIMIO9zIk3leaNtyy49U27CKDgUHOPp6zd3c
dgD3nE4fIE7tU3mCJ4xh5xMHeyoqa/MV3EkE9VDV2vCTP3KyKDFObYqig6XWydeQ
CPmSAr0rRYiriguOvQGGxPeaCWPaUAG+t2W7ydpeju+Dkzl6NHm0q9JdLfpg8zje
BgLekdFxyM4jAK2hCX+vswUrYqbm5m9rptxQUuSYpk27Ew7uWRaomAWWeMLIg+zt
rwIDAQAB
-----END PUBLIC KEY-----
```

The protocol for *signed packets* will be described next. We reserve two highest bits from the protocol field to indicate that the current frame is part of the signed frame:

proto	
00xxxxxx	normal frame (no signature)
01xxxxxx	normal frame, signature will follow
10xxxxxx	the first part of the signature
11xxxxxx	the second part of the signature

The signature frames have the following format:



- *Hash of the public key*, h_k, is the result from SHA-256 for the PEM formatted public RSA key as given in the header file (see the text string below). This is thus a constant for the time being.[1]
- *Hash of the message*, h_m, is the result of SHA-256 for the whole v1 lownet frame (200 bytes).
- *RSA operation* is carried out for the message hash h_m. As the input array for the 2048-bit RSA consists of 256 bytes, the 32-byte hash value h_m shall be prepended with 220 zeroes and 4 ones as follows: yielding the 256-byte "signature block" m_s:

0 . . . 0 (220 octets)	1 . . . 1 (4 octets)	message hash h_m (32 octets)	(256 octets)

  - The zeroes at the start ensure[2] that $s < n$, where n is the modulus, $n = p_1 \cdot p_2$.
  - The four ones ensure that the signed message is never zero.
  - Otherwise the above choice is rather arbitrary . . .
  - Master node computes the RSA operation with its private key e, $s = \mathrm{RSA}(m_s, e) = (m_s)^e \bmod n$, and includes two halves of $s = s_1 : s_2$ in the two signature frames.

[1] Technical note: In principle, the system can support more than one authority, each with their own published key. The public key hash in the signature frames is used to identify the key used for signing each message.

[2] We can assume that typically n is slightly less than 2^{2048}. In our case, the highest byte in n is `0xC4`, i.e., $196 \cdot 2^{2040} < n < 197 \cdot 2^{2040}$. If we placed h_m at the start of m_s, and padded the end (e.g.) with zeroes, then all hash values h_m starting with a byte higher than `0xC4` yield a signature message m_s higher than n.

2.3 Signature Verification

The network layer of an ordinary node only needs to verify digital signatures it receives. It shall buffer the packet m and compute the SHA-256 hash h_m for it.

Matching signature packets are identified based on the message hash h_m. Also the source, destination and protocol field (apart from the upper 2 bits) must match. The node shall also check that it has a matching public key by comparing the key's hash to h_k. Signature packets with no matching message (h_m and the header) or key (h_k) shall be discarded immediately.

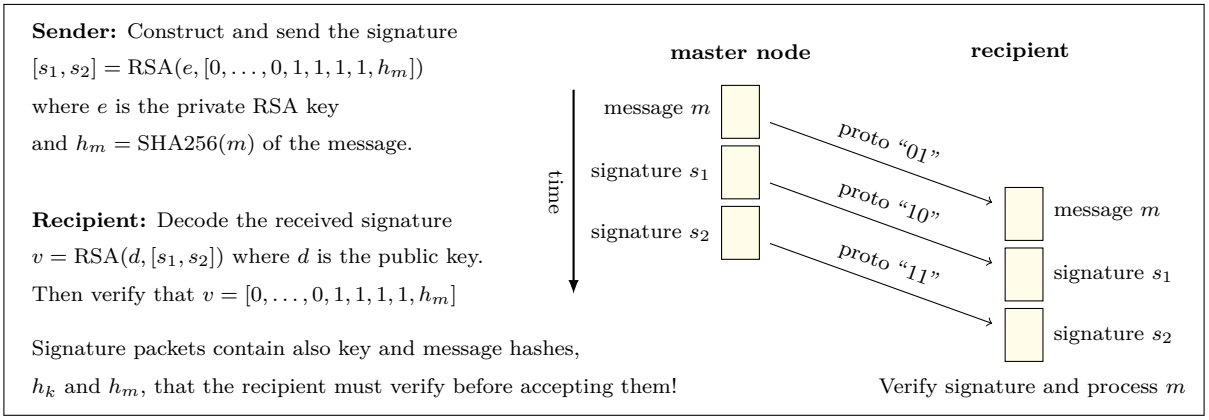
The matching signature must arrive within 10 seconds. If no valid signature frames are received by then, the packet is discarded.

At minimum, the node shall thus reserve space for three packets: the message itself and the two corresponding signature packets. The node MAY also reserve more space in order to support two or more concurrent signed messages.

If the complete signature $s = s_1 : s_2$ for a given m is received in time, the node computes the RSA operation with the public key d, $m'_s = \mathrm{RSA}(s, d) = s^d \bmod n$. Then it constructs the above 256-byte signature message m_s that also the recipient can compute based on the 200-byte message m it received. Only if these two match, i.e., when

$$m'_s = \mathrm{RSA}(s, d) = m_s,$$

the buffered packet m is forwarded up for processing. Hence, the communication pattern is as follows:



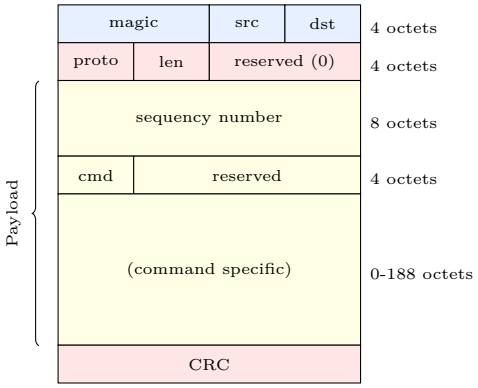
2.4 Time protocol

When encryption is enabled (v2), the (v1) time protocol is deprecated due security concerns. A node **shall not** accept any time updates via the old protocol! Time information will be provided via a new command protocol (see below).

2.5 Command protocol (0x04)

The lownet frame with command protocol is as follows:

- `Protocol` field has value `0x04`
- `Length` field defines the length of the actual message (12, or more)
- `Payload` contains the sequence number, command and some optional command-specific data.



Commands not adhering with the following conditions shall be discarded:

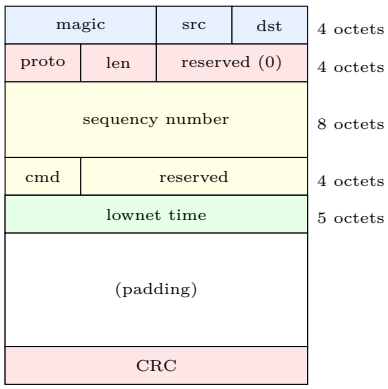
1. All packets shall be signed with the known public key (see section 2.2). Commands without a valid signature shall be discarded.
2. All packets shall have a strictly increasing sequence number. Commands with invalid sequence number shall be discarded.

Note that nodes have to keep track of the (valid) sequence number.

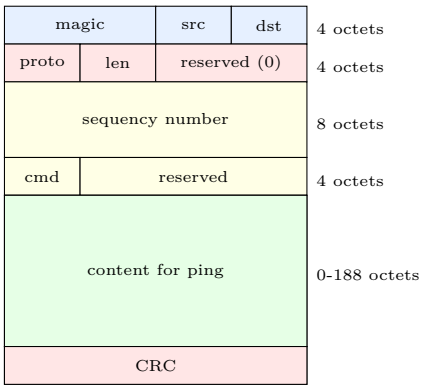
2.6 Command Types:

The command protocol has the following commands:

- **Time** (`0x01`): The payload shall have the time (`lownet_time_t`, 5 octets). The node shall adjust to this network time.
- **Test** (`0x02`): Test command requests the destination node to ping the master node. The command specific data may contain arbitrary content that shall be copied as it is in the optional payload of the ping message (see section 1.3). In this case, the `length` field shall have a value greater than 12, accordingly.



i) Time command



i) Test command