

Flutter Secure QR Implementation Playbook v1

This document is an execution-focused engineering guide intended for developers building a Flutter mobile application that scans, decodes, verifies, and generates Secure Offline QR codes. It complements RFC v1.1 and translates the specification into concrete Flutter implementation steps. The design is explicitly aligned with UIDAI Secure QR behavior observed in Android logcat analysis.

1. Objective for the Development Agent

The agent **MUST** be able to:

- Build a Flutter app from scratch
- Scan UIDAI Aadhaar Secure QR (as much as legally possible)
- Decode binary payloads using 0xFF separators
- Verify RSA signatures offline
- Render demographic data and embedded photo
- Generate test Secure QR codes within the app for sharing and re-scanning

2. Recommended Folder Structure

```
lib/  
  ■■ app/  
    ■ ■■ app.dart  
    ■ ■■ routes.dart  
  ■■ scanner/  
    ■ ■■ qr_scanner_screen.dart  
  ■■ decoder/  
    ■ ■■ base64_decoder.dart  
    ■ ■■ block_splitter.dart  
    ■ ■■ uidai_field_mapper.dart  
  ■■ crypto/  
    ■ ■■ rsa_verifier.dart  
  ■■ generator/  
    ■ ■■ secure_qr_generator.dart  
  ■■ models/  
    ■ ■■ secure_person_model.dart  
  ■■ ui/  
    ■■ result_screen.dart
```

3. Mandatory Packages

Dependencies to be used:

- mobile_scanner – camera & QR scanning
- pointycastle – RSA verification
- convert – Base64 utilities
- qr_flutter – QR generation

The agent **MUST NOT** introduce network, analytics, or cloud dependencies.

4. Scanning Flow (UIDAI-Compatible)

1. Initialize camera using mobile_scanner
2. Capture raw QR string (Base64)
3. Reject non-Base64 payloads
4. Pass raw string into decoding pipeline

No API calls must be made at any stage.

5. Decoding Pipeline (Exact Algorithm)

Input: Base64 string

Step 1: Base64Decode → Uint8List

Step 2: Iterate bytes sequentially

Step 3: Split blocks on byte value 0xFF (255)

Step 4: Preserve empty blocks (reserved fields)

Step 5: Identify last block as RSA signature

Step 6: Identify JPEG photo block by binary markers

This logic mirrors UIDAI Secure QR parsing.

6. UIDAI Field Mapping Reference

Block index mapping (derived from study):

- [0] Version
- [1] Internal reference ID
- [2] Name
- [3] DOB
- [4] Gender
- [5] Reserved
- [6] City
- [7–9] Address lines
- [10] PIN
- [12] State
- [14] City (repeat)
- [15] Area/Sub-district
- [17] JPEG photo
- [18] RSA signature

7. Signature Verification

Algorithm:

- RSA-2048
- SHA-256 digest
- PKCS#1 v1.5 padding

Steps:

1. Extract payload bytes (before final 0xFF)
2. Extract signature bytes (last block)
3. Verify using bundled public key
4. If verification fails → mark QR as UNVERIFIED

8. UI Rendering Rules

- Display demographic data only after decoding
- Photo must be rendered using Image.memory
- Show verification badge (Verified / Unverified)
- Never show Aadhaar number (not present)
- Never auto-export or log decoded data

9. Secure QR Generation (For Testing)

Generator flow:

1. Construct logical data model
2. Encode fields in UTF-8
3. Join fields with 0xFF separators
4. Append JPEG photo bytes
5. Hash payload using SHA-256
6. Sign hash using RSA private key (test key only)
7. Append signature
8. Base64 encode final byte array
9. Render QR using qr_flutter

Generated QR MUST be scannable by the same app.

10. Testing Checklist for Agent

- Scan generated QR → data matches
- Scan UIDAI Aadhaar QR → partial decode succeeds
- Modify 1 byte → verification fails
- Remove separator → parsing fails
- Oversized QR → rejected gracefully

Conclusion

This playbook is intended to be handed directly to a Flutter developer. If followed strictly, the resulting application will replicate UIDAI Secure QR scanner behavior to the maximum legally permissible extent while also supporting custom Secure QR generation for internal testing.