SecureTransfer Protocol Specification v1.0

1. Overview

SecureTransfer is a binary protocol for secure, peer-to-peer file transfers over local networks. It provides end-to-end encryption, device authentication, and integrity verification.

2. Message Format

All messages follow this structure:

- **Length**: Big-endian uint32, total message size excluding length field
- **Type**: Message type identifier (see below)
- **Version**: Protocol version (currently 0x01)
- **Payload**: Variable-length binary data, format depends on type

3. Message Types

3.1 HELLO (0x01)

Purpose: Initial connection, version negotiation

Payload:

```json
{
  "protocol_version": "1.0",
  "device_id": "uuid-v4-string",
  "device_name": "human-readable-name",
  "capabilities": ["chunked_transfer", "resume", "compression"],
  "timestamp": 1234567890
}
```

Response: HELLO_ACK (0x02) or ERROR (0xFF)

3.2 AUTH (0x03)

Purpose: Device authentication using challenge-response

Payload:

```json
{
  "device_id": "uuid-v4-string",
  "challenge_response": "base64-encoded-signature",
  "public_key_hash": "sha256-hash-of-public-key"
}
```

Response: AUTH_SUCCESS (0x04) or AUTH_FAILED (0x05)

3.3 FILE_META (0x10)

Purpose: Initiate file transfer

Payload:

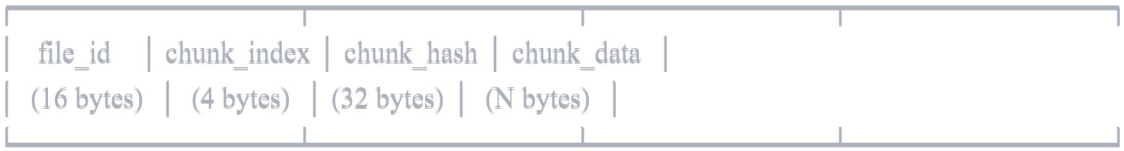
```json
{
  "file_id": "uuid-v4-string",
  "filename": "example.pdf",
  "size": 1048576,
  "hash": "sha256-of-entire-file",
  "chunk_size": 65536,
  "total_chunks": 16,
  "mime_type": "application/pdf",
  "timestamp": 1234567890
}
```

Response: FILE_READY (0x11) or FILE_REJECT (0x12)

3.4 FILE_CHUNK (0x20)

Purpose: Transfer individual file chunk

Payload:



- **file_id**: UUID binary representation
- **chunk_index**: Big-endian uint32
- **chunk_hash**: SHA-256 of chunk_data
- **chunk_data**: Encrypted binary data

Response: CHUNK_ACK (0x21) or CHUNK_RETRY (0x22)

3.5 FILE_END (0x30)

Purpose: Signal transfer completion

Payload:

```
json

{
  "file_id": "uuid-v4-string",
  "final_hash": "sha256-of-entire-file",
  "chunks_sent": 16
}
```

Response: FILE_COMPLETE (0x31) or FILE_CORRUPT (0x32)

3.6 VERIFY (0x40)

Purpose: Request integrity verification

Payload:

```
json

{
  "file_id": "uuid-v4-string",
  "verification_hash": "sha256-hash"
}
```

Response: VERIFY_OK (0x41) or VERIFY_FAIL (0x42)

3.7 ERROR (0xFF)

Purpose: General error message

Payload:

```json
{
  "error_code": 1001,
  "error_message": "Authentication failed",
  "details": "Public key not found in trusted store"
}
```

4. State Machine

```
DISCONNECTED
    |
    ├─(mDNS discovery)──> DISCOVERED
    |
DISCOVERED
    |
    ├─(TCP connect + HELLO)──> CONNECTED
    |
CONNECTED
    |
    ├─(AUTH success)──> AUTHENTICATED
    ├─(AUTH failure)──> DISCONNECTED
    |
AUTHENTICATED
    |
    ├─(FILE_META)──> TRANSFERRING
    |
TRANSFERRING
    |
    ├─(FILE_CHUNK × N)──> TRANSFERRING
    ├─(FILE_END + VERIFY OK)──> COMPLETE
    ├─(Error)──> FAILED
    |
COMPLETE or FAILED
    |
    └─(Close connection)──> DISCONNECTED
```

5. Encryption

5.1 Key Exchange

1. **Initial Pairing**: ECDH (Curve25519) key exchange via QR code
2. **Shared Secret Derivation**: `HKDF-SHA256(ecdh_shared_secret, salt=timestamp, info="securetransfer")`

5.2 Message Encryption

- **Algorithm**: AES-256-GCM
- **Key**: Derived from shared secret using HKDF
- **Nonce**: 12 random bytes, prepended to ciphertext
- **Format**: `[12-byte nonce][ciphertext][16-byte auth tag]`

5.3 Integrity

- Per-chunk: SHA-256 in FILE_CHUNK message
- Full file: SHA-256 in FILE_META and FILE_END for comparison

6. Error Handling

6.1 Connection Errors

- **Timeout**: 30 seconds of inactivity
- **Recovery**: Automatic reconnection (max 3 attempts with exponential backoff)

6.2 Transfer Errors

- **Chunk Hash Mismatch**: Send CHUNK_RETRY, max 5 retries per chunk
- **Connection Loss**: Save state (last successful chunk index), resume on reconnect
- **Disk Full**: Send ERROR, abort transfer, cleanup partial file

6.3 Authentication Errors

- **Invalid Signature**: Send AUTH_FAILED, close connection
- **Unknown Device**: Send AUTH_FAILED with error code
- **Replay Attack**: Reject if timestamp outside ±2 minute window

7. Versioning

7.1 Compatibility

- Clients MUST reject connections with incompatible major versions
- Clients SHOULD negotiate lowest common minor version
- Unknown message types SHOULD be ignored with warning log

7.2 Future Extensions

Reserved message type ranges:

- 0x50-0x5F: Compression
- 0x60-0x6F: Multi-file transfers
- 0x70-0x7F: Folder sync

8. Performance Considerations

- **Chunk Size**: 64KB (configurable in FILE_META)
- **Pipelining**: Up to 4 chunks in-flight for better throughput
- **Flow Control**: Receiver sends ACK for every 4 chunks
- **Buffering**: Sender maintains 256KB send buffer, receiver 256KB receive buffer

9. Security Properties

9.1 Guarantees

- **Confidentiality**: AES-256-GCM encryption
- **Authenticity**: ECDH key exchange + challenge-response
- **Integrity**: SHA-256 hashing at chunk and file level
- **Forward Secrecy**: New session keys per connection

9.2 Limitations

- No protection against traffic analysis (file size, timing)
- Vulnerable to DoS if attacker controls local network
- Assumes secure initial pairing (QR code must be scanned in person)

10. Implementation Notes

10.1 Serialization

Recommended: Protocol Buffers or MessagePack for payload encoding

- Efficient binary format
- Schema evolution support
- Available in most languages

10.2 Testing

- **Unit Tests**: Message serialization/deserialization
- **Integration Tests**: Full connection lifecycle
- **Fuzzing**: Random invalid messages
- **Chaos Tests**: Random disconnections, corrupted data

10.3 Logging

Each message SHOULD log:

- Timestamp
- Message type
- Source/destination device_id
- Success/failure status
- Error details (if applicable)