

RFC-001: LanternSeaCoin NFC Extension

Status: Draft

Authors: LanternSeaCoin Protocol Team

Language: English

Scope: NFC token extension, offline redeem, online redeem, and ledger snapshots

Updated: 6 October 2026. Reference profile, encodings, secure-element profile, key records, local store and references added as §21 to §32; §1 to §20 are unchanged apart from three pointer lines.

Related: RFC-002 (Magnetic Stripe Voucher Extension), RFC-003 (Chip Card Extension), LanternSeaCoin Reference Paper 2.0

1. Abstract

This RFC specifies an NFC-based extension for LanternSeaCoin that supports:

1. NFC token representation;
2. Offline redeem requests;
3. Online redeem settlement;
4. Ledger snapshots for verification, synchronization, and archival use.

The design is intended to remain compatible with the existing LanternSeaCoin concepts of real blocks, `chain_time`, `chain_index`, OTP checkpoints, claim windows, and optional archive manifests.

An NFC token is a portable claim instrument. It is not, by itself, a replacement for on-chain finality. Offline redemption can create a locally recorded claim, but final settlement requires reconciliation with the authoritative ledger.

2. Motivation

NFC provides a convenient way to transfer or present a token without requiring an always-online connection. However, NFC alone does not solve:

- double spending while offline;
- conflicting claims made by different devices;
- loss or rollback of local device state;
- verification against the latest canonical chain;
- synchronization after a period without connectivity.

This RFC separates the process into two phases:

- **Offline redeem:** create and persist a locally verifiable claim request.
- **Online redeem:** reconcile and settle the request against the canonical ledger.

Snapshots provide compact verification context but do not replace consensus or finality.

3. Terminology

Term	Meaning
NFC token	A signed, portable token containing claim and verification data

Term	Meaning
Stamp	The redeemable claim object represented by the token
Offline redeem	Local creation of a redeem request without network access
Online redeem	Submission and settlement of a redeem request against the canonical ledger
chain_time	Protocol-defined logical chain time
chain_index	Protocol-defined logical block/index position
otp_checkpoint	Current checkpoint in the protocol's OTP/hash chain
Snapshot	A signed description of ledger state at a specific height
Canonical ledger	The authoritative chain and state accepted by consensus
Drift tolerance	Maximum permitted difference between locally estimated and canonical chain context

4. Goals

This RFC aims to provide:

- a compact NFC token format;
- deterministic token verification;
- offline operation with explicit limitations;
- safe online reconciliation;
- replay and duplicate-request protection;
- snapshot-based verification;
- forward compatibility through versioning;
- clear error and state handling.

5. Non-goals

This RFC does not define:

- a new consensus algorithm;
- a replacement for the existing LanternSeaCoin transaction model;
- guaranteed offline double-spend prevention using ordinary NFC tags;
- a new cryptographic primitive;
- mandatory archive storage;
- a wallet user-interface standard.

6. NFC Token Extension

6.1 Token structure

A proposed NFC token payload is:

```
LSCNFCToken {
    protocol_version
    network_id
```

```

    token_type

    token_id
    stamp_id

    issuer_address
    beneficiary_address

    amount
    issued_height
    claim_start_height
    claim_end_height

    chain_index
    otp_checkpoint

    snapshot_height
    snapshot_hash

    flags
    signature
}

```

Field definitions

Field	Description
protocol_version	Version of this RFC-compatible token format
network_id	Network identifier, preventing cross-network replay
token_type	Token classification, such as redeemable stamp
token_id	Globally unique token identifier
stamp_id	Identifier of the underlying redeemable stamp
issuer_address	Address that issued the token
beneficiary_address	Intended recipient, if applicable
amount	Amount associated with the redeemable claim
issued_height	Ledger height at issuance
claim_start_height	Earliest valid claim height
claim_end_height	Latest valid claim height
chain_index	Chain index observed when the token was issued
otp_checkpoint	OTP/hash-chain checkpoint associated with the token
snapshot_height	Height of the verification snapshot
snapshot_hash	Hash of the referenced snapshot
flags	Optional feature and policy flags
signature	Issuer or protocol-authorized signature

The exact binary encoding is implementation-defined until a serialization standard is adopted. Canonical encoding is required before signing and hashing.

6.2 NFC payload encoding

Profile v1 fixes this encoding in §22.

The NFC payload SHOULD use a compact binary representation.

A text or JSON representation MAY be used for debugging, test vectors, or interoperability tools, but production tokens SHOULD use a canonical binary encoding to reduce size and ambiguity.

The payload SHOULD include:

```
magic      = "LSCNFC"
version    = uint8
network_id  = compact identifier
payload_len = integer
payload     = canonical encoded token
signature   = canonical signature
```

The implementation MUST reject:

- unknown mandatory versions;
 - malformed lengths;
 - duplicate fields;
 - non-canonical encodings;
 - signatures over a different representation;
 - tokens for another network.
-

6.3 Token signing

The signature MUST cover all security-relevant fields, including:

- network;
- token and stamp identifiers;
- amount;
- claim window;
- chain context;
- snapshot reference;
- flags;
- protocol version.

A conceptual signing input is:

```
token_signing_bytes =
    CANONICAL_ENCODE(
        protocol_version,
        network_id,
        token_type,
        token_id,
        stamp_id,
        issuer_address,
        beneficiary_address,
        amount,
        issued_height,
        claim_start_height,
        claim_end_height,
        chain_index,
        otp_checkpoint,
        snapshot_height,
        snapshot_hash,
```

```
    flags
)
```

The signature is then calculated over token_signing_bytes.

6.4 Token verification

A receiving device SHOULD verify the following before accepting a token:

1. The payload is structurally valid.
2. The network matches the local network.
3. The protocol version is supported.
4. The signature is valid.
5. The token is within its claim window.
6. The referenced snapshot is available or otherwise verifiable.
7. The OTP checkpoint is syntactically valid.
8. The token has not already been locally redeemed.
9. The token is not marked as revoked or invalid.
10. The amount and token type are supported by the wallet.

A successful local verification does not prove that the token has not been redeemed elsewhere while offline.

7. Offline Redeem

7.1 Purpose

Offline redeem allows a device to create a claim request without network access.

The result is a **locally persisted pending request**, not necessarily a finalized ledger transaction.

The device MUST clearly distinguish:

- locally created;
 - locally stored;
 - pending synchronization;
 - accepted by the canonical ledger;
 - rejected by the canonical ledger.
-

7.2 Offline redeem request

```
RedeemRequest {
    request_id
    protocol_version
    network_id

    token_id
    stamp_id

    redeemer_address
    beneficiary_address
    amount
}
```

```

    local_chain_index
    local_chain_time
    local_otp_checkpoint

    token_snapshot_height
    token_snapshot_hash

    created_at
    device_id_hash
    nonce

    request_signature
}

```

Request identifiers

request_id MUST be unique.

A recommended construction is:

```

request_id =
    H(
        network_id ||
        token_id ||
        stamp_id ||
        redeemer_address ||
        nonce
    )

```

The request identifier MUST NOT depend only on local wall-clock time.

7.3 Offline redeem flow

1. Read NFC token.
2. Decode and validate token structure.
3. Verify token signature.
4. Check network and protocol version.
5. Check claim window.
6. Check local duplicate/replay database.
7. Compare token context with local context.
8. Create RedeemRequest.
9. Sign or authenticate the request locally.
10. Persist the request atomically.
11. Mark the request as PENDING_ONLINE.
12. Present the user with the offline status.

The device SHOULD persist the request before displaying success to the user.

If persistence fails, the device MUST NOT report that the redeem was created successfully.

7.4 Local chain context

The existing offline-client model stores the last known real block context, including:

- chain_time;
- chain_index;
- otp_checkpoint.

While offline, the device may estimate elapsed blocks using its local clock. This estimate is not authoritative.

When creating an offline redeem request, the device SHOULD record:

- the last known real chain context;
- the locally estimated chain context;
- the token's original context;
- the local timestamp;
- the uncertainty or drift estimate.

The local OTP chain MUST follow the protocol's existing checkpoint construction.

Conceptually:

```
otp_checkpoint[0] =
    H(genesis_seed || utc_timestamp[0])

otp_checkpoint[n] =
    H(
        otp_checkpoint[n-1] ||
        utc_timestamp[n] ||
        chain_index[n]
    )
```

The exact hash function and serialization MUST follow the existing protocol specification.

7.5 Offline status values

CREATED
 STORED
 PENDING_ONLINE
 SYNC_REQUIRED
 LOCALLY_REJECTED
 EXPIRED_LOCALLY

A wallet SHOULD display PENDING_ONLINE prominently when no canonical settlement has occurred.

7.6 Offline double-spend limitations

Ordinary NFC tags cannot guarantee offline double-spend prevention if:

- the same token can be copied;
- the token can be presented to multiple offline devices;
- devices cannot communicate;
- a secure hardware counter is absent;
- the issuer cannot revoke or reconcile claims.

Therefore, the protocol MUST NOT describe an offline redeem as final merely because:

- the NFC signature is valid;
- the snapshot is valid;
- the local OTP checkpoint matches;

- the local device accepted the request.

Offline anti-double-spend security may be improved through:

- secure elements;
- hardware-backed monotonic counters;
- one-time programmable NFC hardware;
- trusted execution environments;
- bounded offline credit limits;
- short claim windows;
- issuer-side reservation;
- online confirmation before high-value redemption.

These mechanisms are optional and require separate implementation profiles.

8. Online Redeem

8.1 Purpose

Online redeem submits a locally created request to the canonical ledger for validation and settlement.

The online service **MUST** treat the request as untrusted input and independently verify:

- token authenticity;
 - request authenticity;
 - claim window;
 - token ownership;
 - amount;
 - snapshot reference;
 - duplicate status;
 - current stamp state;
 - protocol rules;
 - consensus state.
-

8.2 RPC interface

A conceptual RPC is:

```
redeem_stamp(request: RedeemRequest)
  -> RedeemResponse
```

The response is:

```
RedeemResponse {
  request_id
  token_id
  stamp_id

  status
  canonical_height
  canonical_chain_index
  canonical_otp_checkpoint

  transaction_id
}
```

```
    minted_coin_id
    rejection_code
    server_signature
}
```

8.3 Online redeem flow

1. Submit the persisted RedeemRequest.
2. Verify request structure and signature.
3. Verify token and network.
4. Load canonical stamp state.
5. Check whether the stamp is already claimed.
6. Check claim window and protocol rules.
7. Compare local context with canonical context.
8. Apply drift-tolerance policy.
9. If valid, atomically mark the stamp as claimed.
10. Create or release the associated movable coin.
11. Commit the ledger transaction.
12. Return canonical settlement data.
13. Update the client's local chain context.
14. Mark the local request as SETTLED.

The stamp state transition and coin creation MUST be atomic from the ledger's perspective.

8.4 Response statuses

```
ACCEPTED
PENDING
REJECTED
ALREADY_CLAIMED
EXPIRED
INVALID_SIGNATURE
INVALID_NETWORK
INVALID_TOKEN
INVALID_REQUEST
SNAPSHOT_UNKNOWN
SNAPSHOT_INVALID
OTP_MISMATCH
CHAIN_DRIFT_TOO_LARGE
REVOKED
INSUFFICIENT_AUTHORITY
TEMPORARY_UNAVAILABLE
```

8.5 Idempotency

Online redeem MUST be idempotent by request_id.

If the same request is submitted multiple times:

- it MUST NOT mint multiple coins;
- it MUST NOT mark the same stamp claimed more than once;

- it SHOULD return the original settlement result where available.

The server SHOULD also enforce uniqueness by:

- token_id;
 - stamp_id;
 - canonical claim record;
 - transaction ID.
-

8.6 Reconciliation and drift tolerance

The client's local chain context may differ from the canonical chain.

The online service SHOULD compare:

- local chain_index versus canonical chain_index;
- local chain_time versus canonical chain_time;
- local otp_checkpoint versus canonical checkpoint;
- token snapshot versus current or accepted historical state.

If the difference is within the configured drift_tolerance, the request MAY proceed.

If the difference exceeds the allowed tolerance:

1. the request SHOULD be marked PENDING or SYNC_REQUIRED;
 2. the client SHOULD fetch canonical history;
 3. the client SHOULD re-derive its local chain context;
 4. the request SHOULD be revalidated;
 5. the client MUST NOT silently overwrite evidence of the original local request.
-

9. Snapshot Extension

9.1 Purpose

A snapshot is a compact, signed representation of ledger state at a specific height.

Snapshots may be used for:

- NFC token verification;
- wallet startup;
- offline context;
- synchronization;
- audit;
- optional archival;
- pruning support.

A snapshot does not replace the canonical chain or consensus validation.

9.2 Snapshot structure

```
LCSnapshot {  
    snapshot_version  
    network_id  
    snapshot_id  
}
```

```

    block_height
    block_hash
    chain_time
    chain_index
    otp_checkpoint

    previous_snapshot_id
    previous_snapshot_hash

    state_root
    stamp_root
    coin_root

    total_supply
    created_at

    signer_set_id
    quorum
    signature
}

```

Field definitions

Field	Description
snapshot_version	Snapshot format version
network_id	Network identifier
snapshot_id	Unique snapshot identifier
block_height	Canonical block height covered
block_hash	Hash of the canonical block
chain_time	Chain time at the snapshot height
chain_index	Chain index at the snapshot height
otp_checkpoint	OTP checkpoint at the snapshot height
previous_snapshot_id	Optional previous snapshot reference
previous_snapshot_hash	Optional previous snapshot hash
state_root	Root of the relevant ledger state
stamp_root	Root of stamp state
coin_root	Root of coin state
total_supply	Total supply represented by the snapshot
created_at	Snapshot creation timestamp
signer_set_id	Identifier of the authorized signer set
quorum	Required or achieved signing quorum
signature	Aggregate or authorized snapshot signature

9.3 Snapshot hash

The snapshot hash MUST be calculated over canonical fields.

Conceptually:

```

snapshot_hash =
    BLAKE2b-256(
        CANONICAL_ENCODE(

```

```

        snapshot_version,
        network_id,
        snapshot_id,
        block_height,
        block_hash,
        chain_time,
        chain_index,
        otp_checkpoint,
        previous_snapshot_id,
        previous_snapshot_hash,
        state_root,
        stamp_root,
        coin_root,
        total_supply,
        created_at,
        signer_set_id,
        quorum
    )
)

```

The exact field order and encoding MUST be fixed by the serialization specification.

9.4 Snapshot chaining

Snapshots SHOULD reference the previous accepted snapshot:

```
current.previous_snapshot_hash =
    previous.snapshot_hash
```

This creates a verifiable snapshot chain and helps detect:

- rollback;
- replacement;
- inconsistent histories;
- accidental forked snapshot streams.

A client SHOULD reject a snapshot chain with:

- an invalid previous hash;
 - an unsupported version;
 - a network mismatch;
 - an invalid signature;
 - a height regression;
 - a conflicting snapshot at the same height.
-

9.5 Snapshot validation

A client SHOULD validate:

1. network identifier;
2. snapshot version;
3. signature and quorum;
4. block hash;
5. monotonic height;
6. previous snapshot linkage;

7. OTP checkpoint format;
8. state roots;
9. total supply consistency;
10. compatibility with the requested token.

The client MAY use a snapshot as a verification anchor only if the trust policy permits the signer set and quorum.

9.6 Snapshot retrieval

A conceptual RPC is:

```
get_snapshot(
    network_id,
    snapshot_id | block_height
) -> LSCSnapshot
```

An implementation MAY also support:

```
get_latest_snapshot(network_id)
get_snapshot_by_hash(snapshot_hash)
get_snapshot_range(start_height, end_height)
```

Snapshot retrieval SHOULD support integrity verification before local installation.

10. Optional Archive Integration

Snapshots and archives are independent features.

An archive MAY be anchored to a real on-chain CheckpointRecord. An archive manifest can contain:

```
ArchiveManifest {
    archive_id
    covers_heights
    checkpoint_ref
    content_hash
    segment_types
    destination
}
```

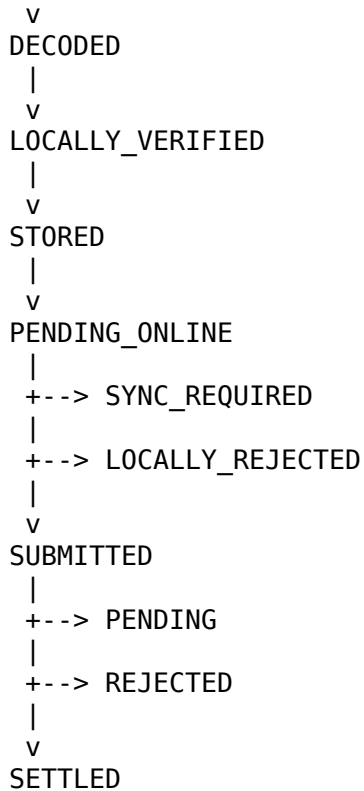
The content hash SHOULD use BLAKE2b-256, consistent with the existing archive concept.

Archives are optional and local/off-protocol unless separately standardized. A snapshot MUST NOT be treated as proof that an external archive is complete unless the archive's content hash and coverage are independently verified.

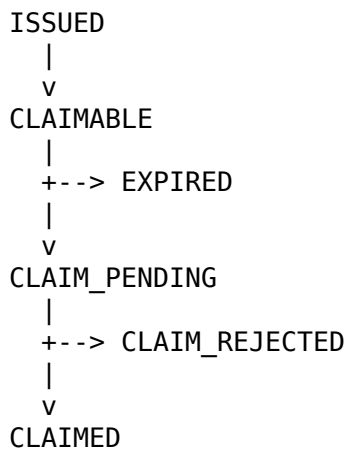
11. State Machines

11.1 Offline request state machine

NEW
|



11.2 Stamp state machine



A stamp **MUST NOT** transition from CLAIMED back to CLAIMABLE without an explicit protocol-level correction or reversal process.

12. Consensus and Governance Rules

The online redeem transaction **MUST** obey the existing consensus rules.

This RFC does not authorize a client to alter canonical state.

Any correction, reversal, or exceptional settlement **SHOULD** use the protocol's existing governance and correction mechanism. Where the existing protocol requires unanimous governance approval

for a correction, that requirement remains unchanged.

A snapshot signer **MUST NOT** independently rewrite ledger state.

13. Error Codes

Additional codes used by the reference implementation are listed in §28.

Suggested error codes:

Code	Meaning
E_VERSION	Unsupported protocol or token version
E_NETWORK	Network mismatch
E_ENCODING	Invalid or non-canonical encoding
E_SIGNATURE	Signature verification failed
E_TOKEN_UNKNOWN	Token not recognized
E_STAMP_UNKNOWN	Stamp not recognized
E_ALREADY_CLAIMED	Stamp has already been claimed
E_REQUEST_REPLAY	Request ID already used with a different payload
E_EXPIRED	Claim window expired
E_NOT_YET_VALID	Claim window has not started
E_SNAPSHOT	Snapshot invalid or unavailable
E_OTP	OTP checkpoint mismatch
E_DRIFT	Local context exceeds drift tolerance
E_REVOKED	Token or stamp revoked
E_QUORUM	Snapshot quorum invalid
E_STATE_CONFLICT	Canonical state conflict
E_TEMPORARY	Temporary service failure

14. Security Considerations

14.1 NFC cloning

NFC data can be copied unless protected by secure hardware or another anti-cloning mechanism.

The signature proves authenticity of the signed payload, not physical uniqueness of the NFC object.

14.2 Replay

Requests **MUST** be idempotent and replay-protected.

At minimum, the system **SHOULD** track:

- request_id;
- token_id;
- stamp_id;
- transaction ID;
- claim state.

14.3 Key compromise

Issuer and snapshot-signing keys require:

- secure storage;
- rotation procedures;
- revocation;
- audit;
- threshold or quorum signing where appropriate.

14.4 Clock manipulation

Local device clocks are untrusted.

A device **MUST NOT** treat its wall clock as authoritative chain time.

14.5 Snapshot rollback

Clients **SHOULD** detect snapshot rollback through:

- monotonic height checks;
- previous snapshot links;
- block-hash validation;
- signer and quorum verification.

14.6 Privacy

NFC tokens **SHOULD** avoid unnecessary personal data.

Where possible:

- use pseudonymous addresses;
- avoid embedding user-identifying metadata;
- encrypt optional private fields;
- separate public verification data from private wallet data.

14.7 Lost devices

A lost device may contain pending redeem requests.

Implementations **SHOULD** encrypt local storage and provide recovery or revocation procedures.

15. Compatibility

The extension **SHOULD** be backward-compatible with clients that do not support NFC.

A legacy client may:

- ignore NFC-specific fields;
- process the underlying stamp through ordinary online redemption;
- reject unknown token types safely.

Version negotiation is required for future incompatible changes.

Unknown optional fields **SHOULD** be ignored only when they are explicitly marked optional. Unknown mandatory fields **MUST** cause rejection.

16. Testing Requirements

Implementations SHOULD include at least the following tests.

16.1 Token tests

- valid token decoding;
- malformed payload rejection;
- signature validation;
- network mismatch;
- unsupported version;
- claim-window validation;
- token tampering;
- canonical encoding consistency.

16.2 Offline redeem tests

- successful offline request creation;
- persistence failure;
- duplicate local redemption;
- invalid token;
- expired token;
- local chain drift;
- device restart and recovery;
- corrupted local request database.

16.3 Online redeem tests

- successful settlement;
- repeated submission;
- already-claimed stamp;
- invalid request signature;
- invalid snapshot;
- excessive chain drift;
- temporary server failure;
- atomic stamp-and-coin settlement.

16.4 Snapshot tests

- valid snapshot signature;
- invalid quorum;
- previous snapshot mismatch;
- height regression;
- block-hash mismatch;
- state-root mismatch;
- total-supply mismatch;
- snapshot serialization determinism.

16.5 Adversarial tests

- NFC payload cloning;
- request replay;
- token substitution;
- cross-network replay;

- clock rollback;
 - snapshot rollback;
 - malicious local database modification;
 - concurrent online redemption attempts.
-

17. Reference Pseudocode

17.1 Offline redeem

```
function offline_redeem(token, wallet):
    decoded = decode_token(token)

    assert decoded.protocol_version is supported
    assert decoded.network_id == wallet.network_id
    assert verify_signature(decoded)
    assert claim_window_is_valid(decoded)

    if wallet.local_claim_db.contains(decoded.token_id):
        return LOCALLY_REJECTED

    context = wallet.get_last_real_chain_context()
    estimated = wallet.estimate_current_context()

    request = RedeemRequest(
        request_id = derive_request_id(decoded, wallet.address),
        protocol_version = decoded.protocol_version,
        network_id = decoded.network_id,
        token_id = decoded.token_id,
        stamp_id = decoded.stamp_id,
        redeemer_address = wallet.address,
        beneficiary_address = decoded.beneficiary_address,
        amount = decoded.amount,
        local_chain_index = estimated.chain_index,
        local_chain_time = estimated.chain_time,
        local_otp_checkpoint = estimated.otp_checkpoint,
        token_snapshot_height = decoded.snapshot_height,
        token_snapshot_hash = decoded.snapshot_hash,
        created_at = local_time(),
        device_id_hash = wallet.device_id_hash(),
        nonce = random_nonce()
    )

    request.request_signature = sign_locally(request)

    atomically:
        wallet.local_claim_db.insert(request)
        wallet.outbox.insert(request)

    return PENDING_ONLINE
```

17.2 Online redeem

```
function online_redeem(request):
    verify_request_format(request)
    verify_request_signature(request)
    verify_network(request.network_id)

    if request_id_exists_with_same_payload(request.request_id):
        return original_result(request.request_id)

    token = load_token(request.token_id)
    stamp = load_stamp(request.stamp_id)

    verify_token(token)
    verify_request_matches_token(request, token)
    verify_claim_window(stamp)
    verify_snapshot_reference(request)
    verify_otp_and_chain_context(request)

    if stamp.status == CLAIMED:
        return ALREADY_CLAIMED

    if drift_exceeds_tolerance(request):
        return PENDING

    atomically:
        mark_stamp_claimed(stamp.id, request.request_id)
        coin = create_or_release_coin(request.amount, request.beneficiary_address)
        tx = commit_redeem_transaction(request, coin)

    return ACCEPTED(tx.id, coin.id)
```

18. Open Decisions

The resolutions used by profile v1 are listed in §21.

The following decisions remain implementation or governance questions:

1. Exact binary serialization format.
 2. Signature algorithm and key format.
 3. Whether NFC tokens are bearer instruments or address-bound.
 4. Whether secure-element profiles are mandatory for high-value tokens.
 5. Maximum offline claim value.
 6. Default claim-window duration.
 7. Exact drift-tolerance value.
 8. Snapshot signer-set governance.
 9. Snapshot frequency.
 10. Whether snapshots are mandatory for NFC verification.
 11. Whether archive manifests become part of the formal protocol.
 12. Recovery behavior for lost devices.
 13. Revocation format and propagation.
 14. Fee handling for online redemption.
 15. Whether a pending request may be amended before settlement.
-

19. Recommended Rollout

Phase 1 — Experimental

- implement token encoding;
- implement signature verification;
- implement local pending requests;
- implement online idempotent redemption;
- implement basic snapshots;
- add test vectors.

Phase 2 — Controlled pilot

- enable low-value offline claims;
- add drift-tolerance handling;
- add monitoring and audit logs;
- test device restart and recovery;
- test concurrent redemption.

Phase 3 — Hardened deployment

- secure-element support;
 - key rotation and revocation;
 - quorum-signed snapshots;
 - formal serialization specification;
 - external security review;
 - operational recovery procedures.
-

20. Conclusion

This RFC defines an NFC extension for LanternSeaCoin with a clear separation between local offline claims and canonical online settlement.

The key security principle is:

An offline NFC redeem is a locally persisted claim request; only canonical online reconciliation can establish ledger finality.

Snapshots improve verification and synchronization efficiency, while optional archives support historical retention. Neither NFC signatures nor snapshots alone eliminate offline double-spend risk.

The extension can be introduced incrementally while preserving the existing LanternSeaCoin chain, OTP checkpoint, claim-window, consensus, and governance concepts.

21. Reference Profile v1

This section records how the reference implementation resolves the open decisions of §18. Profile v1 is the profile that RFC-002 and RFC-003 assume. A default MAY be changed by operator policy; a protocol-level value changes only through the governance mechanism of §12.

#	Open decision (§18)	Resolution in profile v1
1	Exact binary serialization format	Tagged TLV with LEB128 variants, strictly canonical (§22)
2	Signature algorithm and key format	Ed25519 with domain separation (§23); key is the 32-byte raw public key
3	Bearer instruments or address-bound	Address-bound by default; FLAG_BEARER (value 1) makes the presenter the recipient
4	Secure element mandatory for high value	Mandatory above require_se_above (default 50), otherwise optional (§24)
5	Maximum offline claim value	max_offline_amount 100 per token; max_offline_total 500 per device, counted in the secure element
6	Default claim-window duration	1440 blocks (one day at 60-second blocks)
7	Exact drift-tolerance value	5 blocks (five minutes of chain time)
8	Snapshot signer-set governance	The LSC node cohort signs; quorum is $\lceil 2n/3 \rceil$ (16 of 24); changes only through governance (§12)
9	Snapshot frequency	Every 60 blocks, equal to the checkpoint interval; each snapshot coincides with a checkpoint
10	Snapshots mandatory for NFC verification	Required above require_snapshot_above (default 50) or when FLAG_REQUIRE_SNAPSHOT (value 2) is set; otherwise optional
11	Archive manifests	Optional extension (§10): manifest covers height range, segment types and destination
12	Recovery for lost devices	Encrypted, authenticated local store (§26) and revocation of pending tokens (§25)
13	Revocation format and propagation	Signed records of kind TOKEN or KEY, merged monotonically and exchanged between parties (§25)
14	Fee handling for online redemption	A multi-output transfer splits payout and fee to a collector (§29)
15	Amendment of a pending request	Pending requests are immutable; replacement only through supersede, the original stays as evidence (§26)

22. Binary Encoding Profile

This section resolves §6.2 for profile v1.

22.1 Message layout

magic[6] | version[1] | network_id (varint length + bytes) | payload_len (varint) | payload

Message	Magic
NFC token	LSCNFC
Redeem request	LSCRDM
Snapshot	LSCSNP

All integers are LEB128 varints in minimal form. The payload is a sequence of fields in fixed order, each as (tag[1], value). An integer value is a varint, a bytes value is a varint length followed by the bytes. Mandatory fields use tags 1..N in ascending order without gaps. Optional fields use tags from 0x80 upward in strictly ascending order; a decoder skips unknown optional fields (§15). The trailer is the signature (bytes). A snapshot additionally carries the quorum signatures as sorted (public key, signature) pairs.

22.2 Rejection rules

A decoder MUST reject: a wrong magic, an unknown mandatory version (E_VERSION), a token of another network (E_NETWORK), truncated or surplus bytes, a length that exceeds the data, a non-minimal or oversize varint, duplicate, reordered or gapped tags, an unknown mandatory field (tag below 0x80), and a signature over a different representation. Structural faults use E_ENCODING, signature faults E_SIGNATURE.

22.3 Field tables

NFC token (LSCNFC):

Tag	Field	Type
1	protocol_version	int
2	network_id	bytes
3	token_type	bytes
4	token_id	bytes
5	stamp_id	bytes
6	issuer_address	bytes
7	beneficiary_address	bytes
8	amount	int
9	issued_height	int
10	claim_start_height	int
11	claim_end_height	int
12	chain_index	int
13	otp_checkpoint	bytes
14	snapshot_height	int
15	snapshot_hash	bytes
16	flags	int

Redeem request (LSCRDM):

Tag	Field	Type
1	request_id	bytes
2	protocol_version	int
3	network_id	bytes
4	token_id	bytes
5	stamp_id	bytes
6	redeemer_address	bytes
7	beneficiary_address	bytes
8	amount	int
9	local_chain_index	int
10	local_chain_time	int
11	local_otp_checkpoint	bytes
12	token_snapshot_height	int
13	token_snapshot_hash	bytes
14	created_at	int
15	device_id_hash	bytes
16	nonce	bytes

Snapshot (LSCSNP):

Tag	Field	Type
1	snapshot_version	int
2	network_id	bytes
3	snapshot_id	bytes
4	block_height	int
5	block_hash	bytes
6	chain_time	int
7	chain_index	int
8	otp_checkpoint	bytes
9	previous_snapshot_id	bytes
10	previous_snapshot_hash	bytes
11	state_root	bytes
12	stamp_root	bytes
13	coin_root	bytes
14	total_supply	int
15	created_at	int
16	signer_set_id	bytes
17	quorum	int

22.4 Signing input

The signature does not cover the binary bytes. It covers the canonical encoding of the fields: each field is converted to a string, prefixed with its decimal byte length and a colon, and concatenated (the netstring principle). The same signature is therefore valid for the binary and for the JSON representation. Because a decoder rejects every deviation from the single canonical byte sequence, no ambiguous representation exists, except unknown optional fields, which are not signed and have no effect.

22.5 Wire form and size

Strings in the reference implementation are byte buffers (characters 0 to 255); the wire form maps them one-to-one to bytes (Latin-1). A token in the reference test vector occupies 290 bytes in binary form against 997 bytes in JSON.

23. Signature and Domain Separation Profile

Signatures use Ed25519 [RFC8032] with the 32-byte raw public key as key format. The signed message is the domain tag followed by the canonical encoding of the fields (§22.4):

```
message = DOMAIN_TAG || canonical_encode( field_1, ..., field_n )
```

Tag	Used for
LSC-NFC-TOKEN-v1	issuer signature over a token
LSC-NFC-REQUEST-v1	signature of a redeem request
LSC-NFC-SNAPSHOT-v1	snapshot and quorum signatures
LSC-NFC-SE-v1	secure-element operations: attestation and credit reset
LSC-NFC-KEY-v1	key rotation and revocation records

The tags are pairwise prefix-free and follow the domain-separation convention of the main protocol, so that an NFC signature can never be mistaken for a block or coin-spend signature, even if the raw bytes happen to coincide.

The request identifier follows the construction recommended in §7.2:

```
request_id = BLAKE2b-256( canonical_encode( network_id, token_id, stamp_id, redeemer_address ) )
```

It does not depend on wall-clock time and MUST be unique. For a request attested by a secure element, the nonce is `se:<n>`, where `n` is the monotonic counter of the element.

24. Secure Element Profile

A secure element (SE) in profile v1 provides:

- a private key that never leaves the element; the only operation is `sign`;
- a monotonic counter that only increases and cannot be reset, which makes every offline request unique and cloned devices detectable;
- an offline credit counter: the value this device has redeemed offline since the last online settlement;
- reset of the credit counter only on a signed confirmation from the settlement service;
- one-time-write tags that are burned after redemption.

The credit reset message and the liveness attestation are:

```
reset          = "LSC-NFC-SE-v1" || canonical_encode( "credit-reset", device_id_hash, up_to_counter )
attestation    = "LSC-NFC-SE-v1" || canonical_encode( "attest", challenge, counter, device_id_hash )
```

The attestation proves to a verifier that the element is live: it signs the challenge together with the next counter value.

Policy checks. An offline redemption is checked in this order: an amount above `require_se_above` without a secure element is rejected with `E_SE_REQUIRED`; an amount above `max_offline_amount` is rejected with `E_OFFLINE_LIMIT`; a failed local store write is a failure, never a success (`E_PERSISTENCE`).

Clone detection and loss bound. Two different requests with the same counter value are a clone indicator (`E_REQUEST_REPLAY`). If an attacker creates `c` clones with identical state, each can spend the remaining credit before reconciliation:

$L_clone \leq c * (max_offline_total - u_0)$

One-time tags. After redemption the tag payload is replaced by a burned marker beginning with LSCNFC-BURNED]. A second write attempt fails and the redemption is rejected with E_TOKEN_CONSUMED.

Platforms. The interface is a contract that any platform can implement: a Java Card applet, Android Keystore with StrongBox, or the Apple Secure Enclave. The reference implementation simulates the element; “the key never leaves the element” is an API guarantee there and not a hardware property.

25. Key Records: Rotation and Revocation

An issuer key can be stolen or expire. Two signed record kinds exist; both are bound to the network so that they cannot be replayed on another network.

Record	Content	Signatures
Key rotation	old key → new key from a given height	both keys sign: the old key authorizes the successor, the new key proves possession
Revocation	kind TOKEN (one token) or kind KEY (a key) with reason RETIRED or COMPROMISED	

Semantics:

- A key reported as COMPROMISED invalidates all of its tokens; a key reported as RETIRED invalidates only tokens issued later.
- A key MAY revoke only itself and older keys of the same rotation chain, never a newer key.
- Distribution happens by merging: the result is monotonic, idempotent and independent of the order in which records arrive.
- A token whose issuer key is not (or no longer) authorized is rejected with E_ISSUER_UNKNOWN; a revoked token or key with E_REVOKED.

26. Local Claim Store

Offline redemptions exist at first only on the device. A loss of the store loses value, and a leak of the store exposes redemption data. The store therefore MUST:

- encrypt and authenticate its contents (ChaCha20 [RFC8439] and a BLAKE2b MAC [RFC7693] in the encrypt-then-MAC pattern, with separate keys for encryption and MAC); tampering, a wrong key and truncation are detected;
- write atomically through a temporary file and an atomic replace, so that a crash never leaves half a database;
- never overwrite an entry: status changes append to a history;
- allow supersede only for a request that is not yet settled, and keep the original as evidence;
- support recovery after a restart: the operation pending returns what still has to be submitted.

The file begins with the magic LSCDB1. A failure to persist is reported as E_PERSISTENCE.

27. Status Registry

The names used by the reference implementation, as an extension of §7.5, §8.4 and §11:

Group	Values
Offline request (§7.5, §11.1)	CREATED, STORED, PENDING_ONLINE, SYNC_REQUIRED, LOCALLY_REJECTED, EXPIRED_LOCALLY, SUBMITTED, SETTLED, SUPERSEDED
Online response (§8.4)	ACCEPTED, PENDING, REJECTED, ALREADY_CLAIMED, EXPIRED, INVALID_SIGNATURE, INVALID_NETWORK, INVALID_TOKEN, INVALID_REQUEST, SNAPSHOT_UNKNOWN, SNAPSHOT_INVALID, OTP_MISMATCH, CHAIN_DRIFT_TOO_LARGE, REVOKED, INSUFFICIENT_AUTHORITY, TEMPORARY_UNAVAILABLE
Stamp (§11.2)	ISSUED, CLAIMABLE, EXPIRED, CLAIM_PENDING, CLAIM_REJECTED, CLAIMED

SUPERSEDED marks a pending request that was replaced; the original remains stored.

28. Additional Error Codes

In addition to §13, the reference implementation uses:

Code	Meaning
E_PERSISTENCE	Local storage failed; the redemption MUST NOT be reported as successful
E_SE_REQUIRED	The amount requires a secure element
E_OFFLINE_LIMIT	The offline limit was exceeded
E_ISSUER_UNKNOWN	The issuer key is not authorized (any more)
E_TOKEN_CONSUMED	The one-time tag was already burned
E_REQUEST_MISMATCH	The request does not match the issued token

29. Ledger Integration

Online redemption uses the existing ledger objects; it adds no ledger of its own. It executes a claim of the stamp and a transfer to the beneficiary address atomically: if any step fails, a rollback restores the ledger exactly. Only the holder of the stamp can claim. The beneficiary is the address named in the token; with FLAG_BEARER the presenter receives the payout. A fee is paid through a second output of the multi-output transfer to a collector, so the amounts of inputs and outputs balance. The extension does not change the block format and does not affect consensus (§12).

30. Conformance Notes

The reference implementation exists in two styles: pure functions on a plain state, and classes. Both build on one shared module for the canonical encoding, constants and error codes, and MUST produce byte-identical encodings and signatures. The reference test suite runs 113 checks through both styles and across them. It covers token round trips and every rejection rule of §22.2, offline redemption with policy, replay and clone detection, one-time tag burning, rollback, restart and tampering of the encrypted store, key rotation and revocation merging, and snapshot chaining and quorum.

The pseudocode of §17 remains the reference for control flow; this part fixes encodings and default values.

31. Related Specifications

- RFC-002, Magnetic Stripe Voucher Extension, and RFC-003, Chip Card Extension, reuse the redemption model of §7 to §9 and the profile of this part.
 - LanternSeaCoin Reference Paper 2.0 defines the common context: stamps, chain time, the OTP chain, domain separation and encoding, multi-output transfers, checkpoints, the redemption protocol and the banknote-inspired layers.
-

32. References

Titles and numbers are given as known and have not been re-fetched for this revision.

32.1 Normative

- [RFC2119] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, BCP 14, RFC 2119, 1997.
- [RFC8032] S. Josefsson, I. Liusvaara, *Edwards-Curve Digital Signature Algorithm (EdDSA)*, RFC 8032, 2017.
- [RFC7693] M-J. Saarinen, J-P. Aumasson, *The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC)*, RFC 7693, 2015.
- [RFC8439] Y. Nir, A. Langley, *ChaCha20 and Poly1305 for IETF Protocols*, RFC 8439, 2018.
- [LEB128] *DWARF Debugging Information Format*, variable length data (unsigned LEB128).
- [NETSTRINGS] D. J. Bernstein, *Netstrings*, 1997.
- [LSC-PAPER] *LanternSeaCoin Reference Paper 2.0*.

32.2 Informative

- [ISO14443] ISO/IEC 14443, *Identification cards: contactless integrated circuit cards, proximity cards*.
- [ISO7816] ISO/IEC 7816-4, *Identification cards: integrated circuit cards, organization, security and commands for interchange*.
- [NDEF] NFC Forum, *NFC Data Exchange Format (NDEF) Technical Specification*.
- [JAVACARD] Oracle, *Java Card Platform Specification*.
- [ANDROID-SE] Android Developers, *Android Keystore system and StrongBox*.
- [APPLE-SE] Apple, *Apple Platform Security: Secure Enclave*.
- [PCSC] PC/SC Workgroup, *PC/SC specifications* for smart card readers.
- [CC] ISO/IEC 15408, *Common Criteria for Information Technology Security Evaluation*.

- [CFN88] D. Chaum, A. Fiat, M. Naor, *Untraceable Electronic Cash*, CRYPTO 1988.