Polkadot Runtime

Protocol Specification

Contents

Chapter 1

Extrinsics

1.1 Introduction

An extrinsic is a SCALE encoded array consisting of a version number, signature, and varying data types indicating the resulting Runtime function to be called, including the parameters required for that function to be executed.

1.2 Preliminaries

Definition 1 *An extrinsic , tx, is a tuple consisting of the extrinsic version, T_v (Def. 2), and the body of the extrinsic, T_b.*

$$tx := (T_v, T_b)$$

The value of T_b varies for each version. The current version 4 is described in section 1.3.1.

Definition 2 *T_v is a 8-bit bitfield and defines the extrinsic version. The required format of an extrinsic body, T_b, is dictated by the Runtime. Older or unsupported version are rejected.*

The first bit of T_v indicates whether the transaction is ***signed*** *(1) or* ***unsigned*** *(0). The remaining 7-bits represent the version number. As an example, for extrinsic format version 4, an signed extrinsic represents T_v as* `132` *while a unsigned extrinsic represents it as* `4`*.*

1.3 Extrinsics Body

1.3.1 Version 4

Version 4 of the Polkadot extrinsic format is defined as follows:

$$T_b := (A_i, Sig, E, M_i, F_i(m))$$

where each values represents:

- A_i: the 32-byte address of the sender (Def. 3).
- Sig: the signature of the sender (Def. 4).
- E: the extra data for the extrinsic (Def. 5).
- M_i: the indicator of the Polkadot module (Def. 7).
- $F_i(m)$: the indicator of the function of the Polkadot module (Def. 8).

Definition 3 *Account Id, A_i, is the 32-byte address of the sender of the extrinsic as described in the external SS58 address format.*

Definition 4 *The signature, Sig, is a varying data type indicating the used signature type, followed by the signature created by the extrinsic author. The following types are supported:*

$$Sig := \begin{cases} 0, & \textit{Ed25519, followed by: } (b_0, ..., b_{63}) \\ 1, & \textit{Sr25519, followed by: } (b_0, ..., b_{63}) \\ 2, & \textit{Ecdsa, followed by: } (b_0, ..., b_{64}) \end{cases}$$

Signature types vary in sizes, but each individual type is always fixed-size and therefore does not contain a length prefix. `Ed25519` *and* `Sr25519` *signatures are 512-bit while* `Ecdsa` *is 520-bit, where the last 8 bits are the recovery ID.*

The signature is created by signing payload P.

$$P := \begin{cases} Raw, & \textit{if } |Raw| \leq 256 \\ Blake2(Raw), & \textit{if } |Raw| > 256 \end{cases} \tag{1.1}$$
$$Raw := (M_i, F_i(m), E, R_v, F_v, H_h(G), H_h(B))$$

where each value represents:

- *M_i: the module indicator (Def. 7).*
- *$F_i(m)$: the function indicator of the module (Def. 8).*
- *E: the extra data (Def. 5).*
- *R_v: a UINT32 containing the specification version of* `14`*.*
- *F_v: a UINT32 containing the format version of* `2`*.*
- *$H_h(G)$: a 32-byte array containing the genesis hash.*
- *$H_h(B)$: a 32-byte array containing the hash of the block which starts the mortality period, as described in Definition 6.*

Definition 5 *Extra data, E, is a tuple containing additional meta data about the extrinsic and the system it is meant to be executed in.*

$$E := (T_{mor}, N, P_t)$$

where each value represents:

- T_{mor}: *contains the SCALE encoded mortality of the extrinsic (Def. 6).*
- N: *a compact integer containing the nonce of the sender. The nonce must be incremented by one for each extrinsic created, otherwise the Polkadot network will reject the extrinsic.*
- P_t: *a compact integer containing the transactor pay including tip.*

Definition 6 *Extrinsic **mortality** is a mechanism which ensures that an extrinsic is only valid within a certain period of the ongoing Polkadot lifetime. Extrinsics can also be immortal, as clarified in Section 6.*

The mortality mechanism works with two related values:

- M_{per}: *the period of validity in terms of block numbers from the block hash specified as $H_h(B)$ in the payload (Def. 4). The requirement is $M_{per} \geq 4$ and M_{per} must be the power of two, such as* `32`, `64`, `128`, *etc.*
- M_{pha}: *the phase in the period that this extrinsic's lifetime begins. This value is calculated with a formula and validators can use this value in order to determine which block hash is included in the payload. The requirement is $M_{pha} < M_{per}$.*

In order to tie a transaction's lifetime to a certain block ($H_i(B)$) after it was issued, without wasting precious space for block hashes, block numbers are divided into regular periods and the lifetime is instead expressed as a "phase" (M_{pha}) from these regular boundaries:

$$M_{pha} = H_i(B) \ mod \ M_{per}$$

M_{per} and M_{pha} are then included in the extrinsic, as clarified in Definition 5, in the SCALE encoded form of T_{mor} (Sect. 6). Polkadot validators can use M_{pha} to figure out the block hash included in the payload, which will therefore result in a valid signature if the extrinsic is within the specified period or an invalid signature if the extrinsic "died".

Example

The extrinsic author choses $M_{per} = 256$ at block `10'000`*, resulting with $M_{pha} = 16$. The extrinsic is then valid for blocks ranging from* `10'000` *to* `10'256`.

Encoding

T_{mor} refers to the SCALE encoded form of type M_{per} and M_{pha}. T_{mor} is the size of two bytes if the extrinsic is considered mortal, or simply one bytes with the value equal to zero if the extrinsic is considered immortal.

$$T_{mor} := Enc_{SC}(M_{per}, M_{pha})$$

The SCALE encoded representation of mortality T_{mor} deviates from most other types, as it's specialized to be the smallest possible value, as described in Algorithm 1 and 2.

Algorithm 1 ENCODE MORTALITY

Input: M_{per}, M_{pha}

// If the extrinsic is immortal, specify
// a single byte with the value equal to zero.

return $\begin{cases} 0 & if\ extrinsic\ is\ immortal \end{cases}$

Init $factor = \textsc{Limit}(M_{per} >> 12,\ 1,\ \phi)$
Init $left = \textsc{Limit}(\mathrm{TZ}(M_{per}) - 1,\ 1,\ 15)$
Init $right = \frac{M_{pha}}{factor} << 4$

// Returns a two byte value
return $left|right$

Algorithm 2 DECODE MORTALITY

Input: T_{mor}

return $\begin{cases} Immortal & if\ T_{mor}^{b0} = 0 \end{cases}$

Init $enc = T_{mor}^{b0} + (T_{mor}^{b1} << 8)$
Init $M_{per} = 2 << (enc\ mod\ (1 << 4))$
Init $factor = \textsc{Limit}(M_{per} >> 12,\ 1,\ \phi)$
Init $M_{pha} = (enc >> 4) * factor$
return (M_{per}, M_{pha})

- T_{mor}^{b0}*: the first byte of* T_{mor}.
- T_{mor}^{b1}*: the second byte of* T_{mor}.
- $\textsc{Limit}(num,\ min,\ max)$*: Ensures that num is between min and max. If min or max is defined as* ϕ*, then there is no requirement for the specified minimum/maximum.*
- $\mathrm{TZ}(num)$*: returns the number of trailing zeros in the binary representation of num. For example, the binary representation of* `40` *is* `0010 1000`*, which has three trailing zeros.*
- $>>$*: performs a binary right shift operation.*
- $<<$*: performs a binary left shift operation.*
- $|$ *: performs a bitwise OR operation.*

Definition 7 *M_i is an indicator for the Runtime to which Polkadot module, m, the extrinsic should be forwarded to.*

M_i is a varying data type pointing to every module exposed to the network.

$$M_i := \begin{cases} 0, & System \\ 1, & Utility \\ ... & \\ 7, & Balances \\ ... & \end{cases}$$

Definition 8 *$F_i(m)$ is a tuple which contains an indicator, m_i, for the Runtime to which function within the Polkadot module, m, the extrinsic should be forwarded to. This indicator is followed by the concatenated and SCALE encoded parameters of the corresponding function, params.*

$$F_i(m) := (m_i, params)$$

The value of m_i varies for each Polkadot module, since every module offers different functions. As an example, the `Balances` *module has the following functions:*

$$Balances_i := \begin{cases} 0, & transfer \\ 1, & set\_balance \\ 2 & force\_transfer \\ 3 & transfer\_keep\_alive \end{cases}$$