---

title: Recursive-length prefix (RLP) serialization description: A definition of the rlp encoding in Ethereum's execution layer. lang: en sidebarDepth: 2

Recursive Length Prefix (RLP) serialization is used extensively in Ethereum's execution clients. RLP standardizes the transfer of data between nodes in a space-efficient format. The purpose of RLP is to encode arbitrarily nested arrays of binary data, and RLP is the primary encoding method used to serialize objects in Ethereum's execution layer. The only purpose of RLP is to encode structure; encoding specific data types (e.g. strings, floats) is left up to higher-order protocols; but positive RLP integers must be represented in big-endian binary form with no leading zeroes (thus making the integer value zero equivalent to the empty byte array). Deserialized positive integers with leading zeroes get treated as invalid. The integer representation of string length must also be encoded this way, as well as integers in the payload.

More information in the Ethereum yellow paper (Appendix B)

To use RLP to encode a dictionary, the two suggested canonical forms are:

- use `[[k1,v1],[k2,v2]...]` with keys in lexicographic order
- use the higher-level Patricia Tree encoding as Ethereum does

Definition {#definition}

The RLP encoding function takes in an item. An item is defined as follows：

- a string (i.e. byte array) is an item
- a list of items is an item

For example, all of the following are items:

- an empty string;
- the string containing the word "cat";
- a list containing any number of strings;
- and a more complex data structures like `["cat", ["puppy", "cow"], "horse", [[]], "pig", [""], "sheep"]`.

Note that in the context of the rest of this page, 'string' means "a certain number of bytes of binary data"; no special encodings are used, and no knowledge about the content of the strings is implied.

RLP encoding is defined as follows:

- For a single byte whose value is in the `[0x00, 0x7f]` (decimal `[0, 127]`) range, that byte is its own RLP encoding.
- Otherwise, if a string is 0-55 bytes long, the RLP encoding consists of a single byte with value **0x80** (dec. 128) plus the length of the string followed by the string. The range of the first byte is thus `[0x80, 0xb7]` (dec. `[128, 183]`).
- If a string is more than 55 bytes long, the RLP encoding consists of a single byte with value **0xb7** (dec. 183) plus the length in bytes of the length of the string in binary form, followed by the length of the string, followed by the string. For example, a 1024 byte long string would be encoded as `\xb9\x04\x00` (dec. `185, 4, 0`) followed by the string. Here, `0xb9` (183 + 2 = 185) as the first byte, followed by the 2 bytes `0x0400` (dec. 1024) that denote the length of the actual string. The range of the first byte is thus `[0xb8, 0xbf]` (dec. `[184, 191]`).
- If the total payload of a list (i.e. the combined length of all its items being RLP encoded) is 0-55 bytes long, the RLP encoding consists of a single byte with value **0xc0** plus the length of the payload followed by the concatenation of the RLP encodings of the items. The range of the first byte is thus `[0xc0, 0xf7]` (dec. `[192, 247]`).
- If the total payload of a list is more than 55 bytes long, the RLP encoding consists of a single byte with value **0xf7** plus the length in bytes of the length of the payload in binary form, followed by the length of the payload, followed by the concatenation of the RLP encodings of the items. The range of the first byte is thus `[0xf8, 0xff]` (dec. `[248, 255]`).

In code, this is:

```python def rlp_encode(input): if isinstance(input,str): if len(input) == 1 and ord(input) < 0x80: return input return encode_length(len(input), 0x80) + input elif isinstance(input, list): output = '' for item in input: output += rlp_encode(item)
```

return encode_length(len(output), 0xc0) + output

def encode_length(L, offset): if L < 56: return chr(L + offset) elif L < 256**8: BL = to_binary(L) return chr(len(BL) + offset + 55) + BL raise Exception("input too long")

def to_binary(x): if x == 0: return '' return to_binary(int(x / 256)) + chr(x % 256) ```

Examples {#examples}

- the string "dog" = [0x83, 'd', 'o', 'g']
- the list ["cat", "dog"] = `[ 0xc8, 0x83, 'c', 'a', 't', 0x83, 'd', 'o', 'g' ]`
- the empty string ('null') = `[ 0x80 ]`
- the empty list = `[ 0xc0 ]`
- the integer 0 = `[ 0x80 ]`
- the encoded integer 0 ('\x00') = `[ 0x00 ]`
- the encoded integer 15 ('\x0f') = `[ 0x0f ]`
- the encoded integer 1024 ('\x04\x00') = `[ 0x82, 0x04, 0x00 ]`
- the set theoretical representation of three, `[ [], [[]], [ [], [[]] ] ] = [ 0xc7, 0xc0, 0xc1, 0xc0, 0xc3, 0xc0, 0xc1, 0xc0 ]`
- the string "Lorem ipsum dolor sit amet, consectetur adipisicing elit" = `[ 0xb8, 0x38, 'L', 'o', 'r', 'e', 'm', ' ', ... , 'e', 'l', 'i', 't' ]`

RLP decoding {#rlp-decoding}

According to the rules and process of RLP encoding, the input of RLP decode is regarded as an array of binary data. The RLP decoding process is as follows:

1. according to the first byte (i.e. prefix) of input data and decoding the data type, the length of the actual data and offset;
2. according to the type and offset of data, decode the data correspondingly;
3. continue to decode the rest of the input;

Among them, the rules of decoding data types and offset is as follows:

1. the data is a string if the range of the first byte (i.e. prefix) is [0x00, 0x7f], and the string is the first byte itself exactly;
2. the data is a string if the range of the first byte is [0x80, 0xb7], and the string whose length is equal to the first byte minus 0x80 follows the first byte;
3. the data is a string if the range of the first byte is [0xb8, 0xbf], and the length of the string whose length in bytes is equal to the first byte minus 0xb7 follows the first byte, and the string follows the length of the string;
4. the data is a list if the range of the first byte is [0xc0, 0xf7], and the concatenation of the RLP encodings of all items of the list which the total payload is equal to the first byte minus 0xc0 follows the first byte;
5. the data is a list if the range of the first byte is [0xf8, 0xff], and the total payload of the list whose length is equal to the first byte minus 0xf7 follows the first byte, and the concatenation of the RLP encodings of all items of the list follows the total payload of the list;

In code, this is:

```python def rlp_decode(input): if len(input) == 0: return output = '' (offset, dataLen, type) = decode_length(input) if type is str: output = instantiate_str(substr(input, offset, dataLen)) elif type is list: output = instantiate_list(substr(input, offset, dataLen)) output += rlp_decode(substr(input, offset + dataLen)) return output

def decode_length(input): length = len(input) if length == 0: raise Exception("input is null") prefix = ord(input[0]) if prefix <= 0x7f: return (0, 1, str) elif prefix <= 0xb7 and length > prefix - 0x80: strLen = prefix - 0x80 return (1, strLen, str) elif prefix <= 0xbf and length > prefix - 0xb7 and length > prefix - 0xb7 + to_integer(substr(input, 1, prefix - 0xb7)): lenOfStrLen = prefix -
```

0xb7 strLen = to_integer(substr(input, 1, lenOfStrLen)) return (1 + lenOfStrLen, strLen, str) elif prefix <= 0xf7 and length > prefix - 0xc0: listLen = prefix - 0xc0; return (1, listLen, list) elif prefix <= 0xff and length > prefix - 0xf7 and length > prefix - 0xf7 + to_integer(substr(input, 1, prefix - 0xf7)): lenOfListLen = prefix - 0xf7 listLen = to_integer(substr(input, 1, lenOfListLen)) return (1 + lenOfListLen, listLen, list) raise Exception("input does not conform to RLP encoding form")

def to_integer(b): length = len(b) if length == 0: raise Exception("input is null") elif length == 1: return ord(b[0]) return ord(substr(b, -1)) + to_integer(substr(b, 0, -1)) * 256 ```

Further reading {#further-reading}

- RLP in Ethereum
- Ethereum under the hood: RLP
- Coglio, A. (2020). Ethereum's Recursive Length Prefix in ACL2. arXiv preprint arXiv:2009.13769.

Related topics {#related-topics}

- Patricia merkle trie