International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 11 | Nov - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

Crypto Wallet: A Unified Multichain Cryptocurrency Wallet Using Blockchain Technology

Devaraj F V[1], Advith Gowda D A[2], Sujay Ram K G[3], Yashwanth S[4], Samyak U S[5],

Department of Computer Science and Engineering, Jawaharlal Nehru New College of Engineering, Shivamogga, Visvesvaraya Technological University, Belagavi, Karnataka, India

Abstract - Crypto Wallet is a non-custodial dashboard built with Next.js and Tailwind CSS. It supports Ethereum, Polygon, BNB Smart Chain, Base, Solana and Bitcoin through one BIP39 mnemonic, using BIP32/BIP44 derivation paths for each chain. Native and ERC-20 balances are retrieved through application API routes rather than direct browser requests to RPC providers. EVM and Solana transfers are prepared and signed on the user's device with ethers.js and @solana/web3.js; the server receives only signed transactions for broadcasting. Bitcoin support in this phase is limited to key and address derivation with bitcoinjs-lib. Mnemonics and private keys remain on the user's device and are never sent to the server in readable form; signatures are created locally. Server-side Next.js API routes handle RPC requests, transaction history and market prices, alongside a Supabase Postgres database for caching public information and wallet backups. Before upload, the browser encrypts backups using PBKDF2-SHA256 with 210,000 iterations and AES-GCM. The passphrase is not sent to the server, so backups remain ciphertext. We tested wallet creation and import with recovery phrases and raw private keys across six chains, ERC-20 token addition, key export and network switching. The paper reports functionality, issues and improvements, including threshold signatures and Bitcoin transaction signing.

Key Words: blockchain, cryptocurrency wallet, BIP39, BIP44, multichain, client-side encryption

I. INTRODUCTION

A. Background

A cryptocurrency wallet does not, in the literal sense, hold coins. What it holds is a private key, and the private key is what lets its owner sign a message the rest of the network will accept as authorisation to move value recorded against a public address. Bitcoin showed that a ledger could be kept honest without a bank in the middle. Ethereum generalised that into a machine that also runs code. Almost every chain since has inherited the same basic split between an address that is safe to publish and a key that is not, and that part has not really changed no matter how much else has. Ethereum, Polygon, BNB Smart Chain and Base sit this project. Solana uses Ed25519 signatures and its addresses are represented in Base58. Bitcoin uses the Base58Check format, based on a compressed public-key hash. These differences mean that the six networks do not all follow the same signing and address rules. This is one reason it can be difficult to manage assets across several blockchains. For example, MetaMask mainly supports EVM networks, Phantom is commonly used with Solana, and Electrum is designed for Bitcoin. A person using all three may therefore need separate recovery phrases and backup procedures. Keeping everything in one interface was the main idea behind our project; we wanted to avoid placing users' keys on a central server.

B. Motivation

A few things pushed us toward actually building this instead of just writing it up as an idea. First, most wallets in daily use are browser extensions, and an extension will talk to whatever page injects a provider object into it, which is more or less the exact attack surface phishing kits are designed around. Second, custodial risk has not gone away just because nobody wants to bring it up at a hackathon; upward of those losses illustrate the risk of placing many users' keys under one party. Another consideration is the way Web 3.0 applications work: DeFi services often need a wallet to approve and sign transactions within the page. A separate extension window can add another step to that process. Developers who work with several test networks also benefit from having one recovery phrase and the ability to configure RPC endpoints. This does not mean that hardware wallets or browser extensions should be replaced. They serve important purposes. Our aim is narrower: to provide a lightweight web option in which transaction signing stays on the user's device. That is the role Crypto Wallet is intended to fill.

C. Problem Statement

A user who works with Ethereum, Solana and Bitcoin may have to use different wallet applications and maintain a separate recovery phrase for each one. This creates several sets of credentials that must be stored safely and recovered when needed. Existing multichain choices may depend on browser extensions, custodial services or hardware devices. For this project, we wanted to examine a different approach: a browser-based, non-custodial wallet that derives accounts for multiple chains from one BIP39 mnemonic. Transactions are signed on the client device, and the server is not given the private keys.

D. Objectives

Going in, we set the following objectives for the project: 1. Build one web dashboard for EVM networks (Ethereum, Polygon, BNB Smart Chain and Base) as well as non-EVM networks (Solana and Bitcoin). 2. Add BIP39 mnemonic generation and BIP32/BIP44 hierarchical key derivation for each supported chain, using its relevant JavaScript library: ethers.js, @solana/web3.js or bitcoinjs-lib. 3. Retrieve native and ERC-20 balances through RPC and support transaction creation, signing and broadcasting within the application, without relying on a browser extension. 4. Keep key handling and signing on the client. Mnemonics and private keys should remain in the browser's Local Storage and must not be sent to the server in readable form. If cloud backup is used, the browser should encrypt the data with a passphrase before uploading it. 5. Connect RPC providers for all six networks and assess their response times and reliability during use. 6. Keep the interface straightforward for new users while still providing custom-token import and private- key export for users who need those options. These were the six objectives defined before implementation began. The server component described in Section III-E was added later. Initial testing revealed issues that could not be addressed on the client alone, including exposed provider credentials and transaction history that returned no results.

E. Scope

What we actually delivered covers wallet creation from a freshly generated twelve word mnemonic, and wallet import from either an existing mnemonic or a raw private key; native balance display across Ethereum (mainnet), Polygon, BNB Smart Chain, Base, Solana (mainnet-beta) and Bitcoin (mainnet); ERC-20 balance display and custom token import by contract address on the four EVM chains; native token transfer on the EVM chains and on Solana; and per chain private key export. The wallet itself is held in the browser's Local Storage, and no private key and no readable recovery phrase is handled on the server at any point, since the backup route accepts only a ciphertext blob the browser produced and the RPC proxy refuses any method that would ask a server to sign. On top of that the delivered system now includes a server side JSON-RPC proxy, explorer backed transaction history, cached market prices, and optional account based cloud backup of an encrypted recovery phrase. Bitcoin sending was never part of this phase, and Section V says more about why that matters.

II. LITERATURE SURVEY

The idea of keeping a key away from any single trusted party has been approached from a few different directions, and none of them quite land on a browser based multichain dashboard, which is really what makes the space worth walking through before getting into what we actually built. He, Lin, Li and Chen treat key backup as a flow network problem rather than a storage problem, which is a more interesting framing than it might sound [1]. In their Decentralised Multi Constrained Derangement scheme, every participating node is both a source for its own key fragments and a sink for fragments belonging to somebody else, with a derangement constraint stopping any node from ever holding a copy of its own data. An improved Edmonds-Karp max flow algorithm resolves the resulting assignment under extra exclusion rules, keeping a fragment away from any node that already shares transaction history with its owner, for instance. On top of that sits a Shamir-Kademlia-Neighbour redundancy layer: each fragment is split into eleven Shamir shares with a six share reconstruction threshold, and each share is replicated to two Kademlia neighbours. The authors report availability above ninety percent even with half the network offline, against a thirty eight percent vacancy ratio for a naive random distribution baseline. What we take from this is the underlying claim, that distributed, client held fragments can match or beat a centralised custodian on availability without concentrating trust anywhere. Crypto Wallet applies the same principle in a much simpler form, by keeping every fragment, which in our case is just the whole key, on the one device the user already trusts. Hardware isolation is the other classical answer, and Urien has written about it from two angles that are worth taking one at a time. The 2020 paper introduces the Crypto Terminal, an Arduino class device pairing a removable Java Card secure element (EAL5+, running a JVM certified under common criteria) with a touch screen and a Bluetooth Low Energy radio [2]. The whole unit is bare metal, so both the terminal firmware and the BLE firmware can be erased and reflashed on demand, closing off the kind of supply chain tampering that let a 2018 attack pull a seed out of a Ledger Nano S by inserting malicious code between bootloader and firmware. The harder problem Urien actually identifies is trusting the programmer that does the reflashing in the first place. A companion tutorial paper from 2019 answers that with the Big Bang paradigm: a dedicated Integrity Probe hashes every memory region in a pseudo random order and blinks out the resulting integrity code on LEDs before the reprogramming tool is even allowed to run [3]. Average probe timing comes out to about eighteen seconds, with roughly ten bits of entropy in the measured variance, which the paper argues is enough to make a tampered programmer statistically detectable. We borrow the philosophy here more than the hardware. Signing happens on the device the user already controls, with no separate terminal to trust or tamper with, and the price of that is losing the physical air gap a real Crypto Terminal gives you. A broader, more comparative look at what is already on the market comes from Suratkar, Shirole and Bhirud, who benchmark web, mobile, desktop and hardware wallets against supported currencies, key custody model, fiat on ramp support, anonymity, platform reach and recovery method [4]. Their headline conclusions, Jaxx and Guarda ahead among web wallets on anonymity and currency breadth, Coinomi leading on mobile, Atomic Wallet strongest on desktop, Ledger

Nano S the best balanced hardware pick, are worth reading with a small caveat: this survey is from 2020, and Jaxx in particular is not really the actively maintained product it was when the paper was written. What matters more for us is their cross cutting finding, that a twelve to twenty four word BIP39 mnemonic has become the de facto recovery standard across every category they looked at. That is exactly the design choice we made too, a single twelve word phrase as the root for all six chains rather than one phrase per chain. Kumar, Basak, KD and Nalband come at the same space from an architecture angle, proposing a four layer stack, application, blockchain, security and optimisation, aimed at a wallet that is secure, fast and light enough for constrained devices all at once [5]. It reads more like a position paper than an implementation report; there is no working prototype behind it, just the layered architecture and a fairly wide review of what other people have found. Even so, the optimisation layer's reliance on Simplified Payment Verification, off chain state channels and CDN backed delivery to stay synchronised without running a full node is the same trade off we make, just implemented instead of proposed, by querying public and provider hosted RPC endpoints rather than validating blocks ourselves. Their review also repeats a point worth keeping in mind here: Bitcoin's proof of work confirmation window runs to tens of minutes, long enough that any wallet promising instant finality across every chain it supports is glossing over a real difference in how EVM chains and Bitcoin actually settle. Multi party control of a single account is a fairly natural next step once single key custody is treated as the baseline, and Goel, Bisht and Chaudhary describe a working version of it [6]. Their multisignature wallet splits responsibility across a user interface, a wallet server holding private keys, a signature server collecting the required number of approvals before anything broadcasts, and the chain itself, built with Solidity contracts on Polygon's Mumbai testnet, Hardhat and Ethers.js for tooling, and, notably, a Next.js and Tailwind CSS front end, the same stack this project uses. Their standout feature is letting a user delegate signing authority to a nominated representative, useful for organisations where a transaction needs more than one person's sign off before it moves. Crypto Wallet only does single key accounts right now, but this multisignature model, together with the threshold ECDSA construction that came before it in the wider literature, is probably the clearest path to a phase two built around higher value or shared custody addresses. Not every threat to a wallet comes from outside the phone it is installed on. He, Li, Li, Zhu, Chan, Min and Guizani built an adversary model around an attacker who cannot see a target's private key directly, but who can install applications, change device settings, and lean on Android platform features, then validated it against two real wallets, Huobi Wallet and imToken [7]. An accessibility service tuned to fire on window content change events let their proof of concept walk the UI tree and lift a displayed mnemonic or private key straight off the screen. A second attack paired USB debugging with that same accessibility channel to reconstruct a password typed on a randomised on screen keyboard, defeating a security feature that was specifically built to resist exactly this kind of capture. Crypto Wallet is a browser application rather than a native Android app, so the accessibility service and ADB vectors this paper documents do not apply directly to it, but the underlying warning still does: anything rendered on screen can potentially be read by something else running alongside it. That is the actual reason the export screen gates a mnemonic reveal behind an explicit confirmation step instead of just showing it. Wang and Van Cutsem look past the wallet software entirely, to the node providers every wallet quietly leans on [8]. Testing MetaMask's default setup, they measured around one hundred and ten RPC calls in a ten minute window with no transaction initiated at all, mostly chain ID and block number polling, and showed that a single ethCall used just to price a token balance can hand every address a wallet controls to whoever is running the provider on the other end. Across the eight node providers they surveyed, most ask for at least an email address before issuing an API key, and six of the eight take only credit card payment, which quietly attaches a billing identity to what is supposed to be a pseudonymous address. This is directly relevant to the RPC endpoints Crypto Wallet ended up using, which Section V comes back to, because the same trade off between convenience and metadata exposure applies to any browser wallet that has to go through someone else's node to reach a chain at all. Closer to the tracking and portfolio end of things, Yadav, Sharma, Kumar and De describe a MERN stack portfolio manager pulling live price, market cap and volume for over a hundred coins from the CoinGecko API, rendering CanvasJS performance graphs across twenty four hour, seven day and one year windows, and protecting accounts with bcrypt's salted, deliberately slow hashing [9]. It is worth saying plainly that this is not a wallet in the sense the rest of this section uses the word. It never touches a private key. But the layered separation it uses, a public facing layer, an authenticated platform layer, an external data layer, is close to the shape Crypto Wallet's own server layer eventually took, with a public client, an authenticated API tier and cached third party market data behind it. Taken together, the nine papers surveyed here settle into four themes: decentralising key storage across a peer network [1], isolating signing inside dedicated hardware [2], [3], converging on a mnemonic based recovery standard [4], and layering a client so it stays light without running a full node [5]. None of them describes a single, open, browser based wallet that unifies EVM and non-EVM chains under one BIP39 root while keeping every signature on the user's device, and that is the specific gap Crypto Wallet is aimed at. We take the client side custody principle from [1] through [3], the mnemonic convention from [4], and the light client approach from [5], which we implement by proxying RPC traffic through our own routes rather than running a full node. The privacy finding in [8] cut the other way and changed the design: because a browser wallet talking straight to a

provider hands that provider both the user's address set and their IP, we moved provider traffic behind a server route, which also keeps the API keys out of the client bundle.

III. SYSTEM DESIGN

A. Architecture Overview

Crypto Wallet is a Next.js 14 application built on the App Router, rendered as a single mobile width container (app/layout.tsx, app/page.tsx) that switches between three top level screens, a welcome screen, a create-or-import flow, and the wallet dashboard, on client side React state alone. The wallet itself, the custom token list and the settings object are written to the browser's Local Storage through one dedicated module, StorageService, and nowhere else. The application is split across a trust boundary that is worth stating precisely, because the whole security argument rests on where it sits. On the browser side of that line are all the operations that touch secret material: mnemonic generation, key derivation, transaction signing, and the encryption of a backup blob. On the server side, under app/api/, are six route handlers that touch only public data: an RPC proxy, native and token balance lookups, transaction history, market prices, and storage of an already encrypted backup. The server can read a chain and it can hold ciphertext. It cannot sign, and it cannot decrypt. That boundary is enforced rather than merely documented. The RPC proxy at app/api/rpc/[chain]/route.ts rejects eth_sign, eth_sendTransaction, personal_ methods, eth_accounts and eth_requestAccounts outright, so no caller can talk the server into behaving like a key holder, and the backup route inspects incoming payloads and refuses anything that looks like a bare BIP39 phrase rather than ciphertext. We checked the boundary the slow way, by reading every file under app/, components/, lib/ and services/ ourselves rather than trusting the README to tell us.

B. Key Derivation Layer

All key material comes from a single BIP39 mnemonic, generated and validated with the bip39 package. From that seed, three separate derivation paths run depending on chain family. Ethereum, Polygon, BNB Smart Chain and Base all derive the same account at m/44'/60'/0'/0/0 through ethers.js's HDNodeWallet, and because all four share Ethereum's address format and signature scheme, we give them the exact same address and private key instead of four separate ones. That is a deliberate simplification, not something we overlooked. Solana derives at m/44'/501'/0'/0' using bip32, backed by the WASM build of tiny-secp256k1, to walk the HD tree, then @solana/web3.js's Keypair.fromSeed to turn the resulting 32 bytes into an Ed25519 keypair. Bitcoin derives at m/44'/0'/0'/0/0 through the same bip32 tree and hands the key to bitcoinjs-lib, which produces a legacy P2PKH address rather than a newer Bech32 one, mostly because that was the simpler path to get working first. Table I lays out all six chains as configured.

TABLE I. SUPPORTED NETWORKS AND DERIVATION PATHS

Chain	Symbol	Derivation Path	Address Format	Configured RPC / Data Source
Ethereum	ETH	m/44'/60'/0'/0/0	0x, 20-byte hex	/api/rpc/ethereum, resolved server side
Polygon	MATIC	m/44'/60'/0'/0/0	0x, 20-byte hex	/api/rpc/polygon, resolved server side
BNB Smart Chain	BNB	m/44'/60'/0'/0/0	0x, 20-byte hex	/api/rpc/binance, resolved server side
Base	ETH	m/44'/60'/0'/0/0	0x, 20-byte hex	/api/rpc/base, resolved server side
Solana	SOL	m/44'/501'/0'/0'	Base58, 32-byte	/api/rpc/solana, resolved server side
Bitcoin	BTC	m/44'/0'/0'/0/0	Base58Check (P2PKH)	/api/balance/bitcoin, Blockstream upstream

Import mirrors creation, more or less. A supplied mnemonic just regenerates all six accounts through the same derivation code we already had. A raw private key is handled differently depending on chain: an EVM key needs to be sixty four hex characters, with or without a 0x prefix, and once validated it gets applied identically across all four EVM chains at once, since they share an address anyway; a Solana key is accepted in Base58 (the format Phantom exports it in), thirty two byte hex, or sixty four byte hex; a Bitcoin key is accepted in WIF or sixty four character hex. A single chain private key obviously cannot produce keys for chains outside its own family, so importing by private key leaves the remaining chains populated with an explicit placeholder address, a string of repeated zeros for the EVM chains, repeated ones for Solana and Bitcoin, so the dashboard can tell a genuinely imported account apart from one that was never given a key, instead of just showing a zero balance for both and letting the user assume they mean the same thing.

C. Storage and Security Model

StorageService keeps exactly four keys in Local Storage: the wallet object itself, the list of custom tokens the user has added, the list of locally recorded transactions, and a settings object, all of it plain serialised JSON. Nothing under services/ or components/ makes a network call carrying a mnemonic or a private key in readable form; every RPC or REST call BlockchainService makes sends only a public address or an already signed transaction payload, never the key behind the signature. We checked this by reading every call site in blockchainService.ts ourselves rather than taking it on faith. Cloud backup is the one path where key material leaves the device at all, and it does so only after being encrypted. BackupService derives an AES-GCM key from a user chosen passphrase with PBKDF2-SHA256 at 210,000 iterations, the figure OWASP currently recommends for that construction, over a sixteen byte random salt, then encrypts a versioned payload with a twelve byte random nonce. The passphrase is never transmitted and never stored. What the browser uploads is ciphertext, salt, nonce and iteration count, which is exactly what is needed to decrypt later given the passphrase, and useless without it. The cost of that design, which the interface states plainly, is that a forgotten backup passphrase is

unrecoverable, since there is no custodian to appeal to. Two trade offs are worth stating outright rather than burying. Local Storage is still not encrypted at rest, so anyone with script execution in the page's origin, or physical access to the browser profile, can read the wallet; the export screen's own security warning says as much before it will reveal anything. And the server, while it cannot read a key, does now observe which addresses are being queried, which is the same metadata exposure Wang and Van Cutsem describe [8], moved from the provider to us. We consider that a better place for it, since the provider no longer sees the user's IP, but it is a relocation of the problem rather than its elimination.

D. Application and UI Layer

The interface runs on Tailwind CSS with a custom colour theme (trust-blue, #3375BB) chosen to look like the wallet branding people already recognise, and lucide-react supplies the icon set. The dashboard shows one balance card and one asset list per selected chain, with Send, Add Token, Transaction History, Cloud Backup and Export Wallet as separate views reached from that same screen rather than as separate routes, so the whole thing stays a single page as far as the browser's address bar is concerned. When price data is available the balance card also shows an approximate fiat value beneath the native amount.

E. Server Layer and Data Model

The server side is deliberately small. Six route handlers live under app/api/. The RPC proxy at /api/rpc/[chain] accepts a JSON-RPC body, screens the method against a blocklist, and forwards it to whichever upstream endpoint lib/serverNetworks.ts resolves for that chain. That resolution prefers an explicitly configured endpoint, then a keyed provider, then a public fallback, so the wallet still runs with no credentials at all, just with lower rate limits. Because both ethers.js and @solana/web3.js speak JSON-RPC, pointing them at this route left every existing call site unchanged. /api/balance/[chain]/[address] and /api/token-balance/[chain] serve native and ERC-20 balances, /api/tx-history/[chain]/[address] returns normalised transaction history, /api/prices returns cached market data, and /api/backup stores and retrieves an encrypted blob for an authenticated user. Persistence is a Supabase Postgres instance reached through two clients, kept separate on purpose. lib/supabase/server.ts holds the service role client, marked with the server-only package so an accidental import from a client component fails at build time rather than leaking a database key into the bundle. lib/supabase/client.ts holds the browser client, which uses the publishable anon key and is constrained by Row Level Security. Four tables sit behind them: profiles, wallet_backups, cached_transactions and cached_prices. Only wallet_backups holds anything derived from a user secret, and it holds only ciphertext. Every table is scoped by user id under Row Level Security, so a bug in a route handler still cannot hand one user another user's rows. The cache tables carry a fetched_at timestamp and are treated as disposable; a cache miss or a Supabase outage falls through to a live fetch rather than failing the request, which keeps the wallet usable when the database is not configured at all.

IV. IMPLEMENTATION

A. Wallet Creation and Import

Creating a wallet calls WalletService.generateMnemonic(), shows the resulting twelve words blurred out behind a "Reveal Phrase" control, and only unlocks the confirmation button once the user has both revealed the phrase and ticked a box confirming it has been saved somewhere. At that point createMultiChainWallet() derives all six accounts in one pass and StorageService.saveWallet() writes them to storage. Import offers the same mnemonic path, plus a second, per chain private key path, both reachable from a single toggle on the same screen, working exactly the way Section III-B describes.

B. Balance and Token Retrieval

Native balances come through BlockchainService.getBalance(), which branches by chain: ethers.JsonRpcProvider.getBalance() for the four EVM chains and Connection.getBalance() from @solana/web3.js for Solana, both now pointed at /api/rpc/<chain> rather than at a provider directly, and, for Bitcoin, a call to /api/balance/bitcoin, which reaches Blockstream's Esplora API from the server and sums funded_txo_sum against spent_txo_sum to get a confirmed balance in satoshis. ERC-20 balances go through a six function ABI covering name, symbol, decimals, balanceOf and transfer, queried with ethers.Contract. Custom token import calls those same metadata functions before writing anything to storage, so an address that does not answer like an ERC-20 contract gets rejected before it ever reaches the asset list.

C. Transaction Construction, Signing and Broadcasting

EVM transfers are signed locally with ethers.Wallet, either as a plain sendTransaction call for the native asset or a Contract.transfer call for a token, and the code waits on tx.wait() before handing a hash back to the UI. The wallet is bound to a provider aimed at our own RPC route, so what travels to the server is a transaction that has already been signed on the device, never the key that signed it, and the proxy's method blocklist means the server cannot be asked to produce a signature itself. Solana transfers are assembled by hand with SystemProgram.transfer, signed with the derived Keypair, and pushed out with sendRawTransaction through the same route, after which the code polls getSignatureStatus once a second for up to thirty seconds instead of opening a WebSocket subscription, a choice the source comments call out directly as a way of avoiding a dependency on persistent socket support. Bitcoin transfers are not implemented at all; calling sendTransaction for Bitcoin just throws a "not yet implemented" error, which lines up with the project's

own scope statement, since UTXO selection and PSBT signing were never on the list for this phase. Only balance display was.

D. Export

The export view sits behind its own security warning screen, and it will not go any further until the user ticks a box acknowledging the risk. Past that, the mnemonic can be revealed on demand behind an eye icon, and each of the six chains lists its address by default, with a per row eye icon that swaps the display over to the raw private key, and its own copy to clipboard button next to it. That directly covers the "private key export for advanced users" objective from Section I.

E. History, Prices and Encrypted Backup

BlockchainService.getTransactionHistory() is no longer a stub. It calls /api/tx-history, which serves a cached page when one is less than two minutes old and otherwise fetches live. For the four EVM chains that means Etherscan's v2 API, where a single key and host covers all four by chain id; for Solana it means getSignaturesForAddress against the same RPC endpoint the wallet already uses; for Bitcoin it means Blockstream's address transaction endpoint. Each source is normalised into one row shape, direction included, so the UI renders every chain through the same component. Writing to the cache is best effort by design, and a failure there is swallowed rather than propagated, since a caching problem should not turn into a failed history lookup. Prices come from /api/prices, which maps the five native symbols onto CoinGecko ids, caches for five minutes, and, when a live fetch fails, deliberately falls back to stale cached rows rather than returning nothing, on the reasoning that an out of date figure is more useful on a portfolio screen than a blank one. Cloud backup is handled by CloudBackup.tsx over BackupService and /api/backup, with Supabase Auth providing the account. The encryption described in Section III-C happens before the upload, and the route authenticates the caller from a bearer token, resolves the user id server side, and scopes every read and write to it.

V. RESULTS AND DISCUSSION

We ran the application locally and went through it by hand rather than judging it from the source alone, so everything in this section is something we actually watched happen, not something we inferred should happen. This section reports two rounds. The first covered the client only build and is what surfaced the limitations listed in Section V-B. The second followed the server layer, and is what let us mark several of those limitations closed rather than merely acknowledged.

A. Verified Functionality

Wallet creation produced a valid, correctly formatted twelve word mnemonic every time we tried it, and the resulting dashboard listed all six configured chains with addresses in the right format for each family: a 0x prefixed forty character hex string shared identically across Ethereum, Polygon, BNB Smart Chain and Base, a Base58 Solana address, and a Base58Check Bitcoin address starting with the legacy 1 prefix. We exercised both import paths. A freshly generated EVM private key produced a matching address and populated the dashboard right away, and the two chains that key obviously could not derive, Solana and Bitcoin, fell back to the documented placeholder address instead of crashing or quietly showing a wrong balance, which is what we were actually worried might happen. The Add Token screen stayed correctly restricted to the four EVM chains and turned away placeholder or malformed contract addresses before they could be added. The export flow refused to open without the security warning gate first, and once past it, showed distinct, correctly formatted keys for each of the six chains. Table II lines up the six objectives from Section I-D against what this testing actually found.

TABLE II. OBJECTIVE VERIFICATION SUMMARY

#	Objective	Outcome
1	Unified dashboard for EVM and non-EVM chains	Met. All six chains present and independently selectable.
2	BIP39/BIP44 derivation per chain library	Met. Paths and libraries confirmed in source and by address inspection.
3	Real time balances, in-app signing, no extension	Partially met. Balances and signing work for five of six chains; Bitcoin sending is still not implemented.
4	Client side signing, no readable key on the server	Met, with the scope restated. A server layer now exists, but signing stays on the device, the RPC proxy blocks every signing method, and cloud backup stores only browser encrypted ciphertext.
5	Multiple RPC providers, responsiveness observed	Met. Provider selection moved server side with a public fallback, which resolved the CORS and credential failures seen in the first round.
6	Interface usable by novices and advanced users	Met. Guided creation flow alongside private key export and custom token import.

B. Limitations Observed

The first round of testing turned up five concrete issues that reading the source alone would not have caught. All five are now closed, three of them by the server layer and two by a later pass over the client, and it is worth being specific about how each was closed rather than reporting a single tidy number. RPC connectivity was the biggest problem. The public Ethereum endpoint wired up for everyday use, eth.llamarpc.com, rejected browser initiated requests with a CORS preflight failure in our environment, and the Tatum and Ankr credentials sitting in the project's .env file both returned HTTP 401, most likely because they were sample keys left over from the tutorial the project was originally built from rather than credentials issued for this deployment. Related to that, those keys carried a NEXT_PUBLIC_ prefix, which meant they were inlined into the client bundle and readable by anyone who opened developer tools. Both problems are gone in the current build, and for the same reason: provider selection moved into lib/serverNetworks.ts behind /api/rpc/[chain], the keys lost their NEXT_PUBLIC_ prefix and are read only in route handlers, and the browser no longer makes a cross origin request to a provider at all, so there is no preflight to fail. A public fallback endpoint means the wallet still functions with no keys configured. Transaction history

was the third. It used to be a stub that returned an empty array, so an imported address with years of activity showed nothing. It is now served by /api/tx-history from Etherscan, Solana RPC and Blockstream, normalised into one row shape and cached in Postgres for two minutes.

The first round of testing exposed five issues that were not apparent from source inspection. Three were resolved through the server layer and two through later client changes. RPC connectivity was the main problem: eth.llamarpc.com rejected browser requests with a CORS preflight failure, while the Tatum and Ankr credentials in the project's .env file returned HTTP 401. Their NEXT_PUBLIC_ prefixes also exposed them in the client bundle. Provider selection now runs in lib/serverNetworks.ts behind /api/rpc/[chain]; credentials are read only by route handlers without the NEXT_PUBLIC_ prefix, and the browser no longer contacts providers directly. A public fallback keeps the wallet usable when no keys are configured. Transaction history, previously an empty-array stub, is now retrieved from Etherscan, Solana RPC and Blockstream through /api/tx-history, normalised into a common row format and cached in Postgres for two minutes.

Two further client issues were corrected. Duplicate assets were prevented by filtering user-added tokens against the common token list using case-insensitive address comparison. The Send screen's fixed ~0.001 fee was replaced by BlockchainService.estimateFee(), which uses maxFeePerGas and the relevant gas estimate (21,000 for native transfers and 65,000 for token transfers); Solana uses its 5,000-lamport signature fee. When an estimate is unavailable, as for Bitcoin while sending remains unimplemented, the screen displays “Unavailable”. The server layer also has limitations: EVM history requires ETHERSCAN_API_KEY; Solana history includes hashes, timestamps and statuses but not amounts; encrypted cloud backups cannot be recovered if the user forgets the passphrase; and address-query metadata is now visible to our server, although not to the RPC provider. Throughout balance retrieval, signing and broadcasting, the mnemonic and private keys remain in the browser in readable form only. Cloud backup is the sole path by which key-derived material leaves the device, and it is sent only as AES-GCM ciphertext.

VI. CONCLUSION AND FUTURE SCOPE

Building Crypto Wallet demonstrated that one BIP39 mnemonic can support three different signature schemes and address formats without exposing private keys to a server, provided each chain family uses the correct derivation and signing logic on the client. Testing also clarified the practical meaning of a client-side wallet. Direct browser access to chain providers exposed credentials in the client bundle or caused requests to be blocked. Moving read requests behind our own API routes addressed these problems without changing the essential custody property: the server cannot sign transactions or read private keys. This refines our original client-only objective rather than abandoning its security intent. The six objectives were fully met for four chains and mostly met for the other two; Bitcoin transaction signing remains outside the implemented scope. All five issues found in the first testing round were resolved, although the server layer introduced the limitations described in Section V-B.

Future work includes threshold or multisignature approval for high-value transfers, as discussed by Goel, Bisht and Chaudhary [6], and hardware-backed key isolation based on the approaches described by Urien [2], [3]. The remaining practical improvements are encryption of the Local Storage payload at rest and implementation of UTXO-based Bitcoin transaction signing. These changes would strengthen the current proof of concept and make it more suitable for cautious day-to-day use.

REFERENCES

[1] X. He, J. Lin, K. Li, and X. Chen, "A novel cryptocurrency wallet management scheme based on decentralized multi-constrained derangement," IEEE Access, vol. 7, pp. 185250-185263, 2019, doi: 10.1109/ACCESS.2019.2961183.

[2] P. Urien, "Crypto terminal: A new open device for securing blockchain wallets," in Proc. 2020 IEEE Int. Conf. Blockchain and Cryptocurrency (ICBC), Toronto, Canada, 2020, pp. 1-3, doi: 10.1109/ICBC48266.2020.9169410.

[3] P. Urien, "Wallets and crypto terminals for blockchain; introducing the Big Bang paradigm," in Proc. 2019 5th Conf. Mobile and Secure Services (MobiSecServ), 2019, pp. 1-1, doi: 10.1109/MOBISECSERV.2019.8686561.

[4] S. Suratkar, M. Shirole, and S. Bhirud, "Cryptocurrency wallet: A review," in Proc. 2020 4th Int. Conf. Computer, Communication and Signal Processing (ICCCSP), Chennai, India, 2020, pp. 1-7, doi: 10.1109/ICCCSP49186.2020.9315193.

[5] H. Kumar, S. Basak, S. KD, and A. H. Nalband, "Enabling secured and seamless crypto wallets: A blockchain solution," in Proc. 2023 2nd Int. Conf. Vision Towards Emerging Trends in Communication and Networking Technologies (ViTECoN), Vellore, India, 2023, pp. 1-6, doi: 10.1109/ViTECoN58111.2023.10157044.

[6] A. K. Goel, V. S. Bisht, and S. Chaudhary, "Multisignature crypto wallet," in Proc. 2023 8th Int. Conf. Communication and Electronic Systems (ICCES), Coimbatore, India, 2023, pp. 476-479, doi: 10.1109/ICCES57224.2023.10192591.

[7] D. He, S. Li, C. Li, S. Zhu, S. Chan, W. Min, and N. Guizani, "Security analysis of cryptocurrency wallets in Android-based applications," IEEE Network, vol. 34, no. 6, pp. 114-119, Nov./Dec. 2020, doi: 10.1109/MNET.011.2000025.

[8] W. Wang and T. Van Cutsem, "I know what crypto you got last summer: A study on Web3 wallet security and privacy," in Proc. 2024 6th Conf. Blockchain Research and Applications for Innovative Networks and Services (BRAINS), 2024, doi: 10.1109/BRAINS63024.2024.10732070.

[9] R. Yadav, A. Sharma, S. Kumar, and P. De, "Tracking and managing of cryptocurrency wallet," in Proc. 2024 Int. Conf. Advances in Modern Age Technologies for Health and Engineering Science (AMATHE), 2024, doi: 10.1109/AMATHE61652.2024.10582164.