keys $\mathcal{V}$, a signature is considered valid if *all* key owners participate. However, such multi-signature schemes together with a simple script-like DSL can achieve the properties defined in Figure 4 while still providing the benefits of a simple verification procedure, and a smaller signature size.

B Binary Address Format

Excepting bootstrap addresses, the binary format for every address has a 1-byte header, followed by a payload (bootstrap addresses use the binary encoding from the Byron era). The header is composed of two nibbles (two 4-bit segments), indicating the address type and the network id.

$$\begin{aligned} \mathsf{address} &= \mathsf{header\_byte} | \mathsf{payload} \\ &= \mathsf{addr\_type\_nibble} | \mathsf{network\_nibble} | \mathsf{payload} \end{aligned}$$

B.1 Header, first nibble

The first nibble of the header specifies the type of the address. Bootstrap addresses can also be identified by the first nibble.

address type	payment credential	stake credential	header, first nibble
base address	keyhash	keyhash	0000
base address	scripthash	keyhash	0001
base address	keyhash	scripthash	0010
base address	scripthash	scripthash	0011
pointer address	keyhash	ptr	0100
pointer address	scripthash	ptr	0101
enterprise address	keyhash	-	0110
enterprise address	scripthash	-	0111
bootstrap address	keyhash	-	1000
stake address	-	keyhash	1110
stake address	-	scripthash	1111
future formats	?	?	1001-1101

B.2 Header, second nibble

Excepting bootstrap addresses, the second nibble of the header specifies the network.

network	header, second nibble
testnets	0000
mainnet	0001
future networks	0010-1111

B.3 Header, examples

On a **testnet**, the header for a **pointer** address whose payment credential is a **keyhash** is:

00000100

On **mainnet**, the header for a **pointer** address whose payment credential is a **keyhash** is:

00010100

On **mainnet**, the header for a **pointer** address whose payment credential is a **scripthash** is:

00010101

B.4 Payloads

The payload for the different address types are as follows:

address type	payload
base address	two 28-byte bytestrings
pointer address	one 28-byte bytestring, and three variable-length unsigned integers
enterprise address	one 28-byte bytestrings
stake address	one 28-byte bytestrings

The variable-length encoding used in pointers addresses is the base-128 representation of the number, with the most significant bit of each byte indicating continuation (this is sometimes called a variable-length quantity, or VLQ). If the significant bit is 1, then another byte follows.

C Bootstrap Witnesses

C.1 Bootstrap Witnesses CBOR Specification

In the Byron era of Cardano, public keys are transmitted on chain as extended public keys as specified in [Wui12]. The Shelley era of Cardano does not use extended public keys, but does use the same cryptographic signing algorithm, namely Ed25519.

The extended public key consists of 64 bytes, the first 32 of which corresponds to the public key, the second 32 of which correpsonds to something called the chain code:

$$\mathsf{extended\_pubkey} = \mathsf{pubkey} | \mathsf{chain\_code}$$

The chaincode is required for constructing bootstrap witnesses.

The CBOR specification of bootstrap witnesses, named bootstrap_witness, is given in https://github.com/input-output-hk/cardano-ledger/tree/master/eras/shelley/impl/cddl-files.

In the Shelley era, only pubkey address are supported, and are named bootstrap addresses.

The bootstrap witness consists of the public key, the signature, the chain code, and the address attributes. As explained above, the public key and the signature format are the same as for all other witnesses in the Shelley era. The address attributes has the same format as from the Byron era address, as specified by addrattr in https://github.com/input-output-hk/cardano-ledger/blob/master/eras/byron/cddl-spec/byron.cddl.

C.2 Bootstrap Address ID Recovery

The bootstrap address ID, named addressid in the Byron CDDL specification above, can be computed as follows.

First construct the following byte string:

$$\mathsf{b} = \mathsf{0x830082005840} | \mathsf{pubkey\_bytes} | \mathsf{chain\_code\_bytes} | \mathsf{attributes\_bytes}$$

The address ID is then obtained by first applying sha3 256 to b and then applying blake2b44 to the result.

$$\mathsf{address\_id} = \mathsf{blake2b44}(\mathsf{sha3\_256\ b})$$

D CBOR Serialization Specification

Our CBOR (RFC 7049 [Bor13]) serialization scheme is specified using CDDL (RFC 8610 [Bir19]).

The CDDL specification is located at https://github.com/input-output-hk/cardano-ledger/tree/master/eras/shelley/test-suite/cddl-files.

Note that the ledger does **not support a canonical representation**. This is an intentional decision to discourage people from checking signatures and hashes on re-serialized data.

E Implementation of txSize

The minimum fee calculation in Figure 9 depends on an abstract txSize function. We have implemented txSize as the number of bytes in the CBOR serialization of the transaction, as defined in Appendix D.

F Proofs

For the proofs we use the automated theorem prover MetiTarski [AP10] which is specialized for proofs over real arithmetic, including elementary functions.

Proof. The property (**??**) (p. **??**) for the minimal refund can be proven automatically via

```
fof(minimal_refund, conjecture,
! [Dmin, Lambda, Delta, Dval] :
((Dmin : (=0,1=) & Lambda > 0 & Delta > 0 & Dval > 0
=>
Dval*Dmin >= 0 &
(Dval * (Dmin + (1 - Dmin) * exp(-Lambda * Delta))) : (=Dval * Dmin, Dval=)))).

fof(floor_lower_upper, conjecture,
! [X] :
(X >= 0 => X - 1 <= floor(X) & floor(X) <= X)).
```

`minimal_refund` shows that the resulting value is within the interval $[d_{val} \cdot d_{min}, d_{val}]$ and that $d_{val} \cdot d_{min}$ is non-negative, while `floor_lower_upper` shows that the floor of a value x has an upper bound x and lower bound $x - 1$.

□

Proof. For the property (15.19) (p. 105) for reward splitting we actually show a stronger one, by removing the floor function. Using the fractional values we get an upper bound for the real value and showing that this upper bound is bounded by $\hat{f}$ we show that the real value is also bounded by $\hat{f}$. To eliminate the sum, we use the identity $\frac{s+\sum_j t_j}{\sigma} = 1$, see the definition of σ in [SL-D1]. Using this, we show for $\hat{f} > c$

$$\begin{aligned} & 0 \leq c + (\hat{f} - c) \cdot (m + (1-m)) \cdot \tfrac{s}{\sigma} + \textstyle\sum_j (\hat{f} - c) \cdot (1-m) \cdot \tfrac{t_j}{\sigma} && \leq \hat{f} \\ \Leftrightarrow \quad & 0 \leq c + (\hat{f} - c) \cdot m \cdot \tfrac{s}{\sigma} + (\hat{f} - c) \cdot (1-m) \cdot \tfrac{s+\sum_j t_j}{\sigma} && \leq \hat{f} \\ \Leftrightarrow \quad & 0 \leq c + (\hat{f} - c) \cdot m \cdot \tfrac{s}{\sigma} + (\hat{f} - c) \cdot (1-m) && \leq \hat{f} \end{aligned}$$

This can be proven automatically using

```
fof(reward_splitting, conjecture,
! [C, F, M, S, Sigma] :
(
```

```
M : (=0, 1=) & C >= 0 & F > C & Sigma : (0, 1=) & S : (=0, Sigma=)
=>
C + (F - C) * M * S / Sigma + (F - C) * (1 - M) <= F &
0 <= C + (F - C) * M * S / Sigma + (F - C) * (1 - M))).
```

□