LTO NETWORK

LTO NETWORK

On-chain Identities and Credentials
Technical Implementation

Index

Transaction types

Publish Certificate

LTO will introduce a new transaction type to publish X.509 certificates. Nodes must not only check the validity of the transaction but also of the certificate itself. Transactions with invalid certificates will be rejected.

Publishing a certificate doesn't establish ownership to the signer of the transaction, nor does it establish any other kind of relationship between the publisher and the certificate. Proving ownership of the certificate can only be done with a private key that corresponds with the public key within the X.509 certificate.

The transaction contains the certificate in binary (DER) format. The maximum length of a certificate is 65535 bytes.

	Field name	Type	Length
1	Transaction multiple version mark	Byte (constant, value=0)	1
2	Transaction type	Byte (constant, value=20)	1
3	Sender's public key	PublicKeyAccount (Array[Byte])	32
4	Certificate length (L)	Short	2
5	Certificate	DER (Array[Byte])	L
6	Fee	Long	4
7	Timestamp	Long	4
8.1	Proofs version (1)	Byte	1
8.2	Proofs count	Short	2
8.3	Proof 1 length (P1)	Short	2
8.4	Proof 1	ByteStr (Array[Byte])	P1
...	...	...	...

Transaction types

Unlike other transactions on LTO Network, the ‘Publish Certificate’ transaction doesn’t have a fixed fee. Instead, the fee is calculated based on the size of the certificate (in bytes). This is similar to how the tx fee of Data transactions is calculated on Waves.

The fee is 4 LTO + 1 LTO per kilobyte (rounded up).

Root and Intermediate certificates

Unlike other transactions on LTO Network, the ‘Publish Certificate’ transaction doesn’t have a fixed fee. Instead, the fee is calculated based on the size of the certificate (in bytes). This is similar to how the tx fee of Data transactions is calculated on Waves.

The fee is 4 LTO + 1 LTO per kilobyte (rounded up).

Network addresses

Signing with RSA and ECDSA

RSA keys and signatures are significantly longer than those of ED25519. Signing with them would require creating new versions of each transaction type to accommodate the dynamic key and signature length.

LTO is opting for an alternative using the existing transaction data structures. Publishing an X.509 certificate will automatically create a smart account. The address is derived from the public key, but in a different way than with normal accounts.

Normally addresses are generated from the secure hash of the ED25519 public key

```
sha256(Blake2b256(ED25519_public_key))
```

For certificate smart accounts, the address is calculated as

```
sha256(Blake2b256(sha256(RSA_public_key)))
```

The sha256 hash of the RSA (or ECDSA) key can be used in the PublicKey field of the transaction, which requires a 32-bit value. The data structure of transactions already accommodates custom size signatures through the Proofs field.

Network addresses

Derived identities

An address on LTO is generated with the secure hash of the public key. For the address of a derived identity, HMAC is used instead of a regular sha256 to calculate the secure hash. The HMAC secret is a generated nonce.

```
sha256_hmac(blake2b256(ED25519_public_key), nonce)
```

For certificate smart accounts, derived addresses are calculated as

```
sha256_hmac(blake2b256(sha256(RSA_public_key)), nonce)
```

It's recommended to use a random 32-byte value as nonce. If the nonce is sequential, it's trivial to generate a list of derived identities for an account. This might be a privacy concern.

Derived identities can't be used to sign transactions on the LTO Network public blockchain.

Proving an address belongs to you, for verifiable credentials, is similar to the normal procedure. You sign a message using your private key. The nonce is provided together with the public key, so the address can be calculated.

The DID url must include the nonce as query parameter.

Network addresses

```
did:lto:{address}?nonce={nonce}
```

Recursive nonce

The method to calculate the secure hash for a derived identity may be used recursively. For federated identities, the HMAC secret is generated using an identity id, plus the nonce.

```
sha256_hmac(blake2b256(public_key), sha256_hmac(id, nonce))
```

There are no restrictions on the format of the user id. It can be a sequential id, a UUID or any string. To calculate the address, the public key, the user id, and the nonce must be provided.

The DID url must include the user id and the nonce as query parameters.

```
did:lto:{address}?id={id}&nonce={nonce}
```

Network addresses

URI to address

It's possible to create an LTO address for any URI. There is no public key associated with this address. It can only act as an endpoint. The secure hash for calculating the address is simply the sha256 hash

```
sha256(uri)
```

Anybody can make a claim about this URI through associations. Depending on your use case, configure the LTO identity node to only index relevant associations.

For public DIDs, versioning can be done through a query parameter.

For private data exchange, we use HMAC with the version as secret. The version should be random and at least 32 bytes.

```
sha256_hmac(uri, version)
```

A query parameter for versioning could be used instead. But this would be less secure, because it's prone to length extension attacks.

Decentralized Identifiers

Subject

The DID of a subject on LTO Network is simply the address on LTO Network. This may be the public address of the holder or a derived address.

```
did:lto:{address}
```

Credentials

The DID of a verifiable claim relative to the subject address.

```
did:lto:{address}/credentials/{base58:cid}
```

The cid is a sha256 hash of the (unsigned) verifiable credentials. The DID contains a base58 encoded value of this id.

Decentralized Identifiers

Certificates

X.509 certificates have a DID on LTO network, which is relative to the address of the smart account.

```
did:lto:{address}/certs/{base58:fingerprint}
```

The fingerprint is the sha256 hash of the certificate in DER format. The DID contains the base58 encoded value of the fingerprint.

Cross-chain

The DID of an external blockchain can be indexed and used on LTO Network. The DID must be generated from the public key.

A cross-chain DID is equivalent to the DID on the external blockchain. The type is prefixed with lto: to specify the network

```
did:lto:ethr:0xf3beac30c498d9e26865f34fcaa57dbb935b0d74
```

When querying the address on an LTO identity node, the lto: prefix may be omitted.

Associations

Endorsing certificates

Any account can make a claim about a published X.509 certificate through associations.

Association type	Hash	Meaning
0x20	Certificate fingerprint	Endorse certificate
0x21	Certificate fingerprint	Dispute certificate

The association recipient is the smart account address, calculated from the public key.

Multiple certificates can be published for a single account. Therefore it's required to supply the certificate fingerprint as the association hash. The fingerprint is the sha256 hash of the certificate in DER format.

If a single address both endorses and disputes a certificate, the dispute should take precedence.

Decentralized Identifiers

Alias

LTO allows declaring an account as an alias through associations. Alias addresses will be combined into a single DID document, with multiple signing methods.

Association type	Hash	Meaning
0x01	N/A	Alias

Creating an alias requires an alias association transaction from both accounts to each other.

Verified credentials

For verifiable credentials, the holder has a signed copy. The issuer must also create an association on the blockchain.

Association type	Hash	Meaning
0x10	cid	Verified credentials

The cid is the sha256 hash of the (unsigned) credentials.