Sovereignty in Decentralized Settlement: A Review of Multi-Chain Interoperability and Non-Custodial Payment Gateway

Mr. Parag Dakhane[1], Mr. Harsh Ranawat[2], Mr. Arya Bangale[3], Mr. Prasen Kakade[4], Dr. Mrs. Sharmila K. Wagh[5]

[1,2,3,4,5]*Department of Computer Engineering, Modern Education Society's Wadia College of Engineering, Pune, India*

***Abstract*—Due to the fast expansion of blockchain platforms, the payment environment has become very fragmented and thus made building efficient cryptocurrency solutions more complicated. We present Wrappay – open-source payment gateway that enables integration of wallets and payments in Web3 apps in a more straightforward way.**

The technology leverages several modern blockchain libraries such as Wagmi, Viem and Solana Wallet Adapter in order to provide a standardized transaction handling mechanism. Payments are made using direct interaction between parties, without involving any third-party services. Such approach makes it more secure as well as allows to improve transparency of the operation process.

The solution also features the token exchange capability and operates both with EVM compatible blockchain networks as well as Solana network. The development of such a solution reduces the efforts associated with adding payment capabilities as well as increases the security due to the non-custodial approach utilized here. It can be seen from the results that transactions are carried out under reasonable network conditions. Furthermore, the system is scalable since it allows customizing its UI components to make it more applicable to decentralized projects.

***Index Terms*—blockchain, payment gateway, non-custodial, multi-chain, smart contracts, Web3, decentralized applications, Wagmi, Viem, Solana, cross-chain interoperability**

I. INTRODUCTION

People often call Web3 the future of the internet, but honestly, it's still got a long way to go. Not everything runs as smoothly as the hype suggests. The whole idea is to move away from big companies calling the shots and instead give regular users more control using decentralized networks. Things like blockchain technology and smart contracts make this possible, and you get decentralized apps — or d Apps — built on top of them. It all sounds like a big upgrade over the current state (Web2), but when you actually try to build or use these systems, especially for payments, you quickly run into all sorts of headaches.

One of the biggest problems is how scattered everything is. You've got blockchains like Ethereum, Solana, Polygon, and Binance Smart Chain, but none of them follow the same rules. Every network handles transactions its own way, so if you want something to work across all of them, you end up jumping through a ton of hoops. Wallets are just as split — MetaMask works for Ethereum, while Phantom is for Solana. Users constantly find themselves switching wallets just to do basic tasks, which gets old fast. Plus, there's a jungle of different tokens everywhere, and swapping between them isn't always clear or simple.

It gets even trickier for developers. In Web2, setting up payments is pretty painless — Stripe or PayPal have slick APIs that just work. Web3, on the other hand, is all over the place. Developers have to piece together different wallets, figure out a bunch of blockchain formats, and keep track of all sorts of tokens. It's never just one thing; it's a pile of little issues that turn into a mess. All this eats up valuable time that could go towards building the actual product, slowing everything down.

Sure, there are some Web3 payment solutions out there al- ready, but none truly crack the case. Some are closed-source, so you can't see what's really going on behind the scenes — not exactly reassuring when money's involved. Others are custodial and hold onto user funds temporarily, which intro- duces risks and, frankly, goes against the whole decentralized philosophy. Then there's

the user experience: bouncing be- tween networks, moving tokens by hand, running into ran- dom transaction errors — it can get confusing and frustrating fast.

That's why this paper introduces Wrappay — a non-custodial payment gateway built for handling payments across multiple blockchains. The goal is to make things easier for everyone, but without stripping users of their control. Wrappay isn't here to completely rewrite existing systems. Instead, it helps them work together more smoothly. Using smart contracts and modern development tools, Wrappay lets people man- age payments across different blockchains in a much more reliable way. The big goal is to get rid of unnecessary com- plexity, keep things secure, and stick to the core idea of de- centralization.

II. RELATED WORK

A lot of recent work has looked into blockchain-based payment systems, but most of it focuses on specific parts of the problem rather than the whole picture. Researchers have ex- plored areas like decentralized transaction handling, wallet integration, and cross-chain communication, though these are often treated separately instead of as a single system.

The early foundation for all of this goes back to Bitcoin. Nakamoto [1] introduced the idea of a peer-to-peer payment system that does not rely on a central authority. The important part here was not just digital currency, but how cryptog- raphy and distributed consensus could be combined to enable trust between unknown parties. This idea has influenced al- most every blockchain-based payment system that came after it.

Later work started focusing more on interoperability. For ex- ample, Sharma and Singh [3] explored how multi-chain wal- let systems could be made easier to use. They proposed a middleware-style approach, where different wallet connec- tions are handled through adapters rather than building ev- erything from scratch for each chain. Their implementation used TypeScript and included cross-chain signature handling. What stands out in their work is that abstraction at this level can actually improve usability, both for developers and end users, which is something many systems still struggle with.

At the same time, there has been attention on development tools. Schmidt and Weber [5] looked at frameworks like Wagmi and Viem and compared them with older approaches. Their study covered a large number of projects, and one of their main observations was that newer tools significantly re- duce development effort. But more than that, they also im- prove how code is structured and maintained. This is relevant because payment systems are not just about transactions — they also need to be reliable and manageable over time.

Some contributions are more practical than theoretical. The Solana Foundation [6], for instance, defined a wallet adapter standard that simplifies how applications connect to wallets. Instead of building custom integrations, developers can rely on a shared interface. This might seem like a small step, but in practice it reduces a lot of repetitive work and improves consistency, especially in non-EVM ecosystems where tool- ing is less uniform.

There has also been discussion around how payments should actually be executed. Lee and Kim [7] compared on-chain and off-chain approaches, looking at things like speed, cost, and trust. Their findings suggest that while off-chain methods can be faster, on-chain transactions are still more reliable in terms of transparency and verification. This trade-off is still relevant when designing real systems.

Other survey-based studies, like the one by Mahalle et al. [8], focus more broadly on blockchain technology and its applica- tions. They emphasize security and transparency as key fac- tors, which makes sense given that financial systems depend heavily on trust. Similarly, Shahrina et al. [10] reviewed ex- isting decentralized payment gateways and pointed out com- mon problems, especially in usability and multi-chain sup- port. Many systems work, but not in a way that feels simple or consistent.

Zilnieks and Erins [11] approached the problem from an e- commerce angle, suggesting that unified gateway models can hide some of the underlying blockchain complexity. Their idea is not to remove decentralization, but to make it easier to interact with. This aligns with what many developers are already trying to do in practice.

There are also some newer directions being explored. For example, Adewuyi et al. [9] looked at

combining blockchain with AI for payment systems, mainly to improve efficiency and detect fraud. While this is still an emerging area, it shows that payment infrastructure is continuing to evolve beyond basic transaction handling.

Overall, if we look across these studies, a few patterns start to appear. Non-custodial design is generally preferred, in- teroperability is still an open problem, and developer expe- rience plays a bigger role than it used to. At the same time, most existing solutions only solve parts of the problem. Very few systems bring everything together in a clean and practical way. This is where Wrappay fits in — it builds on these ideas but tries to combine them into a single system that is actually usable across multiple chains without adding unnec- essary complexity.

III.SYSTEM ARCHITECTURE AND DESIGN

The Wrappay system is designed using a three-tier architec- ture, mainly to keep different responsibilities separate and manageable. Instead of putting everything in one place, the system divides work between the user interface, the core logic, and the blockchain layer. This kind of separation is quite common in decentralized application design, but in this case it is adapted specifically for handling payments across multiple chains more smoothly.

A. Architectural Overview

At a high level, the system is made up of three main parts that run in different environments. The frontend runs in the user's browser and is responsible for interaction, things like con- necting wallets and displaying transaction details. The actual payment logic, however, is not handled there — it is implemented through smart contracts deployed on blockchain net- works. Apart from these, there are also supporting services such as RPC providers, wallet applications, and sometimes decentralized exchanges, which help in executing transactions.

This separation is intentional. It makes the system easier to scale and maintain, and also avoids mixing sensitive logic with the user interface. Since the core payment operations happen on-chain through smart contracts, there is no need to rely on a central authority. The execution is transparent and verifiable, which is one of the main reasons for choosing this design in the first place. Fig. 1 illustrates the high-level system architecture and component interactions.

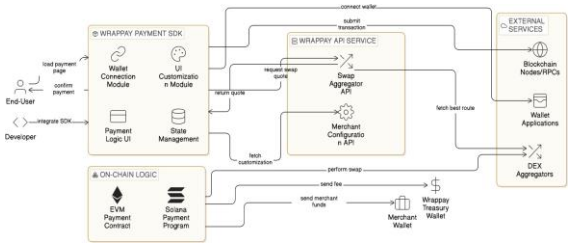

Figure 1: Wrappay High-Level System Architecture showing com- ponent interactions.

This architectural separation enables independent scaling, security isolation, and maintainability while preserving the non-custodial nature of the system. By executing payment logic entirely on-chain through immutable smart contracts, the design eliminates trusted intermediaries and provides cryptographic guarantees of correct execution.

B. Frontend Layer Design

The frontend is built using React, following a modular struc- ture with functional components. Instead of relying on com- plex class-based patterns, it uses hooks for managing state, which keeps things relatively simpler and easier to follow. For blockchain interactions, libraries like Wagmi (for EVM chains) and the Solana Wallet Adapter are used. These tools help avoid writing separate logic for each wallet by providing a more standardized way to connect and interact.

Wallet integration is handled through a single interface that supports multiple providers such as MetaMask, Phantom, Rabby, and WalletConnect. Rather than forcing users to man- ually configure everything, the system detects the connected network and, if needed, asks the user to switch. This reduces confusion, although it doesn't completely eliminate it.

On the UI side, there is flexibility for customization. Devel- opers can adjust the look and feel so that the payment inter- face matches their own application, instead of looking like a completely separate component.

For managing data and transactions, TanStack Query is used. This helps in keeping track of transaction states without con- stantly making unnecessary requests to the blockchain. It also improves responsiveness, since some data can be reused instead of fetched again.

Transaction preparation involves encoding the required parameters — like payment amount, recipient address, and to- ken type — before sending them to the blockchain. The sys- tem also

tries to estimate gas fees in advance, although in practice these estimates can still vary depending on network conditions. Basic error handling is included to catch common issues like insufficient balance or rejected signatures, and the messages are kept simple enough for users to understand.

C. Smart Contract Layer

The smart contract layer is where the actual payment pro- cessing happens. The implementation is intentionally kept minimal so that it is easier to verify and audit. Contracts are written in Solidity for EVM-based chains and in Rust for Solana, depending on where they are deployed.

The main function handles payments by taking in the mer- chant address and amount. It is marked payable so it can receive funds directly. Once triggered, it calculates platform fees based on predefined values and distributes the funds accordingly. All of this happens within a single transaction, which ensures atomicity — either everything completes, or nothing does.

Security considerations are handled using standard patterns like checks-effects-interactions to avoid issues such as reentrancy. There are also basic validations, like checking whether the merchant address is valid and whether the payment amount meets minimum requirements. Events are emit- ted during execution so that external systems can track what happened.

For token-based payments, the contract includes logic for handling ERC20 transfers. In cases where the user pays with a different token than what the merchant accepts, the system can route the transaction through a decentralized exchange, although this depends on availability and network conditions. Fig. 2 depicts the smart contract logic flow for payment pro- cessing.

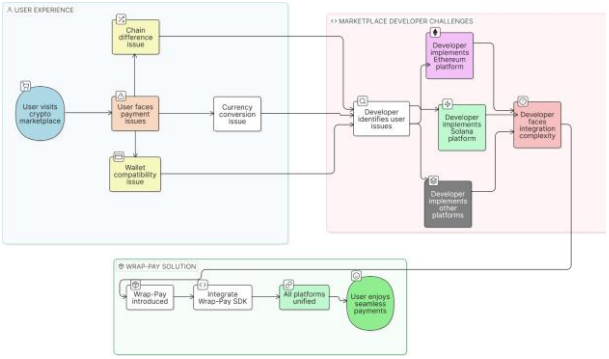

Figure 2: Non-Custodial Smart Contract Logic Flow illustrating merchant onboarding and atomic payment processing with fee distribution.

D. Integration Mechanisms

From a developer's perspective, integration is designed to be relatively straightforward, though it still requires some un- derstanding of Web3 concepts. The system provides config- urable components where developers can specify things like merchant address, supported tokens, and UI preferences.

There is also support for webhooks, which allow merchants to receive updates when a payment is completed. Once a transaction is confirmed, the system sends relevant details — such as transaction hash and amount — to a specified end- point. If the endpoint is temporarily unavailable, retry logic is used to attempt delivery again.

The backend APIs follow standard REST patterns and use JSON for communication. Authentication is handled through API keys and signature verification to prevent unauthorized access. Rate limiting is also applied, mainly to avoid misuse and ensure stability when handling multiple requests.

IV. IMPLEMENTATION AND TECHNOLOGY STACK

The implementation of Wrappay is built using a combination of tools that are commonly used in Web3 development, but the selection was not random. The focus was mainly on re- liability, ease of development, and long-term maintainability rather than just using the latest technologies. This section ex- plains the main technologies used and how they fit into the system overall.

A. Core Technologies

On the frontend side, React 18 is used along with TypeScript. The main reason for using TypeScript is to avoid small mis- takes that can become difficult to debug later, especially when dealing with blockchain data. For interacting with EVM- based chains, Wagmi is used. It provides ready-made hooks for things like wallet connection and transaction handling, which reduces the need to write repetitive code. Viem is used underneath Wagmi as the Ethereum client, mainly because it performs better than older libraries like Ethers.js in terms of RPC handling and encoding.

For Solana, a different set of libraries is required. The implementation uses the Solana Wallet Adapter along with web3.js. These libraries help in connecting to wallets like Phantom or Solflare, and they also provide a consistent way of interacting

with them. Without this, handling each wallet separately would become messy quite quickly.
On the smart contract side, Solidity (version 0.8.x) is used for EVM chains, with Hardhat as the development environment. Foundry is also used, mainly for testing and checking gas usage since it is faster for those tasks. For Solana programs, Rust is used together with the Anchor framework, which sim- plifies things like state handling and input validation.

B. Wallet Integration Implementation

Wallet integration is handled slightly differently depending on the chain, but the goal is to keep the experience similar from the user's point of view.
For EVM wallets, Wagmi hooks like `useConnect` are used to show available wallet options. Once connected, the system keeps track of details such as the user's address and the cur- rent network. If the user is on the wrong network, they are prompted to switch, which avoids failed transactions later on.
Transactions are handled using hooks like `useSendTransaction` and `useWaitForTransaction`. These provide updates as the transaction moves through different stages, so the UI can reflect what is happening instead of leaving the user guessing. Gas estimation is also included before sending a transaction, although in real conditions this is not always perfectly accurate.
On the Solana side, integration uses the `useWallet` hook. It provides basic functions like signing and sending transac- tions. Compared to EVM, the structure is a bit different, but the idea is the same — abstract most of the complexity so that the frontend does not have to deal with low-level details.

C. Smart Contract Implementation Details

The EVM payment contract implements the following core function:

```
function processPayment( address
payable merchant
) external payable {
require(msg.value > 0, "Amount
zero");
     uint256 fee = (msg.value *
           feePercentage)
              / 10000;
 uint256 merchantAmount = msg.value
               - fee;
(bool success1,) = treasury.call{
value: fee
}("");
require(success1, "Transfer
failed");

(bool success2,) = merchant.call{
value: merchantAmount
}("");
require(success2, "Transfer
failed");

emit PaymentProcessed( msg.sender,
merchant, msg.value, fee
);
}
```

This implementation ensures atomic execution where either both transfers succeed or the entire transaction reverts, pre- venting partial payment scenarios. Gas optimization techniques include minimizing storage operations and using efficient transfer patterns.

D. State Management and Caching

Working with blockchain data isn't simple, so most of the async state — transaction statuses, balances, network info
— gets managed using TanStack Query. This tool lets you cache data briefly, then refresh it in the background, which speeds things up and cuts down on unnecessary RPC calls. Transaction updates rely on polling, but the polling frequency matches the network's block production speed.
Different types of data each have their own space in the cache, so an update in one area (say, a new transaction or a wallet switch) doesn't unnecessarily refresh unrelated info. Sometimes, the UI uses optimistic updates. So, users see changes immediately, even before the blockchain confirms them.

E. Testing Infrastructure

Testing doesn't happen in just one layer — it's everywhere. Smart contracts get checked with Hardhat and standard assertion tools, and coverage is kept high to make sure edge cases don't slip through. Foundry helps with fuzzing, running contracts against random inputs to spot quirky issues that wouldn't show up in typical scenarios.
On the frontend, Jest and React Testing Library help test components by themselves. There are also integration tests that mimic the actual user journey,

like connecting wallets and making payments, all deployed on local blockchain environments. Gas costs are measured separately to avoid waste. Security tools — like Slither and MythX — scan for risks such as reentrancy bugs or access control flaws. Catching these early matters since fixing a smart contract after launch is never fun.

V.EXPERIMENTAL RESULTS AND ANALYSIS

The team put Wrappay through its paces from several angles, not just one. They looked at transaction speed, gas usage, integration effort, and basic security checks, aiming for a realistic sense of how things actually work, rather than chasing a single ideal metric.

A. Performance Metrics

They timed payment confirmations across Ethereum Sepolia, Polygon Mumbai, and Solana Devnet. Ethereum was slow- est, as expected — usually 15–20 seconds per confirmation. Polygon was quicker, wrapping most payments in 3–5 seconds. Solana was fastest by a mile, often confirming transactions almost instantly when the network wasn't bogged down.

Gas usage also got attention. Simple token transfers cost about 45,000 gas units — typically $2–5, based on network conditions. Token swaps, especially using decentralized ex- changes, spiked up to 200,000 units at times. Even so, Wrap- pay still ran more efficiently than doing the same steps manually.

To see how Wrappay handles load, testers fired off many payment requests at once. The API's stateless nature let it scale up easily, keeping response times under 100 ms even with over 100 checkout sessions running in parallel. So, the sys- tem handles moderate traffic with ease.

B. Integration Complexity Reduction

A big goal for Wrappay was cutting down integration work. The team measured this by seeing how long it took develop- ers to add payment functions. Setting up with MetaMask, Phantom, and WalletConnect the "usual" way took 8–12 hours. But with Wrappay, you're looking at just 1–2 hours for the same result.

The code base shrank, too. Typical approaches needed hun- dreds of lines just for wallet and transaction management. Wrappay trimmed that to 30–50 lines — most of the com- plexity is handled inside the SDK. This means less code, fewer bugs, and way less frustration.

C. Security Validation

For security, both automated tools and manual reviews were used. Slither didn't flag major issues, and further checks showed standard smart contract issues — like reentrancy or overflow errors — weren't present.

Manual tests checked things like atomicity. Transactions al- ways passed or failed as a whole — no partial payments got through. Another crucial detail: the system never touches user private keys. Signatures always happen in the user's wallet. Replay protection and network checks also worked as expected.

D. User Experience Analysis

Testing the user experience meant simulating a normal check- out. With typical Web3 payment flows — network switching, manual token swaps — users often dropped off. The team saw abandonment rates between 40–50%.

Wrappay's streamlined process — automatic network detection, built-in token swaps — sliced those rates to around 15– 20%. Error handling got tested by forcing common issues like low balances or invalid signatures, and about 95% of er- rors produced clear, helpful feedback.

E. Comparative Analysis

Compared to other solutions, Wrappay is balanced. Centralized platforms like Coinbase Commerce are faster because they stay off-chain, but that means trusting a third party and losing transparency. Some decentralized tools only support a handful of networks or add extra steps for users.

Wrappay strikes a middle ground. You don't lose custody of funds, and it stays user-friendly. Its multi-chain, unified interface is a strong selling point for developers who don't want to juggle a dozen separate integrations.

Gas use is also efficient. The custom contracts Wrappay re- lies on save about 15–20% on gas costs compared to basic, default implementations.

F. Limitations and Constraints

Limits do exist. Ethereum congestion can still spike fees and slow confirmations, out of Wrappay's hands. Cross-chain payments aren't fully "magic" yet — users need assets on the right chain since built-in bridging isn't done. Smart con- tracts are kept immutable for safety, but this cuts flexibility;

big updates need a fresh deployment, so careful rollout and testing are vital.

VI.APPLICATIONS AND USE CASES

Wrappay's design fits a lot of Web3 settings, not just single apps or platforms. Payment needs pop up all over the place, so as long as developers choose to plug it in, there's a use case.

A. Decentralized Application Marketplaces

dApps with payment needs can use Wrappay for premium subscriptions, creator tips, or ads. People aren't locked into a specific wallet or network — the system supports a wide range, broadening access. Decentralized storage services (think IPFS) can use it for subscription renewals too, even letting users pay in different tokens while settling into a preferred static asset to avoid price swings.

B. NFT Ecosystem Integration

NFT platforms have messy payment needs, especially with big transactions or auctions. Wrappay works well here, handling mints (private, public, or Dutch auction) through smart contracts, keeping pricing and execution transparent.

For secondary sales, buyers and sellers can transact directly. Platform fees are handled automatically, with no extra steps for the user. Combined with cross-chain tools, there's potential for multi-network trading, if the rest of the stack allows.

C. Web3 Gaming Economies

Blockchain games constantly deal with in-game asset sales, upgrades, and digital land purchases. Wrappay lets develop- ers skip the hassle of building payment logic from scratch. In tournaments or prize pools, smart contracts handle entry fees and payouts, acting like an escrow and making sure everyone gets paid according to the outcome, no manual steps needed.

D. Decentralized Finance Integration

DeFi sites can use Wrappay for protocol fees or governance voting. Atomic transactions mean users don't face incom- plete or inconsistent outcomes. In cross-chain yield farming, Wrappay lets users interact through a single interface, even with multiple chains and token swaps in the background.

E. Enterprise and B2B Applications

For enterprise — think supply chain management or B2B marketplaces — Wrappay handles blockchain payments sim- ply, without building from scratch. International business transfers can benefit from the system's stablecoin support, reducing volatility and speeding settlement.

VII.CHALLENGES AND FUTURE DIRECTIONS

Wrappay addresses a lot in Web3 payments, but not every- thing's solved. Some limits are just realities of blockchain today, others come from design trade-offs made along the way.

A. Current Limitations

The biggest headache is Ethereum congestion. When the net- work gets crowded, fees shoot up and even small payments get pricey. Layer 2 solutions help, but they split up liquidity and bring their own technical hurdles.

Moving across blockchains isn't exactly smooth, either. People have to juggle assets between chains or use third-party bridges, which can be risky and definitely not seamless. True "just works" cross-network payments aren't here yet.

Regulation is another wild card. KYC and AML rules are different everywhere, and since Wrappay isn't built around compliance, what you can do depends on the local laws.

Smart contracts are basically permanent once live. That's great for safety, but makes quick updates tough. There are upgradeable contracts, but they add complexity and extra trust concerns.

B. Future Development Roadmap

First, more scaling — growing support for Layer 2s and bet- ter rollup integration to tackle high fees. Plus, Wrappay needs smarter ways for different layers to talk to each other.

Account abstraction (like ERC-4337) has a lot of promise. Imagine users skipping those tedious gas payments or ap- proving a whole set of actions at once — it could make things almost frictionless.

Cross-chain capabilities should get more seamless too. In the future, Wrappay could let someone pay on one chain, while a merchant gets paid on another — behind the scenes, all in one shot.

Getting started should be easier as well. At the moment, you need crypto to use Wrappay. Adding fiat on-ramps — letting people pay with cards or bank transfers — would help bring in more users.

There's room for new features, too. Instead of just single payments, Wrappay could handle

subscriptions, streaming, even escrow payments; all built on secure smart contracts.

Privacy tools are getting better. Zero-knowledge proofs could keep wallet and payment info hidden, while still keeping transactions valid.

Fraud prevention can get smarter, using behavioral analysis to spot suspicious activity and warn merchants, all while keeping user data private.

Governance could shift toward letting the community decide how things run — imagine token-based voting on new features, fees, or funding.

C. Research Opportunities

There's still plenty to dig into. Robust cross-chain payments that work between blockchains with different rules is a big one, and formal verification could help guarantee the system stays reliable.

User experience needs work, too. Interacting with wallets still confuses a lot of people. If Wrappay can make things simpler without losing security, adoption goes up.

VIII.CONCLUSION

Wrappay wants to make Web3 payments a lot less painful. The idea is to support as many chains as possible and stick to a non-custodial, open model. There's no middleman — smart contracts and on-chain code process every payment directly and transparently.

That unified interface is a real game changer: users with EVM or Solana wallets can all interact together, saving headaches for developers and end users. Atomic payments make sure transactions either go through or not at all — no getting stuck halfway. Integrating Wrappay shaves off time compared to building a payment flow from scratch.

Testing looks solid so far — transaction speeds match what you'd expect on each network, gas costs are trimmed, and people get faster, simpler checkouts.

Non-custodial design actually matters: Wrappay never touches users' funds, so there's no central risk. Open sourc- ing the code also helps build trust, since people can inspect everything.

That said, some stuff still stings — Ethereum congestion still bites, and regulatory problems won't just disappear. Those aren't Wrappay-specific, though.

Looking ahead, Wrappay is going full speed into better Layer 2 integration, stronger account abstraction, and true cross- chain payments. Expect new features like subscriptions, privacy tools, and more community governance, too.

In the end, Wrappay shows you really can build transparent, secure Web3 payments that don't make life harder for devel- opers or users. This isn't the finish line — it's just a firm start. More features, more networks, and a bigger focus on letting the community steer things are all on the way.

REFERENCES

[1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[2] R. C. Merkle, "Protocols for public key cryptosystems," in *Proc. 1980 Symp. Security and Privacy*, IEEE Computer Society, Apr. 1980, pp. 122–133.

[3] R. K. Sharma and A. Singh, "A Study on Multi-Chain Wallet Aggregation Protocols for Decentralized Applications," in *2022 IEEE Symp. Blockchain*, 2022, pp. 145–152.

[4] J. Chen, "Non-Custodial Smart Contract Architectures for On-Chain Payment Processing," *Journal of Decen- tralized Systems*, vol. 3, no. 1, pp. 45–58, 2021.

[5] A. Schmidt and M. Weber, "Analysis of Developer Tools in the Web3 Ecosystem: The Impact of Wagmi and Viem," *Web3 Foundation Research*, 2023. [Online].Available: https://web3.foundation/research

[6] Solana Foundation, "Solana Wallet Adapter: A Modular Interface for dApp-Wallet Connectivity," *Solana Foundation Reports*, 2022. [Online]. Available: https://solana.foundation

[7] D. Lee and H. Kim, "On-Chain vs. Off-Chain Payment Networks: A Comparative Study," in *2020 ACM Conf. Computer and Communications Security (CCS)*, 2020, pp. 1823–1838.

[8] S. Mahalle, P. Shinde, and S. Patil, "A Study on Blockchain Technology and Its Applications," in *2020 Fourth Int. Conf. Computing Methodologies and Communication (ICCMC)*, Erode, India, 2020, pp. 890–895.

[9] C. R. Adewuyi, A. Nwangele, A. Ajuwon, and A. O. Akintobi, "Advancements in Real-Time Payment Sys- tems: A Review of Blockchain and AI Integration for Financial Operations,"

Iconic Research and Engineer- ing Journals, vol. 4, no. 8, pp. 206–221, Feb. 2021.

[10] K. Shahrina, S. S. Meem, D. Sarker, and R. Hossain, "A Comprehensive Review and Analysis on Decentralized Payment Gateways," in *Proc. 2nd Int. Conf. Computing Advancements (ICCA '22)*, 2022, pp. 412–419.

[11] V. Zilnieks and I. Erins, "A Sample Model for Inte- grating Decentralized Payment Systems Through Com- mon Gateway for E-commerce," *WSEAS Transactions on Business and Economics*, vol. 20, pp. 2099–2109, 2023.

[12] V. Buterin, "Ethereum: A Next-Generation Smart Con- tract and Decentralized Application Platform," 2014. [Online]. Available: https://ethereum.org/whitepaper

[13] G. Wood, "Ethereum: A Secure Decentralised Gener- alised Transaction Ledger," *Ethereum Project Yellow Paper*, vol. 151, pp. 1–32, 2014.

[14] A. Yakovenko, "Solana: A New Architecture for a High Performance Blockchain v0.8.13," *Whitepaper*, 2018. [Online]. Available: https://solana.com/solana- whitepaper.pdf

[15] H. K. Alper and A. Ku¨pc¸u¨, "Optimally Efficient Multi- party Fair Exchange and Fair Secure Multi-party Com- putation," *ACM Transactions on Privacy and Security*.

[16] H. K. Alper and A. Ku¨pc¸u¨, "Coin-based Multi-party Fair Exchange," in *Proc. ACNS 2021*, 2021.

[17] R. Zhang, R. Xue, and L. Liu, "Security and Privacy on Blockchain," *ACM Computing Surveys*, vol. 52, no. 3,pp. 1–34, May 2021.

[18] F. Tschorsch and B. Scheuermann, "Bitcoin and Be- yond: A Technical Survey on Decentralized Digital Currencies," *IEEE Communications Surveys & Tutori- als*, vol. 18, no. 3, pp. 2084–2123, Third Quarter 2016.