**BORAYS Crypto Wallet**  
*Version 1.0*

**Project Overview**

BORAYS Crypto Wallet is a Flutter-based cryptocurrency wallet designed with dual-device security in mind. It implements two-party ECDSA signing and Paillier-based encryption to ensure that private keys are split across two trusted devices. The app interacts with Ethereum-compatible networks (via Web3) and uses Firebase/Cloud Firestore for real-time synchronization between paired devices. Users can:

* Create a wallet by generating a single 24-word mnemonic,
* Split that mnemonic into two 12-word halves (one half stored on each device),
* Set up individual login and transaction passwords on both devices,
* Initiate a transaction on Device 1 and approve/complete it on Device 2, and
* View balances, transaction history, and use QR-code-based utilities for address sharing or transaction payloads.

**Features**

* **Dual-Device Key Management**
  + 24-word BIP39 mnemonic split into two 12-word parts; each device holds one half.
* **Two-Party ECDSA Signing (TSS)**
  + Implements ECDSA Threshold Signature Scheme so that both devices must cooperate to produce a valid signature without exposing the full private key.
* **Paillier Encryption Support**
  + Uses the Paillier cryptosystem for secure partial signing and to protect sensitive ephemeral data during the two-party protocol.
* **Web3 Integration**
  + Leverages web3dart to query balances and submit transactions to an Ethereum (or EVM-compatible) node.
* **Firebase/Firestore Sync**
  + Real-time cloud storage for pending transactions, device pairing, and state synchronization.
* **Secure Local Storage**
  + Sensitive data (passwords, partial key material, metadata) is stored locally via flutter\_secure\_storage.
* **QR Code Generator & Scanner**
  + For quickly sharing transaction data or wallet addresses between devices or with other users.
* **Dynamic Theming**
  + Light/dark mode toggle powered by a Provider-style state management.
* **Cross-Platform Support (Web & Desktop)**
  + Builds for Chrome (web), Edge (web), and Windows desktop work out of the box.

**Note:** As of this version, the project is configured only for **Web (Chrome/Edge)** and **Windows (desktop)** targets. Android support is not yet configured—please see “Running on Android” below for necessary steps.

**Technologies & Dependencies**

* **Flutter SDK** (≥ 2.19.0 < 3.0.0)
* **Dart** (2.19.x)
* **Provider**: State management
* **web3dart**: Ethereum RPC client
* **bip39**: Mnemonic generation/validation
* **flutter\_secure\_storage**: Encrypted local storage
* **local\_auth**: Biometric/device‐level authentication (PIN/biometric enforcement)
* **firebase\_core & cloud\_firestore**: Real-time database for transaction sync
* **http**: For ancillary API calls (e.g., gas estimates)
* **uuid**: Unique identifiers for Firestore documents
* **qr\_flutter**: QR code generation
* **mobile\_scanner**: QR code scanning
* **Paillier & ECDSA TSS Modules**: Custom implementations under lib/crypto/
* **flutter\_svg**: Vector asset rendering

All dependencies are declared in pubspec.yaml.

**Prerequisites**

1. **Flutter SDK** installed (version ≥ 2.19.0 < 3.0.0).
2. **Dart SDK** (bundled with Flutter).
3. **An Ethereum-compatible node or RPC endpoint** (e.g., Infura, Alchemy, or a self-hosted node).
4. **Firebase Project** with Cloud Firestore enabled.
5. **Web & Desktop Toolchains** (no extra installation required beyond Flutter).
6. **(Optional) Android Toolchain**: to build for Android, you must install Android Studio and configure the Android SDK.

**Installation & Setup**

**1. Clone This Repository**

git clone <repository-URL>

cd borays\_crypto\_wallet

**2. Install Flutter Dependencies**

flutter pub get

**3. Firebase Configuration**

1. Create a Firebase project at [console.firebase.google.com].
2. Enable Firestore in **Test Mode** (or set appropriate security rules).
3. Under **Project Settings → Your Apps**, add a new iOS app (even if you don’t build for iOS right now) and/or Android app:
   * Download **GoogleService-Info.plist** and place it in ios/Runner/.
   * Download **google-services.json** and place it in android/app/.
4. For Web, register a new web app in Firebase, then copy the configuration snippet into web/index.html within the <head> tag:
5. <script src="https://www.gstatic.com/firebasejs/9.xx.x/firebase-app.js"></script>
6. <script src="https://www.gstatic.com/firebasejs/9.xx.x/firebase-firestore.js"></script>
7. <script>
8. // Your Firebase SDK configuration
9. const firebaseConfig = {
10. apiKey: "YOUR\_API\_KEY",
11. authDomain: "YOUR\_AUTH\_DOMAIN",
12. projectId: "YOUR\_PROJECT\_ID",
13. storageBucket: "YOUR\_STORAGE\_BUCKET",
14. messagingSenderId: "YOUR\_MSG\_SENDER\_ID",
15. appId: "YOUR\_APP\_ID"
16. };
17. firebase.initializeApp(firebaseConfig);
18. </script>

**Important:** If you do not place these files in the correct folders before running, the application will fail at startup with a Firebase configuration error.

**Supported Platforms**

* **Web**
  + Chrome (web)
  + Edge (web)
  + (Any modern web browser is likely supported, but Chrome/Edge are tested targets.)
* **Desktop (Windows)**
  + You can build and run on Windows without additional setup once Flutter is installed.

**Not Yet Supported:** Android (and iOS) are not configured by default in this version. Follow the “Running on Android” section below if you wish to enable Android support manually.

**Running on Web & Desktop**

1. **Web (Chrome or Edge)**
2. flutter run -d chrome

or

flutter run -d edge

The app will compile and launch in a browser window.

1. **Desktop (Windows)**
2. flutter run -d windows

A native Windows desktop window will appear.

**Supported Devices Displayed in Your IDE**

When you open this project in VS Code or Android Studio (with the Flutter plugin enabled), you should see “Chrome (web)”, “Edge (web)”, and “Windows (desktop)” as valid device targets.

**Running on Android (Manual Setup)**

**Warning:** The Android target is **not pre-configured**. If you attempt flutter run -d android right now, you may encounter errors. To enable Android support, follow these steps:

1. **Install Android Studio**
   * Download and install Android Studio (<https://developer.android.com/studio>).
   * During installation, ensure you install the **Android SDK**, **Android SDK Platform-Tools**, and **Android SDK Build-Tools**.
2. **Configure Your ANDROID\_HOME (or ANDROID\_SDK\_ROOT)**
   * After installing Android Studio, open it once and allow it to download required SDK components.
   * On Windows, ensure your environment variables are set (e.g., ANDROID\_HOME=C:\Users\<YourUser>\AppData\Local\Android\Sdk).
3. **Enable an Android Virtual Device (AVD) or Connect a Physical Device**
   * Open Android Studio → **AVD Manager** → create a Pixel (or other) virtual device running Android 11.0 (API 30+) or higher.
   * Alternatively, connect a physical Android device via USB with “USB Debugging” enabled.
4. **Add Android Platform Files (If Missing)**
   * If you do not see an android/ folder in your project, run:
   * flutter create .

This regenerates platform folders—be careful if you have custom native code; back it up first.

1. **Update android/app/build.gradle**
   * Under defaultConfig { ... }, ensure minSdkVersion is at least 23.
   * In android/gradle.properties, you can add:
   * android.enableAapt2=true
2. **Place google-services.json in android/app/**
   * If you followed “Firebase Configuration” above and placed google-services.json in android/app/, confirm it’s recognized by the Android build.
3. **Sync & Rebuild**
   * In Android Studio:
     + Open the project → “Flutter packages get” → build → run.
   * Or from the command line:
   * flutter run -d android

If you see any errors related to missing SDK components or Gradle issues, follow the console instructions to install or update those components. Once properly configured, the app will run on your Android emulator or physical device.

**Folder Structure**

borays\_crypto\_wallet/

├── android/ # Android platform configurations (if you ran flutter create)

├── ios/ # iOS platform configurations (not used in this version)

├── lib/ # Main Dart source code

│ ├── components/ # Reusable UI components

│ │ ├── nft\_balances.dart # Displays NFT data (if integrated)

│ │ └── send\_tokens.dart # Form to initiate/send transactions

│ ├── crypto/ # Cryptographic protocols

│ │ ├── ecdsa\_tss.dart # Two-party ECDSA TSS implementation

│ │ ├── paillier.dart # Paillier encryption primitives

│ │ └── paillier\_crypto.dart # Helper functions for Paillier operations

│ ├── models/ # Data models

│ │ ├── wallet\_data.dart # Holds public data (addresses, etc.)

│ │ └── transaction.dart # Transaction metadata (nonce, gas, etc.)

│ ├── pages/ # Screens / pages

│ │ ├── landing\_page.dart # Initial screen (Create/Import choice)

│ │ ├── create\_or\_import.dart # Wallet creation/import flow

│ │ ├── login\_page.dart # Device login screen (12-word + PIN)

│ │ ├── device\_specific\_verify\_page.dart # Device-pairing verification

│ │ └── home\_page.dart # Main wallet dashboard

│ ├── providers/ # State management (Provider)

│ │ ├── wallet\_provider.dart # Wallet logic, key splitting, sync

│ │ └── theme\_provider.dart # Light/Dark theme toggling

│ ├── routes.dart # Named routes for Navigator

│ ├── secure\_storage.dart # Abstraction over `flutter\_secure\_storage`

│ ├── wallet\_api.dart # Web3 RPC wrappers (eth\_getBalance, sendRawTransaction)

│ └── widgets/ # Smaller UI elements

│ ├── faucet\_dialog.dart # Requests test ETH from a faucet (for dev/testnets)

│ ├── info\_popup.dart # Generic popup dialog for alerts

│ ├── qr\_generator.dart # Generates QR code from address or tx data

│ ├── qr\_scanner.dart # Scans QR codes for address/tx info

│ └── transaction\_info.dart # Displays a single transaction’s details

├── services/ # Business logic / platform-specific services

│ ├── firestore\_service.dart # Firestore CRUD operations (transactions, pair requests)

│ ├── wallet\_initialization\_service.dart # Orchestrates initial wallet setup

│ ├── debug\_wallet\_service.dart # Local debug/mock implementations (bypassing Firebase)

│ └── cloud\_wallet\_service.dart # Production Firebase service wrapper

├── test/ # Unit & widget tests (basic coverage)

├── web/ # Web entrypoint & Firebase initialization

├── windows/ # Windows-specific configs (CMakeLists, runner code)

├── analysis\_options.yaml # Dart analyzer rules

├── pubspec.yaml # Project metadata & dependencies

└── pubspec.lock # Locked dependency versions

**Usage Flow**

1. **Landing Page**
   * Choose to **Create Wallet** or **Import Wallet**.
2. **Create Wallet**
   * Generates a 24-word BIP39 mnemonic.
   * Prompt user to enter the first 12 words on **Device 1** → set login/transaction PIN on Device 1.
   * Prompt user to enter the last 12 words on **Device 2** → set login/transaction PIN on Device 2.
   * Once both devices complete setup, a pairing document is written to Firestore to link Device 1 & Device 2.
3. **Import Wallet**
   * On either device: enter the corresponding 12-word half + login PIN → if paired data exists in Firestore → finish import and navigate to **Home Page**.
4. **Home Page (Dashboard)**
   * View combined balance (read by querying both public keys).
   * Browse transaction history (read from Firestore + on-chain).
   * **Send Tokens**:
     + On Device 1: fill out “Send Tokens” form (recipient address, amount, gas settings).
     + App constructs an unsigned ECDSA transaction.
     + Pauses local signing, pushes a “pending transaction” document to Firestore.
     + Device 2’s listener sees pending transaction → prompts user to Approve → Device 2 uses its half of the key (Paillier + TSS steps) to finish the signature.
     + Fully signed raw transaction is broadcast via wallet\_api.dart.
     + Both devices update UI once Tx hash is confirmed.
5. **QR Code Features**
   * **QR Generator**: Display your wallet address or transaction payload, share with another device or user.
   * **QR Scanner**: Scan a recipient’s QR code to auto-fill the address field.
6. **Theme Toggle**
   * Access via the top-right corner → switch between light/dark. State is persisted in ThemeProvider and local storage.

**Firebase Firestore Schema**

* **Collection**: wallet\_sessions
  + Document ID: <UUID>
  + Fields:
    - device1\_pubkey (hex)
    - device2\_pubkey (hex)
    - paired (bool)
    - createdAt (Timestamp)
* **Collection**: pending\_transactions
  + Document ID: <UUID>
  + Fields:
    - fromPubKey (hex)
    - toAddress (hex)
    - amountWei (string)
    - gasPrice (string)
    - gasLimit (string)
    - nonce (int)
    - partialSig1 (string) – first device’s partial signature blob (Paillier encrypted)
    - isApproved (bool)
    - createdAt (Timestamp)
* **Collection**: completed\_transactions
  + Document ID: <TxHash>
  + Fields:
    - rawTx (hex)
    - txHash (hex)
    - status (string)
    - timestamp (Timestamp)

The firestore\_service.dart class centralizes create/read/update operations for these collections.

**Configuration Details**

1. **RPC Endpoint**
   * In lib/wallet\_api.dart, locate:
   * static const String rpcUrl = 'https://mainnet.infura.io/v3/YOUR\_INFURA\_PROJECT\_ID';
   * static const String chainId = '1'; // Mainnet = 1; Ropsten = 3; etc.
   * Replace with your own RPC (Infura, Alchemy, QuickNode, or a self-hosted Ethereum node).
   * If you plan to use a testnet (Ropsten, Goerli, etc.), change chainId accordingly and ensure your RPC URL points to that network.
2. **Biometric & PIN Settings**
   * In lib/secure\_storage.dart, look for keys such as LOGIN\_PIN\_KEY, TX\_PIN\_KEY.
   * These are used by the local\_auth plugin to lock/unlock the wallet. You can disable biometrics by editing authenticate() in login\_page.dart.
3. **Testing on Local Firebase Emulator (Optional)**
   * If you prefer not to use production Firestore, spin up the Firebase Emulator Suite:
   * firebase emulators:start --only firestore
   * Update firebase\_options.dart (if using flutterfire\_cli) or manually point to localhost:8080 in firestore\_service.dart.

**Running Tests**

Basic unit tests are located under the test/ directory. To run all tests:

flutter test

Ensure your emulators or mock services are configured if any test relies on Firestore or Web3 calls.

**Contributing**

1. Fork this repository.
2. Create a feature branch:
3. git checkout -b feature/YourFeatureName
4. Commit your changes with a clear, descriptive message.
5. Push to your fork:
6. git push origin feature/YourFeatureName
7. Open a Pull Request describing your improvements or bug fixes.

Please follow the existing code style and add corresponding tests for any new functionality.

**Known Limitations & TODOs**

* **Android & iOS Support**
  + Currently, **only Web and Windows Desktop** targets are configured. Android support must be enabled manually (see “Running on Android”). iOS is not implemented.
* **Gas Estimation**
  + Users enter gas parameters manually. Future improvement: automatic gas estimation via eth\_estimateGas.
* **Testnet Faucet Integration**
  + FaucetDialog is stubbed out; it needs to call a real testnet faucet for Ropsten/Goerli.
* **Offline/No-Internet Handling**
  + Improve error handling/UI when connectivity is lost.
* **Transaction History Caching**
  + Currently, completed transactions are only stored in Firestore; consider a local SQLite cache for faster access.
* **Multi-Account Support**
  + Currently supports a single wallet split across two devices. Expand to multiple wallets/accounts in future versions.

**Troubleshooting**

* **“Firebase Permission Denied” Errors**
  + Verify Firestore rules under **Firestore → Rules**. For development, you may use:
  + rules\_version = '2';
  + service cloud.firestore {
  + match /databases/{database}/documents {
  + match /{document=\*\*} {
  + allow read, write: if true;
  + }
  + }
  + }
  + Be sure to tighten these rules before deploying to production.
* **Missing GoogleService-Info.plist / google-services.json**
  + The app will crash if these files are not present in ios/Runner/ or android/app/. Confirm they exist and have correct contents.
* **Web Build Fails with “Firebase method not found”**
  + Ensure you added the <script> tags in web/index.html with the correct Firebase SDK versions. Double-check the version numbers and that they match your pubspec.yaml dependency versions.
* **Biometric Authentication Not Triggering**
  + On iOS:
    - Make sure you have enabled Face ID/Touch ID under “Signing & Capabilities” → “Keychain Sharing” in the Xcode project.
  + On Android:
    - Confirm minSdkVersion ≥ 23 in android/app/build.gradle.
    - Make sure the device/emulator has a secure lock screen set up (PIN/Password).
* **Android Build Fails or “No connected devices”**
  + Install Android Studio → open **AVD Manager** → configure an emulator, or connect a real device via USB (with USB Debugging enabled).
  + Run flutter doctor -v to see any missing dependencies. Install them as directed.

**Acknowledgments**

* **BIP39**: For mnemonic generation and seed recovery.
* **ECDSA TSS Implementation**: Adapted from academic literature on threshold signatures.
* **Paillier Crypto Module**: Based on standard big-integer cryptosystems for additive homomorphism.
* **web3dart**: Simplifies Ethereum RPC calls in Dart.
* **Provider**: Chosen for lightweight state management.
* **Firebase**: Real-time sync between paired devices.

**Enjoy using BORAYS Crypto Wallet!** If you encounter any issues or have feature requests, please open an issue or submit a pull request.