HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

MASTER THESIS

Mr

Dominik Bepple

Trust-Minimized On-Chain Asset Purchases: An Investigation into the Integration of Online Payment Methods in Smart Contracts

Mittweida, August 2025

Faculty of **Applied Computer Sciences & Biosciences**

MASTER THESIS

Trust-Minimized On-Chain Asset Purchases: An Investigation into the Integration of Online Payment Methods in Smart Contracts

Author:

Dominik Bepple

Course of Study:

Blockchain & Distributed Ledger Technology

Seminar Group:

BC21w1-M

First Examiner:

Prof. Dr.-Ing. Andreas Ittner

Second Examiner:

Dipl.-Volkswirt Mario Oettler

Submission:

Mittweida, 10.08.2025

Bibliographic Description

Bepple, Dominik: Trust-Minimized On-Chain Asset Purchases: An Investigation into the Integration of Online Payment Methods in Smart Contracts, 60 pages, Mittweida, Hochschule Mittweida – University of Applied Sciences, Faculty of Applied Computer Sciences and Biosciences.

Master Thesis, 2025

Abstract

Within the realms of a growing cryptocurrency ecosystem, and its associated increase in user adoption, the demand for secure, transparent, and user-friendly fiat-to-crypto solutions has grown significantly. This work elaborates the implementation and design of trust-minimized bridges between the traditional online payment systems (in this case, PayPal), and Ethereum-based blockchain applications. The architecture supporting this bridge is based on PayPal's API, Chainlink Functions and smart contracts to introduce automated, verifiable settlement of fiat payments with minimal reliance on trusted intermediaries. By combining structured off-chain transaction data with on-chain escrow mechanisms, the system reduces counterparty risk while preserving usability. The developed prototypes demonstrate how fiat payments can securely trigger asset transfers on-chain, providing a practical model for integrating established payment rails into decentralized finance and contributing to the broader vision of a safer, more interoperable financial ecosystem.

Table of Contents

List of Figures

List of Tables

Listings

1 Introduction

The cryptocurrency landscape has experienced remarkable growth and rapid development in the past ten years, producing groundbreaking technologies and changing user behavior. By 2024, the global cryptocurrency market reached a value of around 4.57 billion, and forecasts for further growth indicate up to $ 13.19 billion by 2032 [1]. In parallel to this market expansion, the spread of cryptocurrencies has increased significantly, with over 560 million users holding digital assets worldwide [2]. This far-reaching acceptance has also re-shaped the user preferences and in particular promoted the demand for safe and private methods for the storage of digital assets.

A particularly outstanding trend within the cryptocurrency ecosystem is the departure from centralized stock exchanges towards private, self-custodial wallets. Providers such as MetaMask recorded considerable user growth and reported over 30 million monthly active users by 2024 . This change underlines a fundamental preference of users for direct control over their assets, driven by increasing concerns about data protection, security and autonomy. Despite this trend, an essential part of cryptocurrency purchases are still based on centralized financial infrastructures such as traditional bank channels, credit card networks and payment service providers (PSPs). These conventional payment systems naturally contain trustworthy intermediaries and thus bring potential weaknesses and dependencies with them that contradict the decentralized ethos of cryptocurrencies. [3]

This contrast between decentralized possession of digital assets and centralized payment infrastructures is a significant challenge. Users who want to participate in the cryptocurrency market must make complex trade-offs: they want the comfort and speed of familiar payment platforms, but at the same time want to minimize the trust and dependence on central actors. Therefore, there is an urgent need for innovative solutions that enable the safe, trust-minimized swap of Fiat currencies to digital assets on the blockchain.

This work is specifically dedicated to this challenge by examining how conventional online payment systems – in particular PayPal – can certainly interact with blockchain-based decentralized applications in order to reduce the dependency on trustworthy third parties. The central research question that leads this work is therefore:

"How can on-chain asset purchases be implemented securely and in a trust-minimized manner using consumer online payment systems?"

To answer this question, the work pursues a methodological approach in several steps. First of all, a comprehensive analysis of current technologies and strategies for fiat-to-crypto changes is carried out, with the strengths and weaknesses of custodial, hybrid, and decentralized payment solutions to be identified. Novel prototypes are then developed, which integrate PayPal's payment verification mechanisms with Ethereum Smart Contracts utilizing Chainlink Functions – a decentralized oracle system. This approach aims to enable safe, automated validation for off-chain transactions. It should trigger asset transfers directly on the blockchain, without manual interventions or excessive trust in individual intermediaries.

The results of this research are intended to show a practical approach to the safe connection of established Fiat payment networks with blockchain infrastructures. The developed prototypes will specially demonstrate how PayPal transactions can trigger on-chain assets transfers in Ethereum-based systems autonomously and reliably. By providing verifiable evidence via the off-chain payment compensation directly to a blockchain's smart contract, this solution significantly reduces the usual user trust assumptions and thus increases security, transparency and user-friendliness.

The work is divided into eight chapters that clearly structure the course of research. After the introduction of context and motivation in Chapter 1, Chapter 2 follows an overview of the state of the art with regard to centralized, hybrid and decentralized fiat-to-crypto payment solutions. Chapter 3 then lays the conceptual foundation by defining trust-minimized payment systems and identifying relevant interfaces between off-chain and on-chain systems. Chapter 4 describes the architectural design of the proposed system, followed by a gradual implementation in Chapter 5. In Chapter 6, the developed prototype is systematically evaluated, including detailed analyses on performance, reliability, security and trust-minimization. Chapter 7 discusses further considerations, including design implications, limitations, and potential improvements. Finally, Chapter 8 summarizes the most important findings, draws conclusions and provides an outlook on future developments and improvements.

In short: With the development and evaluation of this innovative fiat-to-crypto bridge solution, this work contributes to the further development cryptocurrency settlement and ultimately supports the overarching vision of a safer, more transparent and user-centered financial ecosystem.

2 State of the Art

The development of cryptocurrency exchanges illustrates an ongoing conflict of goals between the minimization of trust and user-friendliness and reflects the complexities with which users are confronted with the acquisition of digital assets. Originally, Bitcoin transactions required considerable trust and combined cryptographic security with a paradoxical trust in strangers. Satoshi Nakamotos WhitePaper from 2008 proposed a "peer-to-peer electronic cash system", which explicitly aimed to eliminate the need for trustworthy intermediaries through the use of cryptographic evidence instead of personal trust [4]. In the early days of Bitcoin, however, practical reality was far from this vision. The acquisition of Bitcoin was often complicated, risky and insecure and was heavily based on informal channels. Enthusiasts were often instructed on internet forums such as Bitcointalk to exchange goods or cash for Bitcoin. This process was jokingly described by an early user as "digital alchemy hits Craigslist-Roulette" – an expression for the unpredictable and risky nature of such early transactions. At the time, the safest method was, although extremely impractical to meet with strangers personally in order to exchange cash for cryptocurrency. This practice was humorously referred to in the community as "Irl Atomic Swaps" (In-Real-Life Atomic Swaps) – an ironic allusion to the inherent and the risk [5]. Participants had to have extensive technical knowledge, including creating and managing wallets, the verification of blockchain transactions, and at the same time spend a significant level of personal trust if they met with unknown places.

Between 2010 and 2013, efforts have been made to formalize these peer-to-peer transactions and make them more secure. There were many examples such as the LocalBitcoins platform, which introduced structured escrow services and reputation systems to reduce fraud. Nevertheless, these improvements continued to require users' trust in central escrow agents who acted as referees in disputes, often on the basis of subjective or incomplete information. At the same time, the introduction and growth of centralized stock exchanges (CEXs) such as Mt. Gox (active from 2011 to 2014) changed the ecosystem, since the user trust from individuals shifted to institutional platforms. Central exchanges enabled faster and more convenient trade by allowing users to hold fiat currencies and cryptocurrencies on accounts that were managed by third parties. However, this was accompanied by custody risks. Users exchanged their control over digital assets against transaction comfort, a vulnerability that was dramatic when Mt. Gox collapsed in 2014 and around 850,000 BTC was lost. This event made the fragility and the inherent risks of centralized custody clear. [6]

The period after 2015, the so-called Ethereum age, produced a transformative innovation: programmable trust through smart contracts. Smart contracts automated escrow processes and reduced human interventions and bias. Decentralized stock exchanges (DEXs), such as EtherDelta in 2016, made it possible to act directly from their wallets by using on-chain order books and thus strengthening user autonomy. However, previous DEX platforms suffered from problems such as low liquidity, high transaction latency and user-unfriendly interfaces. Only the introduction of Automated Market Makers (AMMS) such as Uniswap in 2018 improved the experience of decentralized exchanges significantly. The complexity was abstracted and transferred to more intuitive workflows, which increased user-friendliness. In parallel to these innovations, Fiat on-ramps, such as MoonPay, introduced in 2018, tried to

bridge the gap between traditional finance and cryptocurrencies. They integrated regulatory requirements such as “Know Your Customer” (KYC) and “Anti-Money Laundering” (AML) directly in wallets and decentralized applications (DApps). Although these developments lowered technical barriers, they also introduced new trust dependencies, especially with regard to centralized KYC providers and third-party price feeds. [7]

This historical development shows a central area of tension in the introduction of cryptocurrencies: minimizing trust is not an absolute goal, but exists on a spectrum that is determined by technical possibilities, regulatory framework and changing user expectations. In the following sections, this work systematically examines the current custody models and analyzes in detail how each model is balanced, operationalized, or compromised in order to achieve scalability and broad acceptance.

2.1 Centralized Custodianship: The Rise of Centralized Exchanges

Centralized exchanges like Coinbase and Binance act as critical bridges between conventional finance and cryptocurrency markets. These platforms emphasize ease of use, rapid execution, and deep liquidity, but they also require users to place significant trust in multiple layers of intermediaries. To begin, users fund their exchange accounts by depositing government currency through familiar channels such as bank transfers, credit card payments, or digital services like PayPal. Once deposited, that money is converted to digital tokens and stored in wallets that the exchange controls. Trading on these platforms occurs off-chain on proprietary order books managed by the exchange. Only when a user requests a withdrawal, the system does interact with the blockchain, broadcasting transactions to transfer assets from the exchange’s custody to the user’s own wallet.

This model depends on three core tiers of intermediary infrastructure. First, the on-ramp and off-ramp for fiat currency rely on legacy systems such as international bank settlement networks, credit card networks like Visa, and payment processors such as PayPal. These systems introduce the familiar delays, fees, and inefficiencies of traditional banking. Second, exchanges act as custodians by holding private keys in their own secure environments, effectively becoming digital vaults or banks. This arrangement runs counter to the original vision laid out by Bitcoin’s creator, who sought to eliminate reliance on any trusted third party. Third, liquidity is provided through algorithms and trading desks operated by the exchange itself. Such centralized liquidity management can obscure pricing, create hidden costs, and offer limited transparency into how spreads are determined. [8]

Regulatory compliance further reinforces centralization. Exchanges must implement robust identity verification and anti-money laundering controls under regimes such as the European Union’s Markets in Crypto Assets Regulation. These requirements force users to share personal documents and transaction histories, boosting institutional confidence but also creating concentrated stores of sensitive data. These repositories become prime targets for hacking, threatening financial assets and personal privacy. Many exchanges choose to register in jurisdictions with relatively light oversight, which can complicate legal recourse and asset recovery when things go wrong. [9]

From a technology standpoint, centralized platforms simplify blockchain complexity. They provide polished user interfaces and stable API endpoints that support algorithmic trading and portfolio management tools. Institutional traders and bots rely on standardized REST and

streaming interfaces to monitor prices and submit orders. However, these services often impose strict rate limits and may fail under extreme market conditions, exacerbating liquidity shortages. Withdrawal requests are frequently subjected to manual security checks, creating additional delays between trade execution and actual ownership of the assets.

Despite these trade offs, centralized exchanges remain the dominant venue for crypto trading because they offer the familiar features of traditional financial services. Integration with major payment networks enables near instantaneous fiat transactions, and advanced liquidity management attracts retail and professional investors with competitive pricing and streamlined experiences.

In the end, this reliance on convenience creates a custodial paradox. Users willingly relinquish the decentralized control promised by blockchain in exchange for smooth, accessible interfaces. That dynamic reinforces the very intermediaries that crypto was designed to circumvent, illustrating the ongoing struggle between reducing trust assumptions and delivering the seamless experiences needed for mainstream adoption.

2.2 Hybrid Custodianship: Fiat On-ramps and Embedded Gateways

Hybrid custodianship models have become a bridge between traditional finance and decentralized cryptocurrency ecosystems, offering users the convenience of familiar payment methods while preserving direct control over their digital tokens. These solutions, often known as fiat on-ramps, enable purchases of cryptocurrency using credit cards, bank transfers, or online payment services. Leading on-ramp providers such as MoonPay, Wyre and Transak integrate seamlessly into software wallets and decentralized applications, streamlining the process for new and experienced users alike.

The way fiat on-ramps operate involves close collaboration with banks and payment networks to authorize and settle government currency transactions, then immediately convert those funds into digital assets that are sent to users' self-custodial wallets. MoonPay, for example, provides a customizable widget that developers can embed in web or mobile applications via simple application programming interfaces and webhooks, along with software development kits for both front end and server side environments. This architecture lets individuals retain sole ownership of their private keys and removes many of the technical hurdles that once discouraged mainstream crypto adoption [10].

Yet this convenience comes at the cost of new trust dependencies. Users must place confidence in third-party on-ramp providers to handle their funds securely and to follow evolving regulatory requirements. To meet anti-fraud and legal obligations, these services enforce thorough identity verification and transaction monitoring processes, requiring customers to share personal data and transaction details. While these measures bolster the credibility of digital asset markets and attract institutional participation, they also reduce privacy and user autonomy, potentially deterring those who prioritize anonymity.

The widespread integration of fiat on-ramps into digital wallets and applications has undeniably expanded access to cryptocurrencies. Partnerships with prominent platforms such as OpenSea, MetaMask and Trust Wallet have allowed on-ramps to offer intuitive purchase experiences for NFTs and token swaps. Nonetheless, the reliance on centralized identity services and external price oracles injects fresh points of vulnerability into the ecosystem,

reintroducing single points of failure that stand in tension with blockchain's decentralization ethos [11].

A provider of on-ramp services continues to play a decisive position in relation to opening up such cryptocurrency markets. Since such entities harvest and archive sensitive user information and transaction records in single points of storage and consolidation, they represent company invitations to data theft and cyber attacks. Therefore, they have to adopt effective encryption algorithms, safe data-processing practices and initiate frequent security checks to safeguard the resources and personal data of clients.

Anti-money launder regulation, licensing and protectives of consumers remain heavily fragmented globally as jurisdiction applies different standards. To secure the seamless operation and transnational compliance, a persistent regulatory monitoring, a versatility in adapting processes, and a whole suite knowledge in legal matters is a must.
The more familiar payment experience and ease of use in the application of cryptocurrencies in the form of a hybrid custodianship approach, such as a fiat on-ramp, make cryptocurrencies less exclusive. However, they create new incarnations of centralization and trust assumptions which are bound to be handled. The ongoing issue in front of the developers, service providers and regulators is to find a right balance between ease of access, regulatory compliance, security and the decentralization principles the blockchain technology is built around.

2.3 Decentralized Custodianship: Peer-to-Peer (P2P) Platforms

Peer-to-peer (P2P) platforms represent the purest embodiment of decentralized custody within cryptocurrency transactions, since they enable direct trade between users without central intermediaries. Services such as BISQ, Hodlhodl and Agoradesk illustrate this model by combining cryptographic security measures with jointly controlled governance structures. This approach allows users to carry out fiat-to-crypto trading, while they keep complete control over their private keys, which fundamentally preserves the decentralized ethos, which was originally sought with blockchain technology.

The operating frame of P2P platforms is naturally non-protected. Users come into contact directly via decentralized networks in order to negotiate transaction conditions independently. A prominent example is BISQ, which uses a decentralized desktop application that is operated via the data protection-oriented Tor network in order to offer users anonymity and increased security. On BISQ, individuals can create purchase or sales offers with certain payment methods, such as banking or cash deposits, and react to them. Each transaction is secured by the use of multi-signature (multisig) ecrow smart contracts, which are typically structured in a 2-of-2 configuration. This means that funds are securely blocked and only released if both parties confirm the success of the trade. It is crucial that this design eliminates the risk of custody because user money is never kept by the platform itself. [12]

Platforms such as Hodlhodl further expand the decentralized custody through the use of atomic swap technology. Atomic swaps enable users to carry out cross-chain cryptocurrency shells, such as Bitcoin against Monero, without relying on escrow mediators. This mechanism uses cryptographic tools, so-called Hashed Timelock Contracts (HTLCs) to ensure that transactions are carried out atomic. Either both parties successfully complete the

trade according to the agreed conditions, or the transaction is completely undone, which excludes the possibility of partial or fraudulent fulfillment.

P2P platforms use several robust countermeasures to cope with potential fraud and security concerns that are inherent in decentralized interactions:

Reputation systems: Platforms such as Agoradesk integrate reputation-based models that help reduce information asymmetries between trading partners. Users collect trust points that are derived from previous successful transactions, transaction historia and peer reviews. This promotes transparency and well-founded decision-making among the participants. [13]

Decentralized arbitration: In the event of a dispute, conflict resolution works via decentralized arbitration mechanisms. Instead of central mediation, platforms such as BISQ rely on the collected arbitrators that store cryptocurrency as security in order to ensure impartial judgments, or on deterministic smart contracts that automatically execute predefined arbitration rules. [14]

Encrypted communication: Platforms consistently rely on end-to-end-encrypted communication channels and use the Tor network to protect the identity of the users and to maintain the confidentiality of transactions in close agreement with the pseudonymous character that is inherent in blockchain technology.

Despite their significant agreement with the principles of decentralization, P2P platforms are faced with several significant systemic challenges:

Liquidity fragmentation: In contrast to centralized stock exchanges, P2P networks typically do not have aggregated liquidity pools. This fragmentation often leads to greater tension between buying and sales offers, lower overall liquidity and slower trade execution times which potentially deteres users who value speed and comfort.

Regulatory ambiguity: The lack of comprehensive KYC and AML procedures on P2P platforms exposes these considerable regulatory risks and exams. Regulatory frameworks such as the European Union's Markets in Crypto-Assets Regulation (MiCA) classify these decentralized and non-custodial exchanges as “unregulated entities,” which complicates their compliance efforts and creates operational uncertainties.

Technological complexity: The model of decentralized safekeeping requires users to deal with technically complex tools such as multi-sig wallets, HTLCs and data protection oriented networks such as Tor. For many potential users, especially those without extensive technical knowledge, these requirements represent a large entry-level hurdle and thus limit the broad acceptance.

Although decentralized P2P platforms effectively maintain the original ideals of blockchain by enabling the minimized trust, direct fiat-to-crypto transactions and their broader acceptance is hindered by challenges in relation to liquidity, regulatory conformity and technical complexity. Coping with these problems remains crucial to enable wider use and increase the practicality and attractiveness of decentralized custody models.

3 Conceptual Foundation

The integration of the traditional systems of fiat payments with those blockchain inevitably implies a complex technical and regulatory challenge. Payment platforms like PayPal exist in centralized, reversible, and heavily regulated networks of money and Ethereum is built on deterministic, irreversible smart contract logic with no central figure. Merging the two areas thus involves a close reflection on the introduction of trust, and how to sustain it when needed and most preferably keep it as minimum as possible.

Despite the growth of self-custodial wallets, most cryptocurrency users continue using a pay mechanism that they are used to, off-chain payments, when purchasing digital assets. Centralized trading platforms and fiat gateways feature excellent user experiences and fast transactions, as covered in Chapter 2, but they require high levels of trust in third parties. Completely decentralized peer-to-peer networks, in their turn, do not depend on the third party as much but are less convenient to operate, slower, and more decentralized.

This chapter establishes the theoretical and architectural principles underlying the integration of centralized fiat payment systems with decentralized blockchain environments. It defines the notion of trust-minimization as it applies to hybrid payment models, identifies the unavoidable “irreducible trust interfaces” between off-chain and on-chain components, and examines how these trust boundaries can be constrained, verified, and made observable. The discussion proceeds by analyzing the operational constraints of mainstream online payment rails such as PayPal, Stripe, and card networks, the design of oracle systems for off-chain verification, and the architectural strategies that allow smart contracts to act as neutral, rule-bound execution agents. Together, these concepts form a structured framework for building systems that bridge traditional payment infrastructures with blockchain networks while reducing reliance on centralized intermediaries.

3.1 Trust in Hybrid Payment Systems

The combination of centralized payment providers and decentralized blockchain applications ones intuitively brings trust as a prominent element of the entire provision. Only DApps can use smart contract mechanisms to enforce their own rules, while hybrid applications must connect to external actors and services. Think of payment processors, backend servers, and oracle networks. All of these factors require trust.

After all, trust can never be completely eliminated. We may properly identify where it appears, handle it cautiously, and reduce its scope and ramifications. As an example, you might need to trust that a PSP correctly processed a transaction, that an oracle has recorded the right information, or that a backend is playing honestly. No single party should be too important to the system, otherwise it will become weak. [15]

That leads us to the primary design objective of this thesis: minimization of trust. Instead of assuming every actor is perfectly reliable, a trust-minimized setup limits what any individual

can do, makes sure crucial steps are verifiable, and sets up fallback mechanisms when things go wrong.

To achieve this, we need to understand how trust enters the system, what types of trust we really need, and which ones can be reduced or removed using cryptography or architectural techniques. This section discusses these aspects of trust in more detail.

3.1.1 What "Trust Minimization" Really Means

The term “trustless” tends to come up often when analyzing blockchain technologies in academic texts. However, it is usually misleading to refer to a system as trustless since in the real world, systems need to rely on faith, particularly when traditional financial systems such as PSPs are used. A better model is trust minimization: designing systems to require users to trust fewer components, and to trust the rest of them only in a limited and verifiable way. [16]

Consider our hybrid model. PSP users depend on the company to make payments accurately. The smart contract has faith in an oracle to report the success or failure of every payment. And the system itself relies on infrastructure like webhooks or off-chain servers to work normally.

A trust-minimized design deals with these points in three ways. First, it attempts to minimize the fallout in case of any component failure e.g. in case the oracle is unresponsive, the contract should timeout and refund the buyer. Second, it is targeted towards resilience: when a backend restarts, it should be able to resume without requiring privileged information. Finally, it keeps control tight: no single party should be able to move funds without some on-chain validation.

3.1.2 Irreducible Trust Interfaces

The ideal of a perfectly decentralized system is not likely to occur, since any interaction with an outside service will always require some degree of trust. These unavoidable points are usually called irreducible trust interfaces, and they mark the areas where decentralized logic must depend on external input, behavior, or infrastructure that can’t be verified on-chain. [17]

Imagine a fiat-to-crypto bridge that uses a payment service provider (PSP) and a smart contract capable blockchain. Here the three key interfaces of trust are:

Payment Processing Trust

The bridge has to have faith that the payment service provider is doing the right thing with transactions, applying its own rules, and giving accurate updates through its API and webhook system. There is no cryptographic proof on-chain that a PSP transaction occurred or that funds were held or captured. This should be provided by PSP itself, directly or through an intermediate oracle. Although a PSP could be a very established actor, the system has to assume that its APIs will act consistently and that payments will not be arbitrarily reversed or frozen without any reason. Such trust is unavoidable but can be limited in time and scope e.g. by not releasing cryptocurrency until payment confirmation is received, or by canceling pending transactions after timeout.

Oracle Trust

Smart contracts cannot directly access outside data and thus have to use oracles to report external information. Oracles add a trust layer even despite multiple nodes and economic incentives: the system has to be able to assume that most of the nodes involved will act honestly and that the oracle runtime will run the desired logic. Although oracles may provide cryptographic attestations and economic incentives to behave honestly, they still depend on the correct configuration, safe run-time environments, and correct access to external APIs. [18]

Backend Trust

Although crypto architecture aspires to push every possible computational task on-chain, real life doesn't work that way. Creating payment requests, handling webhook events, and coordinating oracle calls all require off-chain interaction. Backend systems involved in these processes inevitably gain some degree of influence over the workflow, even if they are designed to be stateless or restricted in authority. Even the most minimal version still temporarily stores sensitive data like authentication tokens, communicates with PSP APIs, and initiates oracle requests. We can make the backend such that it cannot settle funds unilaterally, but we cannot get rid of trust altogether. [8]

These interfaces are not implementation flaws, they are structural realities. The challenge now is to diagram out how trust flows across them, to constrain the consequences of any failure, and ideally to make the consequences visible to all who are watching the system.

3.2 Payment Rails: Constraints and Realities

The most widely used mechanisms for initiating fiat payments, such as PayPal, card payment processors like Stripe, and the global card networks operated by Mastercard and Visa, originated primarily to serve e-commerce transactions. Optimized for convenience, buyer protection, and fraud prevention in online retail, these systems were not designed with decentralized settlement in mind. As a result, their technical architecture and underlying assumptions present significant challenges for direct integration with blockchain infrastructure.

All of these payment rails have a similar flow: the user initiates the transaction using a card or digital wallet, the processor requests authorization with the issuer, and the final settlement is immediate or takes several days. Since the necessary information to confirm the exchange is not easily available in a blockchain context, the pressure points of reversibility and dispute resolution, which are the characteristics that make fiat systems consumer-friendly, are incompatible with the irreversibility of smart contracts.

3.2.1 Online Payment Rails in Scope

Online payments have a variety of on-ramp protocols, but three have the broadest global reach, technical maturity, and accessibility for development: PayPal, Stripe, and direct credit card networks. Each provides standardized APIs to support extensive monitoring capabilities, thus allowing quick integration with existing software environments. However, they differ in their applicability to blockchain-enabled applications in terms of the quantity of information they provide, the temporal relevance of this information, and the extent to which proofs of payment can be checked independently within decentralized infrastructures.

PayPal

PayPal forms one of the most widely used transactional provider services in the world with a mature REST API which allows the merchants to programmatically set up various order flows. Merchants can take every payment lifecycle event under control, e.g. determining when to move the process to completion or issue refunds or cancellations and the merchants can annotate every transaction with meaningful, structured data. Major fields that are first-class include *custom_id, invoice_id,* and *reference_id*, with the usage of extensible JSON payloads carrying application context between initialisation and settlement.
To be observable and auditable, PayPal offers consistent identifiers on all payment events and broadcasts on webhooks with 2-part certificate based signatures that can be verified by third parties (e.g. oracle nodes, independent auditors), without sharing any shared secrets. A workflow-like sandbox also allows safe end-to-end testing of the whole workflow. Until payment is finalized after requiring buyer protection or card-issuer rules, payments can be disputed or charged back, as happens with other fiat rails, thus deterministic finality is not assured. Taken altogether, the verifiable webhooks, excellent intent/metadata fields and widespread user base makes PayPal a solid reference rail in this respect as trust-minimized fiat-to-crypto bridges are concerned. [19]

Stripe

Stripe provides a unified developer experience across cards, bank transfers, and digital wallets via a comprehensive API and a state-machine model (e.g., *PaymentIntent*) that lets developers coordinate when a transaction should be completed or reversed. Like PayPal, Stripe supports attaching rich metadata to tie fiat payments to off-chain or on-chain context, offers consistent event identifiers, and provides an extensive test mode for end-to-end prototyping. Webhook notifications are signed and include replay-protection features. However, verification relies on symmetric endpoint secrets, which means authenticity can only be proven by parties that possess the shared secret – less convenient for independent, third-party verification or decentralized oracle attestations. Stripe inherits the same finality limitations as card networks (e.g., chargebacks that can extend for months), so irreversible on-chain actions must still be gated. Overall, Stripe is fully capable as a fiat on-ramp and offers excellent tooling and global coverage, but its webhook verification model makes externalized, trust-minimized verification more involved than with PayPal's certificate-backed approach. [20]

Credit Card Networks (Visa, Mastercard)

Visa and Mastercard constitute the underlying card rails that define authorization, clearing, and settlement between issuers and acquirers; most developers access these networks indirectly through PSPs/processors (e.g., PayPal, Stripe, Adyen, Checkout.com). At the network level, settlement is batch-cleared and subject to post-transaction dispute and chargeback regimes, so payment "completion" is not equivalent to cryptographic finality. Direct developer interfaces from the networks are limited. Standard, developer-friendly constructs such as JSON payloads with first-class custom fields, public webhook signatures, and testable end-to-end flows are typically provided by the processor layer rather than by the networks themselves. Consequently, from a trust-minimization standpoint, card networks used directly offer limited on-chain-friendly observability or verifiability. They are best integrated through a PSP that supplies structured APIs, metadata fields, webhook attestations, and sandbox environments. [21]

Since every rail of payments has different constraints and trust assumptions, choosing an integration partner directly affects the architecture and security posture of a system in general. PayPal offers the most visible, provable pattern of off-chain event verification. Its orders API has structured, first-class fields that can be cross-referenced to on-chain identifiers and its webhooks are signed with public keys backed by certificates and thus can be independently verified without establishing shared secrets. Comparatively, Stripe and direct card-network integrations create more opaque trust boundaries. PayPal benefits from unmatched global adoption, quick setup, a stable and well-documented API, and a sandbox environment that mirrors production, making it both accessible for rapid prototyping and rigorous enough for evaluating the limits of a trust-minimized fiat-to-crypto bridge.

Challenges of Fiat Verification

The task of constructing a secure fiat-to-crypto bridge involves engineering a mechanism that enables on-chain smart contracts to respond to fiat transactions while minimizing reliance on off-chain actors. The inherent challenge is that none of the large online payment systems were made to be verified by decentralized, tamper-proof computation. These systems offer rich APIs and extensive transaction metadata, but were designed to be used by centralized servers, not by distributed validation agents in the public.

The core challenge can be described as a verification gap, the disconnect between how fiat payments are settled and our ability to audit those settlements in a cryptographically secure, decentralized manner [22]. This gap is due to the fact that a blockchain smart contract is necessarily restricted in the visibility of events occurring outside the blockchain. First, it cannot transmit HTTPS requests to a payment provider's API, precluding direct access to off-chain transaction data. Second, it cannot validate off-chain messages such as webhook notifications unless those messages are signed using a verifiable public key infrastructure, a practice that many payment providers do not support. Lastly, it cannot tell the finality of a fiat payment. An API response may show a transaction as being completed, but in fact be reversible many days (or even months) later, because of dispute mechanisms or chargebacks. These constraints create a trust boundary that must be managed with care in any hybrid fiat-to-crypto system.

For fiat-denominated payments to be amenable to on-chain settlement, three stipulations must be met: stability, provability, and timeliness. Stability requires that information issued will

always be the same and unchangeable after it is issued; the transaction identifier, e.g. must always refer to the same payment event and cannot be revoked later. Provability requires that the origin of this data, payment service providers like PayPal or Stripe, be verified with methods other than just HTTPS, so that it can be verified by smart contracts or decentralized oracles. Lastly, timeliness means that the data must be received before a predetermined expiration limit in order that escrowed funds can be released or refunded in a predictable fashion.

Standard online payment rails oppose a uniform compliance record: PayPal secures transaction identifiers through API stability and post-capture queryability, yet lacks definitive on-chain verifiability. Stripe, despite providing quick event notifications through webhooks, allows revising the status of transactions within 120 days, effectively undermining finality. Even worse problems are seen in Visa and Mastercard: their clearing process obscures finality, real-time webhook access is not available, and official APIs give little more than black box responses.

Consequently, on-chain logic remains insufficiently equipped to transact fiat without relying on a trusted off-chain entity, be it a centralized back-end server or a decentralized oracle network, to report and verify fiat-denominated events.

This dependency underscores a fundamental challenge: although blockchains are purpose-built for on-chain validation and secured execution, the off-chain infrastructure essential for converting fiat transactions into consensus-friendly data constitutes a mandatory intermediary, and the chain of trust it introduces raises serious questions about resilience, auditability, and attacks.

3.3 Oracle Design and Off-Chain Verification

Blockchain oracle systems constitute one of the foundational challenges governing the interface between deterministic on-chain logic and indeterministic off-chain events. Smart contracts are mathematically precise, globally consistent across remotely distributed nodes, but they are necessarily closed to external systems like payment processors, Web APIs and banking systems. This isolation necessitates the reliance on trusted intermediaries capable of retrieving, authenticating, and relaying off-chain data in a secure and verifiable manner.

In a trust-minimized architecture, the oracle functions not merely as a data source but as a security boundary that can compromise the decentralized design of a blockchain when poorly engineered [23]. On the other hand, well-designed oracle systems may maintain the security properties of a blockchain when connecting to mutable and revocable real-world services, such as PayPal, Stripe, and Visa.

3.3.1 Oracle Architectures

Oracle designs have evolved to meet varying requirements around latency, cost, decentralization, and security assurances. The table below compares five leading architectural models that have emerged in recent years, each with different trade-offs.

Table 1: Oracle Systems Comparison [24]

Oracle Type	Trust Model	Decentralization	Latency	Cost	Security Mechanism	Use Case Fit
Centralized Oracles (e.g., Oraclize)	Single trusted entity	None	Low (seconds)	Low	TLS notary proofs, TEE attestation	Prototyping, internal apps
Threshold Networks (e.g., Chainlink)	Economic consensus	High (multi-node)	Medium (minutes)	Medium	Quorum signatures, staking & slashing	DeFi, cross-chain bridges, payments
Oracle Chains (e.g., Pyth, Band)	Blockchain consensus	Medium (validators)	Low (seconds)	Medium	PoS consensus, validator slashing	High-frequency data, price feeds
Enshrined Oracles (e.g., Cosmos)	Validator-layer integration	Inherits from L1	Very low (blocks)	Low	Native slashing, validator reputation	Native blockchain state feeds
Hybrid Models (e.g., Chainlink Functions)	TEE + Economic consensus	High	Medium	High	SGX enclaves, quorum, encrypted secrets	Fiat verification, custom computation

Threshold Attestations and Cryptographic Guarantees

In decentralized oracle systems like Chainlink, trust is no longer based on trusting a single data provider but is instead based on a shared consensus of economically-incentivized actors. Threshold-based attestation schemes require a predetermined quorum of oracle nodes to retrieve, verify, and sign a data item, for example, a PayPal capture event, before the outcome is incorporated on-chain. Such an arrangement significantly reduces the likelihood of collusion or silent failure.

The verification protocol is identical across all nodes, and is usually run in a trusted execution environment (TEE) like Intel SGX or AMD SEV. TEEs separate the execution code and inputs, even in case of system compromise. Once executed, the isolated environment generates a cryptographically signed payload which is then combined into an attestation certificate that the smart contract can verify without communicating with the individual nodes. [25]

Emerging Techniques: ZK and Privacy-Aware Verification

The investigation into oracle mechanisms has progressed beyond TEEs toward zero-knowledge proofs (ZKPs) that permit off-chain computation while preserving both privacy and verifiability. ZKPs allow an oracle to prove that it has correctly followed a particular logic, e.g., to show that a PSP response contains a valid capture ID, without sharing raw data or internal computational process information. Although they are cognitively complex, the techniques provide strong privacy guarantees and are particularly useful when sensitive information like user identity or payment credentials are being processed. [24]

Economic Incentives and Behavioral Alignment

The security of oracles in the long term depends not only on cryptographic schemes, but also on the community of stakeholders on which the system is based. A multi-layered incentive architecture that combines, e.g., Chainlink and other threshold oracle networks:

- **staking:** nodes put tokens as collateral, and can be slashed in case of dishonest behavior;
- **slashing:** nodes that provide wrong or tampered information are slashed by losing proportionate amounts of their stakes, thus discouraging malicious actions.
- **reputation:** nodes that perform well have access to more requests, and nodes that perform poorly are demoted, creating a market incentive to be accurate and available. [26]

Taken together these mechanisms form a robust incentive structure that rewards honest participation and penalizes deviation, thus removing the need to have centralised control.

3.3.2 Why Chainlink Functions?

This thesis adopts Chainlink Functions as its reference oracle architecture for verifying fiat payments. While other models offer valuable features, such as ultra-low latency or native blockchain integration, Chainlink Functions uniquely balances the requirements of security, programmability, privacy, and auditability. Specifically:

- It supports secure HTTP calls to external APIs like PayPal, even for endpoints requiring OAuth2 credentials.
- Code is executed in SGX-secured enclaves, ensuring data integrity and isolation.
- Results are cryptographically signed and traceable on-chain, allowing for post-hoc auditing.
- Secrets and API keys are stored securely using DON-managed encrypted secret stores.
- The oracle layer acts purely as an attestation system, so never holding funds or triggering settlements unilaterally. [27]

These capabilities make Chainlink Functions especially well-suited for the hybrid trust model explored in this thesis, where off-chain verification is required, but control over on-chain state must remain transparent and deterministic.

3.4 Design a Trust-Minimized System

The preceding sections have shown that fiat payment systems and public blockchains operate under fundamentally different assumptions. Fiat rails prioritize flexibility, reversibility, and consumer protection. Blockchains prioritize determinism, finality, and censorship resistance. Any attempt to connect these two domains must navigate a space where assumptions collide: payments can be reversed days after confirmation, APIs return unsigned JSON, and transaction success may depend on private keys stored inside centralized infrastructure. Within this gap lies the core problem: how to allow smart contracts to respond to fiat-based events, without introducing opaque trust dependencies or handing control to centralized actors. While no architecture can eliminate trust completely, it is possible to minimize and structure trust in such a way that all parties know what is trusted, when it is trusted, and what happens if trust fails.
This section defines the conceptual framework used throughout the rest of the thesis to evaluate architectural choices. It articulates the design principles that guide system behavior, particularly in how off-chain data is consumed, how time and reversibility are handled, and how authority is distributed between backend, oracle, and contract layers.

Bounded Trust: Time and Scope Over Authority

The concept of bounded trust is the basis of trust minimization. In hybrid systems, some entities, e.g. a backend server making a PayPal payment or an oracle confirming a PSP capture, must be trusted, but their power must be tightly limited in terms of both scope and time.
Take as an example an oracle that is involved in the process of deciding whether a PayPal payment has been completed. This decision should be followed by a clear deadline; the contract should not be waiting indefinitely to be confirmed. Likewise, when a backend is used to provide metadata to initiate a transaction, the metadata has to be converted to immutable form when passed to the contract. In both cases, the power of the actor is limited by a foreseeable, accountable frontier.
These restrictions prevent any party backend, oracle or user, to unilaterally slow down or hijack the transaction. Every trust assumption thus comes with a fail-safe: timeout, fallback path, or deterministic default. [26]

Verifiability as a Substitute for Trust

Trust-minimized system architecture is predicated on the observation that claims can be positioned for verification instead of reliance. Whenever off-chain entities submit inputs to on-chain contracts – for instance, by asserting that "payment X has been captured" – such inputs must be accompanied by evidence that can be independently validated.

In the case of oracles, this is usually fulfilled by cryptographic signatures, quorum attestations, or enclave-generated proofs. As an example, a Chainlink Function may provide a signed response that a PayPal payment status is "*COMPLETED*". The contract does not depend on the integrity of the backend processor or its internal logic. Instead it verifies that the signature correlates with a predefined public key and that the outcome conforms to the contract's own on-chain expectations.

This principle outsmarts webhook-based architectures where data come without substantiation or where verification is dependent on proprietary backend processes. Rather, it

encourages reproducible assertions, verifiable oracles and deterministic contract verification. If any off-chain input cannot be confirmed by independent verification mechanisms, it should not precipitate irreversible state transitions on the blockchain. [11]

Escrow as an Execution Gatekeeper

In the absence of instant payment finality, blockchain-based transfer systems require an intermediary holding mechanism that protects both the sender and receiver while off-chain verification is pending. Escrow contracts offer impartial execution platforms where money is only released upon fulfillment of the set conditions.
Escrow, though, is more than a financial cushion, it acts as a rational referee. A smart contract encodes the transaction rules:

1. the off-chain event that must occur (e.g., PayPal capture),
2. the evidence required to substantiate it (e.g., a signed oracle result), and
3. the time limit within which this evidence must appear.

In case all the conditions are met prior to the deadline, the money is transferred to the seller, otherwise, the buyer is refunded.
This model provides that the users do not have to trust each other and that both parties cannot be left in limbo. The contract acts as a rule-bound, neutral judge that automates resolution and enforces fairness by transparently programmed logic. [28]

Minimizing Backend Authority

The centralization hazard of giving a lot of backend power, to perform event verification, issue trust decisions, or even to control the execution of contracts, can be especially sharp in the context of early-stage architectures. Such arrangements can be suitable in the rapid prototyping but not in production systems. Consequently, a trust-minimized design must confine backend participation to that of a passive coordinator or relay. The backend can still trigger fiat payment or send metadata to the oracle, but should not make the final decision on whether the transaction was successful or not. This decision must lie with verifiable entities: oracles that can bear witness to the witnessed events, and smart contracts that can reason about the oracles proofs. [29]

Another need is the limitation of backend control once the execution is in progress. After the user funds have been committed or a transaction initiated, the backend must not be able to override, cancel or otherwise disrupt the ongoing process so that control is not in the hands of an individual participant but in the system.

Towards a Structured Model of Hybrid Trust

Combining these principles results in a system where trust becomes explicit, time-bounded, and externally verifiable. Off-chain actions are constrained by smart contract logic. Verification occurs via reproducible oracle attestations. Escrow enforces fairness without discretion. And the backend, while necessary for fiat coordination, is excluded from decision-making authority.
This structured model of hybrid trust does not eliminate risk – it acknowledges it, scopes it, and wraps it in a framework where failure modes are predictable and recoverable. It also enables architectural experimentation: different system variants can shift the trust boundary between backend, oracle, and contract layers while still adhering to the same design principles.

In the next chapter, this framework is operationalized across three progressively decentralized system designs. Each makes a different trade-off between backend simplicity, oracle complexity, and contract autonomy. Together, they demonstrate how architectural choices can express different trust models without ever breaking the core guarantees laid out here.

4 Architecture

This chapter explains the architecture of the designed trust-minimized fiat-to-crypto bridge. It outlines a three-layered system that includes off-chain services, on-chain smart contracts and interfaces that connect between them. The discussion makes it clear how Chainlink Functions, API of PayPal and Ethereum work together in a synergistic manner to ensure automated, verifiable and secure settlement. The constituent components are shown by illustrative diagrams which show the data flows related to the components, and hence form an accurate blueprint to be executed further.

4.1 Ethereum Layer

Ethereum functions as the foundational settlement and execution layer within this hybrid fiat to crypto payment system, providing a public, tamper resistant environment where business logic is encoded as smart contracts and execution is guaranteed by distributed consensus rather than any centralized authority. When users interact with the system, they sign transactions from their wallets that either transfer or escrow Ether into smart contracts, which only release funds upon external verification such as confirmed PayPal settlement and can automatically trigger refunds or expiration timeouts without requiring human intervention.

Deterministic Execution and Consensus Properties

A fundamental characteristic of Ethereum is its deterministic execution model. Every transaction must be processed identically by all participating nodes, which independently execute contract code and must arrive at the same state to maintain consensus. This deterministic requirement means smart contracts cannot make arbitrary web calls or access private data sources directly; they have no native capability to query external payment providers like PayPal. Instead, external facts must be introduced through specialized services called oracles, establishing a clear separation between on chain logic and off chain data sources.

The Ethereum architecture distinguishes between externally owned accounts (EOAs) controlled by users through private keys and contract accounts that contain executable code and can hold assets but act only when invoked. In the payment flow, a user's cryptographically signed transaction expresses intent to exchange fiat for cryptocurrency, while the escrow contract holds Ether in a locked state until oracle attestation confirms that the corresponding fiat transaction has cleared successfully. [29]

Probabilistic Finality and Block Confirmation

Finality on Ethereum operates on a probabilistic basis rather than offering immediate certainty. Transactions become increasingly irreversible with each subsequent block that builds upon them, following Ethereum's Gasper consensus mechanism which typically

requires 64 to 95 slots for full finalization under normal network conditions. Off chain components such as backend services, monitoring dashboards, and audit systems therefore wait for multiple block confirmations before treating events as final, thereby avoiding race conditions that could occur during potential blockchain reorganizations.

Research on probabilistic finality indicates that this model, while introducing latency, provides robust security guarantees against various attack vectors. Studies show that the probability of successful chain reorganization decreases exponentially with block depth, making transactions with sufficient confirmations practically immutable even without absolute finality. [28]

Event Logs and State Monitoring

Ethereum's event logging system creates a crucial bridge between on chain state changes and off chain services. When smart contracts register orders, validate payments, or release funds, they emit structured event logs that are cryptographically committed to the blockchain state. These logs are immutable, efficiently queryable through bloom filters, and require no contract code re execution, making them ideal for bookkeeping, regulatory compliance, and dispute resolution. [29]

From an integration perspective, the off chain backend maintains a minimal interaction surface with the blockchain. It monitors escrow deposits for mining confirmation, queries contract storage to determine order status, and subscribes to event logs that announce registration or payout events. Because the smart contract itself enforces all business rules autonomously, the backend remains stateless and non custodial, never making unilateral decisions about fund release but simply reacting to deterministic on chain signals.

Layer 2 Compatibility and Scaling Solutions

Although the prototypical testing of the proposed contracts was done on the Sepolia testnet to enable cost-efficient development, the contracts can be deployed on the Ethereum mainnet or any EVM-compatible Layer 2 solution, e.g., Arbitrum, Optimism, and Polygon, with minimal code changes. Such networks maintain the execution semantics and event models defined by Ethereum but improve the transaction throughput and lower the cost. Such compatibility ensures that deterministic execution, public auditability, and the clear separation between on-chain logic and off-chain observation remain intact across diverse deployment environments.

Layer 2 solutions, in particular, can substantially lower the costs associated with smaller transactions without compromising the trust-minimized properties of the underlying Ethereum settlement layer. The modular design allows merchants to choose their favored execution venue based on cost, speed, and security trade-offs without requiring any fundamental changes to the escrow logic and oracle integration. [30]

By combining deterministic execution, immutable event logs, probabilistic finality with robust security guarantees, and a narrow interface for off-chain observers, Ethereum provides the cryptographically secure foundation upon which this hybrid payment architecture operates. The blockchain allows users to trust predetermined rules to be enforced without human involvement, and allows developers and auditors to verify all state transitions by examining permanently recorded, verifiable data structures.

4.2 PayPal integration

PayPal serves as the primary fiat payment rail that interfaces with the on-chain escrow mechanisms within this trust-minimized architecture. The platform's settlement workflow is strategically structured around three distinct, reversible transaction phases that enable controlled fund management while preserving the option for automated rollback mechanisms. This three-stage process comprises

1. authorization, which establishes a temporary hold on funds without immediate transfer;
2. capture, which converts the authorized hold into an irreversible transaction upon blockchain confirmation, and
3. void, which releases the hold in cases of system failure or user cancellation. [19]

Authorization and Capture Workflow

The authorization and capture model used by PayPal is the main way of aligning the finality of settlement expectations of the traditional fiat payments system with the probabilistic nature of settlement in blockchain payments.

In the authorization stage, PayPal places a time hold on the payment device of the buyer on a fixed period, typically up to 29-days on most kinds of transactions, without paying the merchant account. This interim constraint functions as a critical synchronization point linking the probabilistic finality of blockchain events with the traditional payment settlement cycle, thereby enabling PayPal to reconcile the temporal mismatch between liabilities established on-chain and those assumed off-chain.

The capture process proceeds only after the banking infrastructure receives cryptographic confirmation that the corresponding on-chain escrow has successfully processed the buyer’s cryptocurrency deposit and an oracle-assured attestation that indicates successful settlement. This ordering means that fiat money can never be irreversibly moved until blockchain settlement has been conclusively ascertained, and thus prevents custodial overlap between traditional and digital payment systems. Conversely, the mechanism of void offers an automated rollback feature that immediately releases authorized funds in the event that blockchain settlement conditions fail to be met within the pre-determined period of time.

REST API Integration and Authentication

The payment infrastructure of PayPal can be reached only through its standardized REST API endpoints, which are all secured with the OAuth2 authentication protocol. When a purchase request comes in, the backend service will generate an order object, update the intent parameter to the value AUTHORIZE and send a richer JSON response containing all the metadata necessary to use at the time of verification and settlement. The response includes valuable information such as the unique order ID, the authorization status, purchase unit information, and action links to approve the buyer workflows.

```
1 {
2   "id": "8AC05046EH509023B",
3   "intent": "AUTHORIZE",
```

```
  "status": "PAYER_ACTION_REQUIRED",
  "purchase_units": [{
    "reference_id": "0x34cc7d...d368",        //e.g secretHash
    "amount": { "currency_code": "USD", "value": "10.00" },
    "custom_id": "0xAAe422F15BC254e64c49",  //e.g buyerAddress
    "description": "Fiat-to-crypto on-ramp"
  }],
  "links": [{
      "rel": "self",
      "href":
"https://api.sandbox.paypal.com/v2/checkout/orders/8AC05046EH509023B",
      "method": "GET"
    },{
      "rel": "payer-action",
      "href":
"https://www.sandbox.paypal.com/checkoutnow?token=8AC05046EH509023B",
      "method": "GET"
    }]
  }
```

Listing 1: Example Paypal Order Payload (JSON)

The OAuth2 implementation of PayPal adheres to the security standards of the industry, where client credentials are to be stored securely and access tokens refreshed according to the security requirements of PayPal [31]. This approach provides reasonable authentication levels and facilitates automated processing of transactions without intervention of human beings in routine activities. The API design supports stateless communications, with every request including all the context it needs, and minimizing the need to maintain a persistent session state and increasing system resilience.

Webhook Security and Event Processing

The webhook capability of PayPal is an asynchronous notification service that aligns fiat settlement procedures with blockchain processing logic in real-time. Notifications are transmitted via HTTPS, ensuring confidentiality while being augmented with mechanisms that guarantee message authenticity and resistance to unauthorized access. The verification process is a critical security measure, as it prevents adversarial entities from relaying spoofed messages that could result in unauthorized blockchain transactions or the externalization of system compromise. Scholarly literature on payment system security underscores the pivotal role of webhook verification in mitigating the risk of man-in-the-middle attacks and in maintaining a secure communication channel between merchant platforms and payment service providers. [32]. In the PayPal system, webhooks enable quick processing of only the most notable event, namely ORDER.APPROVED, which indicates successful buyer authorization. Any other payment state transitions (voids, disputes, and chargebacks) are managed via PayPal proprietary interface and are isolated to the automated cryptocurrency delivery process.

Sandbox Testing Environment

Both development and integration testing are supported by PayPal in a dedicated sandbox environment. A complete replica of production functionality is simulated, which is functionally equivalent to the production system, but uses simulated payment instruments and

test accounts. The environment can perform all the payment functions, i.e., order creation, buyer approval workflows, webhook delivery, authorization management, and settlement processing, without involving live monetary transactions. This architecture enables full end-to-end testing, with ngrok tunnels used to provide local development webhook endpoints, so developers can test full payment flows, including external webhook notifications, during the development process. This kind of functionality is essential to the synchronization of payment authorization timings and blockchain confirmation requirements; the interaction of these systems involves complex, asynchronous processes that must be strictly validated before they can be deployed to the production environment. [19]

Data Persistence and Recovery

The system maintains transactional state resiliency by using a hybrid architecture that combines local database storage with direct calls to the PayPal API. In deployments where persistent local infrastructure is used, critical metadata, namely PayPal order identifiers, authorization information, and timestamped processing information, is persisted to support fast recovery and recovery of processing after system interruptions. In the absence of local storage, the application, instead, checks the order status and fetches the necessary transaction data through the PayPal REST API, using order identifiers already stored on-chain. This API-based validation mechanism provides a full history of transactions and current status, which allows the system to rebuild the entire payment state and make the right next steps without relying on locally stored data.

The architecture allows the reliable operation in different deployment environments, as it supports two operational modes, one of which utilizes local storage, and another one that solely uses the PayPal API, which allows the capacity to recover in diverse failure scenarios. In case the backend service goes offline in the middle of processing transactions, the recovery system can automatically identify incomplete orders by querying the database where possible or systematically polling the PayPal API, and thus avoid losing transactions or having them remain in limbo indefinitely.

4.3 Chainlink

The isolation of blockchain networks has been introduced by design, to protect the integrity of the consensus and security properties. Such architectural design imposes a fundamental constraint: smart contracts are unable, by definition, to directly retrieve off-chain data sources, including payment confirmations, market prices, or sensor readings. Bridging this divide between deterministic on-chain execution and external data requires specialized infrastructure generally termed oracles. In this regard, Chainlink Decentralized Oracle Network (DON) forms the critical middle-ground between the payment verification system of PayPal and Ethereum smart contracts.

Chainlink Functions Architecture

Chainlink Functions constitutes a serverless computing framework that permits smart contracts to execute arbitrary JavaScript code within a trust-minimized environment. The framework functions on the basis of request-and-response: consumer contracts send computation requests with source code and parameters to the Chainlink network. The decentralized network has each oracle node running the given code in isolated runtime

environments, making external API calls, and processing the received data according to the logic provided.

The architecture has three main layers of security that makes it different to centralized oracle solutions. One is that the decentralized node network does not have single points of failure because computation is distributed among various independent operators. Second, the off-chain reporting protocol aggregates responses from multiple nodes using cryptographic consensus mechanisms, ensuring that no single node can manipulate the final result. Third, the system maintains complete audit trails of all requests and responses on-chain, enabling transparent verification of oracle behavior and data integrity. [33]

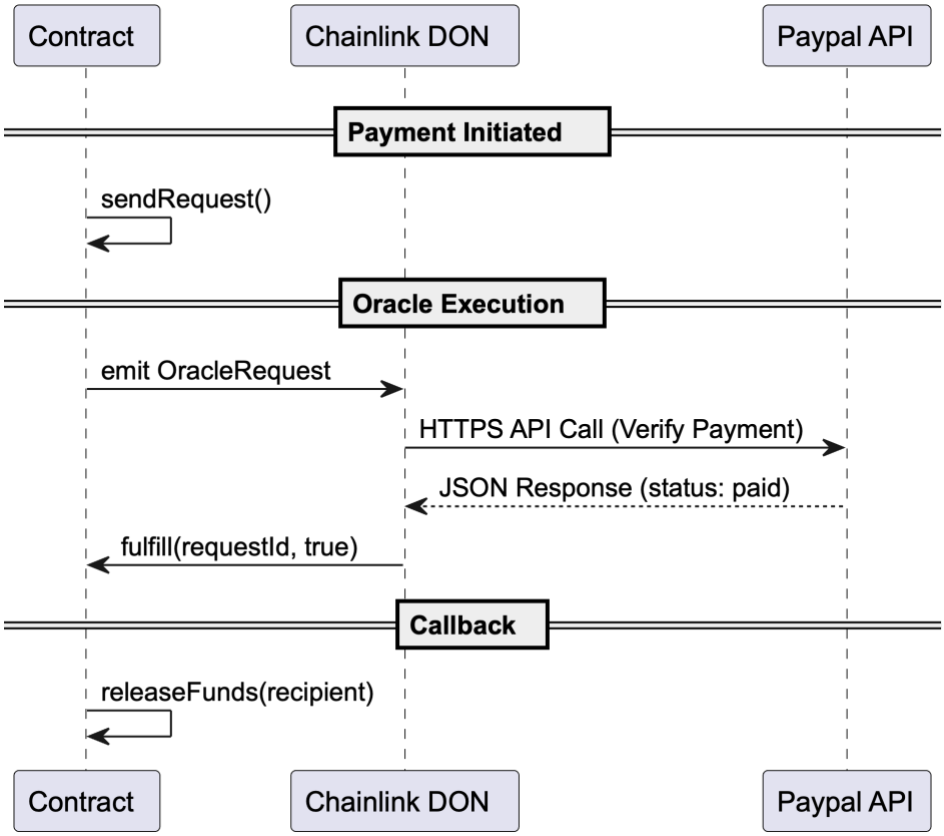

Figure 1: Chainlink Functions Example Sequence Flow

Trusted Execution Environments and Verification

Chainlink Functions use Intel SGX enclaves, in particular Trusted Execution Environments (TEEs), to isolate code execution and protect sensitive data against unauthorized access, even in case the host operating system is compromised. TEE-enabled nodes enable more security guarantees than software-based trust models by establishing a perimeter of hardware around the critical operations, and they maintain confidentiality of internal states and external communications.

In the payment verification process, TEEs provide security that oracles cannot leak OAuth credentials or API response data to third parties. Nodes continue to perform the verification logic, but the resulting attestations are publicly verified using cryptographic verification, so that the oracle network can perform any necessary verification tasks without needing to trust individual operators.

Decentralized Consensus and Aggregation

The consensus mechanism of Chainlink Functions is not limited to binary majority voting and uses advanced aggregation methods that consider node reputation, response validity, and time consistency. Scholarly research on decentralized oracle networks shows that threshold-based consensus mechanisms can provide security guarantees similar to the underlying blockchain and be resistant to collusion and data manipulation [26]. The response of each individual node is both locally and globally validated by the process of aggregation in the network, and cryptographic signature verification is used to ensure authenticity and outlier-detection algorithms are used to detect suspect or malfunctioning nodes. Only the responses that pass quality thresholds predetermined by the user are used in the final aggregated result sent to the requesting smart contract.

Secrets Management and API Integration

Commercial payment application programming interfaces (APIs) require authenticated communication, which is based on sensitive credentials and the integrity of which must be maintained during oracle execution. Chainlink Functions addresses this challenge with an advanced secrets-management infrastructure that combines threshold encryption and decentralized key management. The API credentials are encrypted using the public key of the network and then broadcasted to oracle nodes in pieces, thus no node has the full credential set at any one time.

This architecture can accommodate network-hosted and self-hosted secret-storage plans. In the case of PayPal integrations, OAuth tokens and client credentials are encrypted on the client-side and then posted to the secure storage infrastructure of the network. Immediately before execution, oracle nodes will only decrypt the parts of credential data that are needed to perform their particular verification.

Attestation Scope and Trust Boundaries

Limiting the scope of the oracle network to only attestation and verification means that direct asset loss is never possible in the case of an oracle failure or compromise. The system never houses user funds or possesses unilateral settlement authority: instead, it produces cryptographically verifiable evidence of off-chain payment status that is received by on-chain smart contracts, which then execute predetermined logic in response. This division of duties defines the boundaries of trust in such a way that the oracle network is responsible to ensure the accuracy and availability of data, and the smart contract layer continues to have custody and settlement capability [13]. In situations where oracle latency or unavailability exceeds the configured timeouts, the smart contract can automatically trigger refund functions or invalidate authorized payments, such that buyers and sellers do not become permanently locked out of their money as a result of oracle network latency/unavailability.

4.4 Prototypes

4.4.1 Backend-Centric One-Shot Settlement (Prototype A)

Prototype A deliberately starts from the most conventional position: a webserver remains in charge of the entire life cycle while Ethereum acts only as an escrow vault and final arbiter. A purchase begins when the buyer posts *amount* and *ethAddress* to the backend. The server:

1. creates a PayPal order in *AUTHORIZE* mode
2. checks that the escrowing contract holds at least the Ether equivalent
3. stores the resulting metadata in a lightweight SQLite table with order Id, hashed order Id, amount, buyer address, and an initial *pending* status.

This database is the sole off-chain source-of-truth for in-flight trades; if the process crashes, the server can reconstruct state by rereading those rows.
After the buyer approves the transaction in the PayPal pop-up, PayPal fires the CHECKOUT.ORDER.APPROVED webhook. The backend immediately calls PayPal's */authorize* endpoint, placing the fiat amount in a reversible 29-day hold (see Section 4.2). As soon as the *authorized* status is confirmed the server assembles a Chainlink Functions request: two string arguments (order ID, buyer address) and a short JavaScript snippet that will ask the backend itself for a signed receipt. The payload is compressed with CBOR and pushed on-chain via *sendChainlinkRequest*. From that moment the escrow contract waits for a callback – the standard Chainlink sequence described in Section 4.3.
While the oracle network is busy, the backend changes the local row to *requested* and records the transaction hash for audit. When the DON returns, the contract decodes the tuple *(buyer, cents, order ID)*, re-computes the ETH amount with *convertUsdToEth*, and, provided escrow is sufficient, transfers the funds to the buyer. It then *emits PaymentFulfilled(orderHash, …)*. The backend, subscribed to that event, captures the PayPal authorization and marks the row *fulfilled*. If the DON reports a mismatch or never replies before a timeout, the server voids the authorization instead, leaving the escrow untouched.

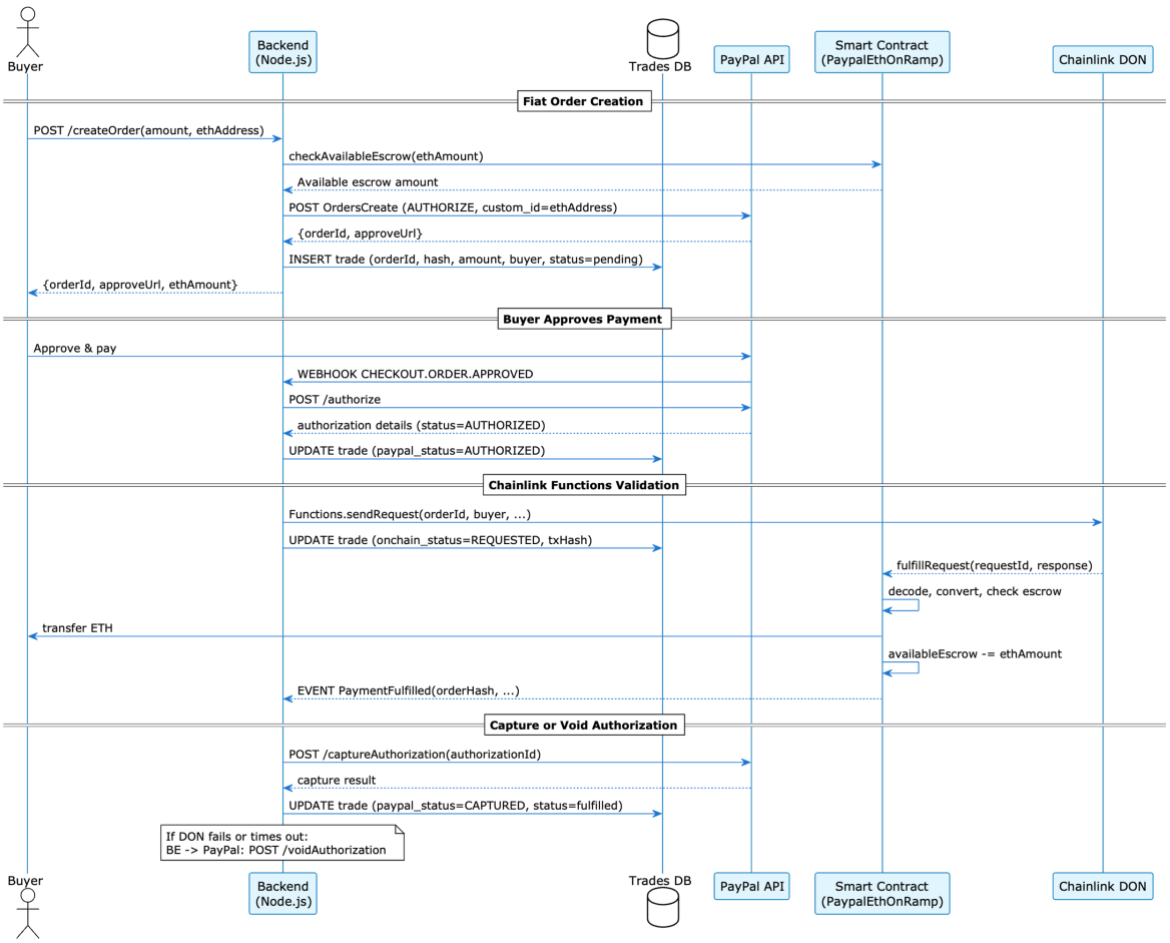

Figure 2: Backend-Centric One-Shot Settlement (Prototype A) Sequence Flow

By design the smart contract in Prototype A keeps almost no pre-settlement state. Only a per-order record is saved that prevents double-spending so the backend shoulders operational complexity. This arrangement still satisfies the thesis' trust-minimization goal: ETH is never released without an oracle-attested fiat payment, and every successful payout is visible, immutable, and self-verifiable on the blockchain. The trade-off is centralization of orchestration and recovery logic, a responsibility progressively pushed on-chain in the next two prototypes.

4.4.2 On-Chain Registration (Prototype B)

Prototype B keeps the authorize-capture mechanic of Section 4.2 but moves the order ledger itself onto the blockchain. Right after it calls PayPal's */authorize* endpoint, the backend executes *registerOrder(...)*, passing the PayPal ID (hashed for constant-size storage), the buyer's payout address and the USD amount. The contract records these values, reserves the Ether that will be needed for settlement, and emits *OrderRegistered*. From that instant every party
– including a restarted backend – can query the canonical state directly in contract storage or watch the event stream; no private database is required.

The next step is fiat verification. Instead of funnelling the check through the server as in Prototype A, the backend now asks Chainlink Functions to talk to PayPal directly. It mints a short-lived OAuth token, uploads it to the DON (decentralized oracle network) as a DON-hosted secret, and calls *sendChainlinkRequest*. The oracle script written in plain JavaScript fetches */v2/checkout/orders/:id*, confirms the order is *COMPLETED*, locates the active authorization, checks that the dollar amount and *custom_id (buyer address)* match the on-chain record, and finally ABI-encodes the tuple *(buyer, cents, orderId)* that the contract expects (cf. Figure 1).
When the callback arrives, the contract compares every field against the values it stored at registration and, if they line up, transfers the reserved Ether to the buyer and emits *PaymentFulfilled*. Only then does the backend retrieve the PayPal authorization ID (a lightweight GET) and call */capture*, irreversibly crediting the merchant. If the oracle never answers or returns a mismatch before the 29-day hold lapses, the backend voids the authorization instead. No automated retries are attempted for our prototypes.

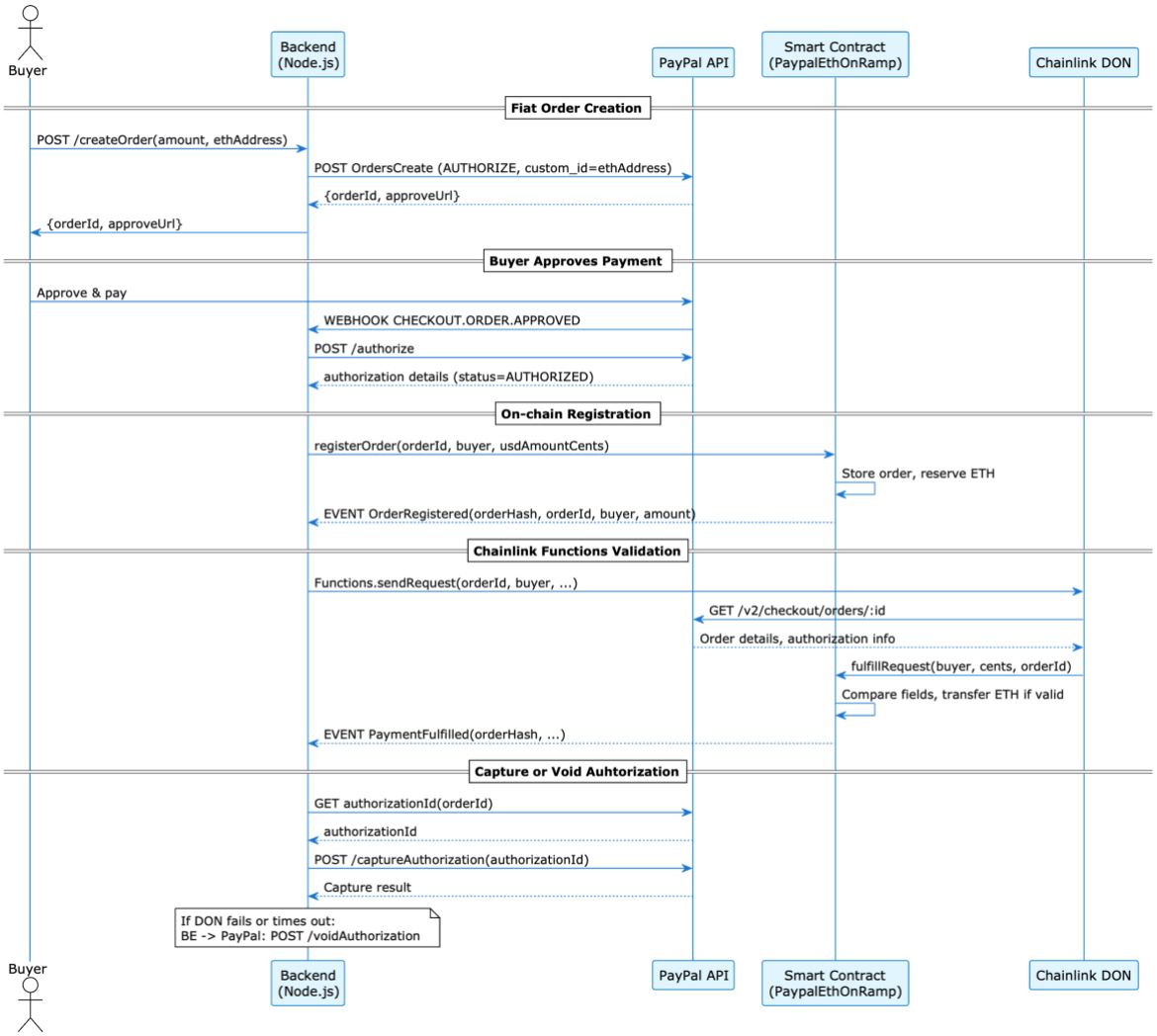

Figure 3: On-Chain Registration (Prototype B) Sequence Flow

Because the blockchain is now the single source of truth for pending orders, the server becomes stateless orchestration glue. A crash or deployment restart merely pauses progress until it reconnects, re-subscribes to *OrderRegistered* and *PaymentFulfilled*, and continues processing. All financial risk remains bounded: Ether cannot move without a valid oracle

signature, and fiat cannot settle without an on-chain success event. The result is a workflow that is visibly more decentralized and self-auditable than Prototype A while still relying on the same reversible PayPal authorize-capture flow introduced earlier.

4.4.3 Commit–Reveal Scheme (Prototype C)

Prototype C takes the decentralization trajectory one step further by handing the final act of settlement to the buyer. Instead of trusting the backend to push a "payout" transaction once the oracle says yes, the contract pays only when the user proves knowledge of a secret that was committed earlier in the flow. Everything else – the PayPal hold (Section 4.2), the oracle bridge (Section 4.3), the registration event model just introduced with Prototype B – remains in place; what changes is who can unlock the escrow and the reservation of funds.
The sequence begins with the buyer generating a random 32-byte value locally and hashing it with Keccak-256. That hash, together with the payout address, is sent to the backend's */create* endpoint. When the backend manufactures the PayPal *AUTHORIZE* order it places the address in *custom_id* and the hash in *reference_id*, preserving them verbatim inside PayPal's data model. As soon as PayPal fires the *CHECKOUT.ORDER.APPROVED* webhook the server authorizes the order and calls *registerOrder* on-chain, writing four immutable facts to storage: the PayPal ID (orderId), the buyer's address, the USD amount and the secret hash. The contract immediately deducts the corresponding Ether from *availableEscrow* and emits *OrderRegistered*.
From here the backend's role is already smaller than in Prototype B. It still generates an OAuth token and asks Chainlink Functions to check "is this order *COMPLETED*, does the authorization exist, do amount and hash match?" If the oracle network answers yes, the contract flips the order's status from *Registered* to *Validated* and emits a second event, but critically it pays nothing yet.
The buyer now has a one-hour (configurable) window to claim. A simple *claimFunds(orderId, secret)* call re-computes *keccak256(secret)* inside the EVM and verifies three things: the hash matches the commitment, the caller is the recorded address, and the deadline has not lapsed. On success the contract transfers the reserved Ether and logs *OrderClaimed*. Should the deadline pass unclaimed, anyone (typically the backend) may execute *expireOrder*, returning the Ether to the pool and letting the server void the still-reversible PayPal authorization.

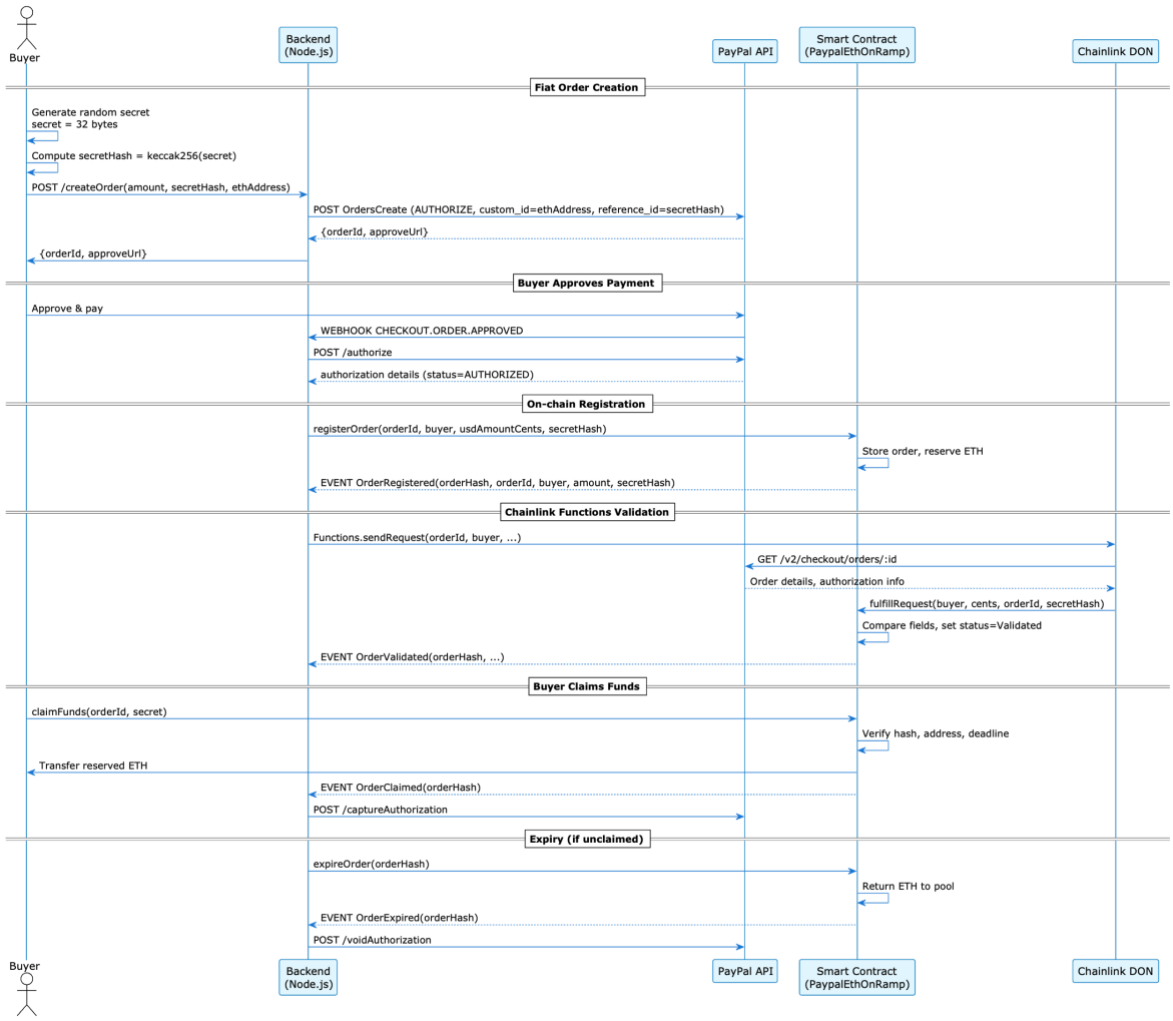

Figure 4: Commit-Reveal Scheme (Protoype C) Sequence Flow

All payout authority therefore rests with either a legitimate claimant or an explicit expiry path; the backend can neither accelerate nor block settlement. Front-running is effectively impossible because an attacker would need both the secret and control of the registered address, while the oracle retains its narrow attest-only duty. What emerges is a flow in which custody, validation and execution are fully disentangled, giving Prototype C the strongest trust-minimized posture of the three designs without departing from the standard PayPal authorize-capture model.

5 Implementation

In this chapter, the operationalization of the architecture as described in Chapter 4 is explained. The implementation includes three main parts:

- Solidity contract,
- off-chain backend, and
- the integration of the Chainlink oracle.

Payloads, request flows and event triggers are shown to demonstrate how the prototypes process an end-to-end fiat-to-crypto transaction.

5.1 Shared infrastructure and tooling

All three prototypes run on a common scaffold that keeps contract development, backend orchestration and oracle interaction uniform, so that only the business logic changes from A to C. Smart contract work is done in Solidity with the Hardhat toolchain. Hardhat's local EVM and plug-and-play testing framework allow contracts to be fuzzed and unit-tested before a single wei touches a public chain. Deployments target Sepolia, a lightweight Ethereum test network that is already integrated with Chainlink's Functions router and price feeds, making it the path of least resistance for oracle-heavy experiments. Core security primitives like *ReentrancyGuard* to block recursive calls and simple *Ownable* modifiers for administrative exits come straight from the OpenZeppelin library so they do not have to be reinvented. [34]

Every contract needs a dollar price to decide how much Ether must be set aside for a PayPal purchase. Instead of letting the backend inject that number, the contracts pull it from Chainlink's canonical *ETH/USD* price feed and convert *USD cents* → *wei* on-chain. Using the same oracle family for price and for PayPal verification keeps all external data under one vendor and one set of security guarantees. [35]

The backend is a small Node.js/Express service. Its responsibilities are pragmatic rather than stateful: create an *AUTHORIZE* order through the PayPal REST API, listen for PayPal webhooks, relay significant events to the contract (e.g., *registerOrder* in Prototypes B/C) and, once the chain is certain, fire the *capture* or *void* call. During local development the server runs behind ngrok, which turns *localhost* into a public HTTPS endpoint so that PayPal's Sandbox can post webhooks without the developer hosting a real TLS site. [36]

Chainlink Functions provides the missing link between those two worlds. In Prototype A the DON merely calls back to the backend for a signed receipt, so no secret material is required. Prototypes B and C, however, have the oracle query PayPal directly; the OAuth bearer token needed for that request is uploaded to Chainlink's encrypted secret store just before each validation run. Figure 5 sketches the sequence: the backend fetches a short-lived token from PayPal, pushes it to the DON as *slot n, version m*, and then invokes *sendChainlinkRequest(...)*, passing only the slot reference on-chain. Nodes can read the secret inside their secure enclave; outsiders – including the contract – see nothing but an opaque identifier. To keep it simple, backend is uploading a new token for every order.

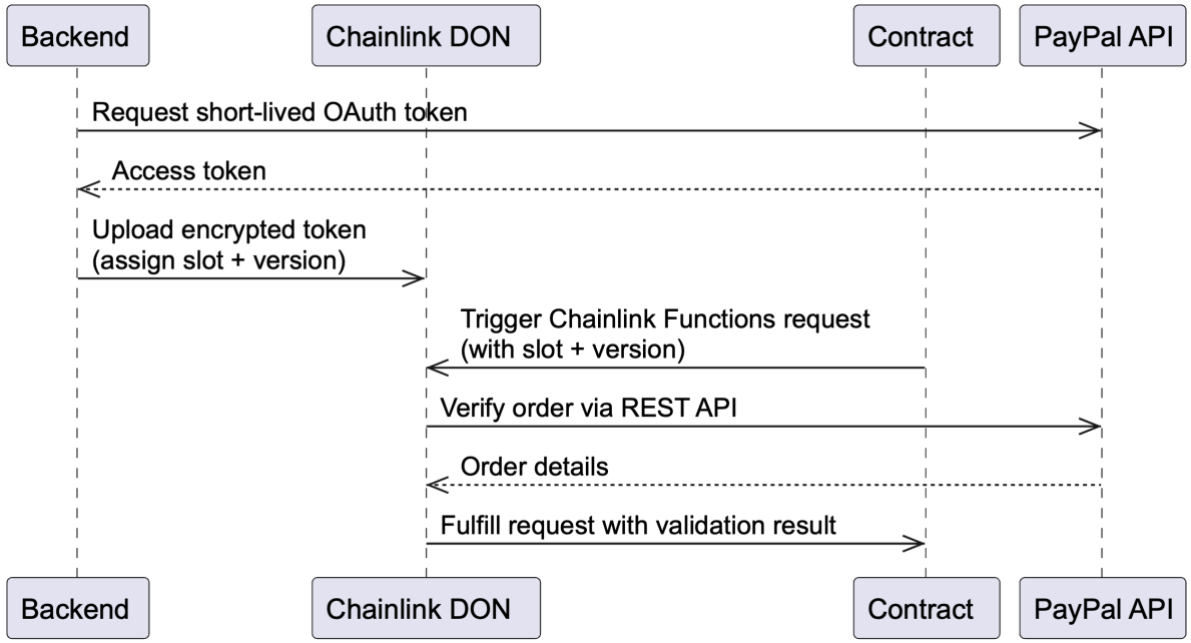

Figure 5: Uploading DON-Secret Example Sequence Flow

With Hardhat + Sepolia, Express + ngrok and Chainlink Functions + hosted secrets in place, each prototype can concentrate on how orders are recorded, validated and released rather than on plumbing. Changing the trust model therefore means touching a handful of Solidity functions and a single JavaScript validation script, not rebuilding the stack from scratch.

5.2 Smart Contract Implementation

Every prototype is built around a single, strict Solidity escrow that knows only how to accept Ether, earmark it, and release it under conditions verified elsewhere. The contract never sees PayPal credentials. It simply trusts two inputs:

1. Chainlink's ETH/USD feed for deterministic price conversion, and
2. a byte-encoded verdict from the oracle layer (Sec. 4.3).

By confining complexity off-chain, the on-chain surface remains small, auditable, and gas-efficient. The pool is funded through *depositEscrow()*, which refuses zero-value transfers and emits *EscrowDeposited* so indexers can monitor liquidity. Surplus funds – i.e., wei *not* reserved for open orders – can be withdrawn by the owner via a CEI-compliant, non-reentrant *withdraw()*.

```
function depositEscrow() external payable {
    require(msg.value > 0, "No ETH sent");
    availableEscrow += msg.value;
    emit EscrowDeposited(msg.sender, msg.value);
}

function withdraw(uint256 amount) external onlyOwner nonReentrant {
    uint256 w = amount == 0 ? availableEscrow : amount;
    require(w > 0 && w <= availableEscrow, "Invalid amount");
```

```
10      availableEscrow -= w;
11      (bool ok,) = payable(owner()).call{value: w}("");
12      require(ok, "Transfer failed");
13      emit EscrowWithdrawn(owner(), w);
14  }
```

Listing 2: Escrow Management

Price conversion is delegated to Chainlink's eight-decimal ETH/USD feed (noted in Sec. 4.3). Using cents as the input unit avoids floating-point ambiguity, and the helper is reused verbatim by all prototypes:

```
1 function convertUsdToEth(uint256 usdCents) public view returns (uint256) {
2     (, int256 p,,,) = priceFeed.latestRoundData();           // 8 deci-mals
3     uint256 usd18 = (usdCents * 1e18) / 100;                 // cents → 18 dec
4     return (usd18 * 1e8) / uint256(p);                       // feed → wei
5 }
```

Listing 3: Price Conversion USD to ETH

A read-only *getOrder(bytes32)* accessor exposes the stored *Order* struct so backend services or anyone with an RPC endpoint can query state without scanning logs:

```
1 function getOrder(bytes32 orderHash) external view returns (Order memory) {
2     return orders[orderHash];
3 }
```

Listing 4: Get Order

The most important commonly used function is the request dispatcher for Chainlink Functions, responsible for assembling the call payload and submitting it to the Decentralized Oracle Network (DON). In Prototype A, the DON-hosted secrets parameters (*donHostedSecretsSlotID, donHostedSecretsVersion*) are unused and explicitly passed as *0* or *null*, since no encrypted secrets are stored on the network. This reduces complexity and makes the request purely inline-JavaScript driven. The function initializes the request, attaches arguments, and submits it with a fixed subscription ID and gas limit:

```
 1 function sendChainlinkRequest(
 2         string memory source,
 3         bytes memory encryptedSecretsUrls,
 4         uint8 donHostedSecretsSlotID,
 5         uint64 donHostedSecretsVersion,
 6         string[] memory args,
 7         bytes[] memory bytesArgs,
 8         uint64 subscriptionId,
 9         uint32 gasLimit,
10         bytes32 donID
11     ) external onlyOwner returns (bytes32 requestId) {
12         FunctionsRequest.Request memory req;
13         req.initializeRequestForInlineJavaScript(source);
14
15         if (encryptedSecretsUrls.length > 0)
```

```
            req.addSecretsReference(encryptedSecretsUrls);
        else if (donHostedSecretsVersion > 0) {
            req.addDONHostedSecrets(
                donHostedSecretsSlotID,
                donHostedSecretsVersion
            );
        }

        if (args.length > 0) req.setArgs(args);
        if (bytesArgs.length > 0) req.setBytesArgs(bytesArgs);

        requestId = _sendRequest(
            req.encodeCBOR(),
            subscriptionId,
            gasLimit,
            donID
        );

        emit ChainlinkRequestSent(requestId);
        return requestId;
    }
```

Listing 5: Send API Request through DON

With these primitives in place, each prototype only adds the logic that differentiates its trust model:

Prototype A: a single *fulfillRequest()* that releases funds immediately after a valid oracle callback.

Prototype B: an up-front *registerOrder()*, plus the same *fulfillRequest()* to settle once the DON's response matches the on-chain record.

Prototype C: a *registerOrder()* which reserves funds, an additional *claimFunds(secret)* for the buyer-driven reveal, and an *expireOrder()* that re-credits escrow when the deadline passes.

All other behaviour like price feeds, re-entrancy guards, and access control come straight from this shared skeleton and the imported OpenZeppelin libraries, keeping the contracts aligned, testable, and easy to differentiate during audits. Now moving to the specific implementations.

Prototype A

The contract stores one record per PayPal order keyed by the keccak-256 hash of the PayPal ID. A second mapping guards against replayed oracle callbacks. Once a request ID is marked in *processed*, any duplicate delivery is rejected before it can reach business logic.

```
struct Order {
    string   orderId;
    address  buyer;
    uint256  ethAmount;
    bool     fulfilled;
```

```
6 }
7
8 mapping(bytes32 => Order) public orders;
9 mapping(bytes32 => bool)  public processed;
```

Listing 6: Data Layout (Prototype A)

The *fulfillRequest* function is invoked exactly once per order directly by Chainlink's DON. The decoded tuple is trusted only after three checks:

1. the callback must be new,
2. the order must not have been settled before, and
3. the escrow pool must still contain sufficient Ether.

If all hold, the Ether amount derived from the shared *convertUsdToEth* helper is transferred atomically and the ledger updated. Any failure throws, leaving PayPal's authorization free to be voided off-chain.

```
1 function fulfillRequest(
2     bytes32 requestId,
3     bytes   calldata resp,
4     bytes   calldata
5 ) internal override nonReentrant {
6     require(!processed[requestId], "duplicate");
7     processed[requestId] = true;
8
9     (address buyer,uint256 usd,string memory id) =
10        abi.decode(resp,(address,uint256,string));
11
12    bytes32 h = keccak256(bytes(id));
13    require(!orders[h].fulfilled, "paid");
14
15    uint256 ethAmt = convertUsdToEth(usd);
16    require(availableEscrow >= ethAmt, "insufficient-escrow");
17
18    availableEscrow -= ethAmt;
19    (bool ok,) = payable(buyer).call{value: ethAmt}();
20    require(ok, "transfer-failed");
21
22    orders[h] = Order(id,buyer,ethAmt,true);
23    emit PaymentFulfilled(h,buyer,usd,ethAmt);
24 }
```

Listing 7: Oracle Fullfillment (Prototype A)

Prototype B

Immediately after PayPal authorizes the fiat leg, the backend writes a canonical copy of the order on-chain. The function only checks that the pool can cover the forthcoming payout; it does not debit the pool yet. This keeps liquidity flexible until the oracle has the final word.

```
1 enum Status { None, Registered, Fulfilled }
2
3 struct Order {
4     string   orderId;
5     address  buyer;
6     uint256  usdCents;
```

```
    uint256  ethAmount;
    Status   state;
}

function registerOrder(
    string calldata id,
    address buyer,
    uint256 usdCents
) external onlyOwner {
    bytes32 h = keccak256(bytes(id));
    require(orders[h].state == Status.None, "exists");

    uint256 ethAmt = convertUsdToEth(usdCents);
    require(availableEscrow >= ethAmt, "insufficient-escrow");

    orders[h] = Order(id,buyer,usdCents,ethAmt,Status.Registered);
    emit OrderRegistered(h,id,buyer,usdCents,ethAmt);
}
```

Listing 8: Order Registration (Prototype B)

The callback must mirror the previously registered facts. Any discrepancy, be it a wrong buyer, mismatched amount or a lost registration, halts execution. Once the conditions align, Ether leaves escrow and the state switches to *Fulfilled*, allowing the backend to capture the PayPal authorization with confidence that the crypto leg is complete.

```
function fulfillRequest(
    bytes32 reqId,
    bytes   calldata resp,
    bytes   calldata
) internal override nonReentrant {
    require(!processed[reqId], "duplicate");
    processed[reqId] = true;

    (address buyer,uint256 usd,string memory id) =
        abi.decode(resp,(address,uint256,string));

    bytes32 h = keccak256(bytes(id));
    Order storage o = orders[h];

    require(o.state == Status.Registered, "unknown");
    require(o.buyer == buyer && o.usdCents == usd, "mismatch");
    require(availableEscrow >= o.ethAmount, "insufficient-escrow");

    availableEscrow -= o.ethAmount;
    o.state = Status.Fulfilled;

    (bool ok,) = payable(buyer).call{value: o.ethAmount}();
    require(ok, "transfer-failed");

    emit PaymentFulfilled(h,id,buyer,usd,o.ethAmount);
}
```

Listing 9: Oracle Fullfillment (Prototype B)

Prototype C

Unlike Prototype B, the Ether is reserved up front. The call debits availableEscrow and guaranteeing funds for the buyer the moment the oracle marks the order as valid. The pre-image hash ties PayPal metadata to the buyer's yet-to-be-revealed secret.

```
enum Status { None, Registered, Validated, Claimed, Expired }

struct Order {
    string   orderId;
    uint256  usdCents;
    uint256  ethAmount;
    bytes32  secretHash;
    address  buyer;
    uint256  deadline;
    Status   state;
}

function registerOrder(
    string calldata id,
    uint256 usdCents,
    bytes32 secretHash,
    address buyer
) external onlyOwner {
    bytes32 h = keccak256(bytes(id));
    require(orders[h].state == Status.None, "exists");

    uint256 ethAmt = convertUsdToEth(usdCents);
    require(availableEscrow >= ethAmt, "insufficient-escrow");

    availableEscrow -= ethAmt;                                  // hard reserve
    orders[h] = Order(
        id, usdCents, ethAmt, secretHash, buyer,
        block.timestamp + 1h,
        Status.Registered
    );
    emit OrderRegistered(h,id,usdCents,secretHash,buyer);
}
```

Listing 10: Order Registration (Prototype C)

The DON now confirms the fiat leg and the embedded secret hash. No Ether moves. Instead, the order progresses to *Validated*, signalling the buyer that the claim window has opened

```
function fulfillRequest(
    bytes32 reqId,
    bytes   calldata resp,
    bytes   calldata
) internal override nonReentrant {
    require(!processed[reqId],"duplicate");
    processed[reqId] = true;

    (uint256 usd,bytes32 sh,string memory id) =
        abi.decode(resp,(uint256,bytes32,string));

    bytes32 h = keccak256(bytes(id));
```

```
13     Order storage o = orders[h];
14
15     require(o.state == Status.Registered           &&
16             o.usdCents == usd && o.secretHash == sh, "mismatch");
17
18     o.state = Status.Validated;
19     emit OrderValidated(h);
20 }
```

Listing 11: Oracle Fullfillment (Prototype C)

The buyer's address and the secret pre-image must both match for a successful claim, otherwise the call reverts. If the buyer does nothing, the owner (or a keeper) can recycle the reservation after the one-hour horizon by calling *expireOrder*, restoring liquidity without manual book-keeping.

```
 1 function claimFunds(
 2     string calldata id,
 3     string calldata secret
 4 ) external nonReentrant {
 5     bytes32 h = keccak256(bytes(id));
 6     Order storage o = orders[h];
 7
 8     require(o.state == Status.Validated, "not-validated");
 9     require(block.timestamp <= o.deadline, "deadline");
10     require(msg.sender == o.buyer, "caller");
11     require(keccak256(bytes(secret)) == o.secretHash, "bad-secret");
12
13     o.state = Status.Claimed;
14     (bool ok,) = payable(o.buyer).call{value: o.ethAmount}();
15     require(ok, "transfer-failed");
16
17     emit OrderClaimed(h);
18 }
19
20 function expireOrder(bytes32 h) external onlyOwner {
21     Order storage o = orders[h];
22     require(
23         o.state == Status.Registered || o.state == Status.Validated,
24         "not-expirable"
25     );
26     require(block.timestamp > o.deadline, "not-expired");
27
28     o.state = Status.Expired;
29     availableEscrow += o.ethAmount;
30     emit OrderExpired(h);
31 }
```

Listing 12: Claim and Expire Mechanism (Prototype C)

5.3 Backend Orchestration & Resilience

The Node.js / Express service acts as the glue between three otherwise-independent actors: PayPal's REST + webhook API, the Chainlink DON, and the Solidity Contract. Its only responsibilities are

1. turning a user-initiated "buy $X" click into a PayPal *authorize* call,
2. listening to webhooks and on-chain events,
3. forwarding the right payload to the oracle layer, and
4. calling *capture* or *void* once the contract has reached a terminal state.

Everything else like price conversion, custody, timing guarantees lives on-chain or in the DON. Because 4.2 already laid out PayPal's *authorize* → *capture* → *void* semantics, the text below focuses purely on how the backend wires those calls into each prototype's trust model.

Prototype A

The first backend keeps a private ledger in a lightweight SQLite file named *trades.db*. At start-up the server executes a single *CREATE TABLE IF NOT EXISTS* statement. The resulting table contains one row per PayPal order and only ten columns:

column	**purpose**
order_id (PK)	the PayPal ID in plain text
order_id_hash	*keccak256(order_id)* – used by the Solidity escrow
eth_address	buyer's payout address
amount_usd	purchase amount (decimal dollars)
paypal_status / onchain_status	high-level state machines, initialized to *CREATED / PENDING*
tx_hash	the Chainlink request that will settle the order
authorization_id	PayPal hold identifier, filled after *authorize*
created_at / updated_at	Unix timestamps for audit and recovery

Table 2: Order Management Database

When PayPal fires a *CHECKOUT.ORDER.APPROVED* webhook, the Express route *handleWebhook* immediately acknowledges the *POST*, drops the payload onto a work queue, and returns 200 so PayPal will not retry. The asynchronous worker then:
calls *authorize*, stores the *authorization_id* plus the status = *AUTHORIZED* in *trades*, assembles a tiny JSON "receipt" (*orderId*, *buyerEthAddress*) and passes it to *sendChainlinkRequest*.

```
const auth = await authorizeOrder(orderId);
await updateTrade(orderId, {
      paypal_status: 'AUTHORIZED',
      authorization_id: auth.id });
const receipt = await verifyOrder(orderId);
await sendChainlinkRequest(receipt.orderId, receipt.buyerEthAddress);
```

Listing 13: Backend Flow (Prototype A)

The second long-running task attaches to the escrow contract with ethers.js and listens for *PaymentFulfilled*. Whenever that log appears the listener:

1. pulls the human-readable *orderId* from the contract's *getOrder(hash)* accessor,
2. flips *onchain_status* to *FULFILLED* in SQLite,
3. captures the PayPal authorization, thereby making the fiat leg final,
4. and writes *CAPTURED* into the same database row.

Any error along this path remains local. If *captureAuthorization* throws, the entry stays *AUTHORIZED* and an operator can retry manually else the 29-day hold voids itself.
The combination of database and event logs is what gives Prototype A its resilience. Should the server die mid-flight, it simply re-runs *SELECT * FROM trades WHERE onchain_status='PENDING'* at boot, reconnects to the contract, and resumes listening for the missing events. Nothing on-chain ever depends on the database, and nothing in PayPal can become irreversible without the on-chain proof, so user funds are never at risk. Worst-case could be an order timeout or failure, the backend voids the hold, and the user can try again.
With the fully stateful coordinator described, the next section shows how the stateless, event-driven backend of Prototype B removes the SQLite dependency entirely while keeping the same safety guarantees.

Prototype B

Prototype B removes SQLite entirely and treats the blockchain itself as the canonical order book. The webhook handler still answers PayPal in real time, but instead of writing a row to a local table it immediately records the purchase on-chain through *registerOrder*. All further state transitions are driven by contract events. If the server restarts it simply re-subscribes to the log stream and nothing is lost.
After PayPal signals *CHECKOUT.ORDER.APPROVED* the backend verifies and authorizes the order, converts the dollar amount into cents, and pushes the tuple (orderId, buyer, usdCents*)* to the contract:

```
const details = await verifyOrder(orderId);                // PayPal GET
const auth    = await authorizeOrder(orderId);              // PayPal POST
const cents   = Math.round(details.amount * 100);

await registerOrder(                                      // on-chain Tx
    details.orderId,
    details.ethAddress,
    cents
);
```

Listing 14: Backend Flow (Prototype B)

No local persistence is required. Should the HTTP call fail the hold remains in *CREATED*, the user can retry, and no Ether has moved. The Node.js listener attaches to two contract

events. *OrderRegistered* fires first and the callback launches a Chainlink Functions request that asks the DON to validate the PayPal authorization using a short-lived OAuth token uploaded as a DON-hosted secret. When (and only when) that oracle responds positively the contract emits *PaymentFulfilled*. The listener then captures the PayPal authorization, finalizing the fiat leg:

```
escrow.on('OrderRegistered', async (hash, id, buyer, cents) => {
    await sendChainlinkRequest(id, buyer, cents.toString());   // DON
});

escrow.on('PaymentFulfilled', async (/* …id… */) => {
    const authId = await getAuthorizationId(id);               // PayPal GET
    await captureAuthorization(authId);                        // PayPal POST
    console.log(`fiat leg captured for ${id}`);
});
```

Listing 15: Event Subscriptions (Prototype B)

The only in-memory state is a set of recent webhook IDs that guards against duplicate HTTP posts. An hourly sweep truncates the set so the service cannot leak memory during long runs. Because every authoritative fact – order hash, buyer address, amount, status – is stored in the contract, the backend is free to crash or be redeployed at any time. When it comes back online it:

1. reads the latest block height,
2. re-attaches the two listeners,
3. fetches any outstanding PayPal authorizations that have not yet been captured.

Nothing relies on an internal database, and no manual reconciliation is needed. The event log is the single source of truth. Operationally this reduces maintenance overhead and eliminates a whole class of "what happens if the DB is out of sync?" failures that affected Prototype A.

Prototype C

With the commit–reveal model the backend's job is reduced to a relay: it writes an immutable commitment to the contract, launches a single oracle check, and then steps out of the way. Every later transition – validation, claiming, expiry – comes from the chain itself, so a rebooted server can always catch up simply by replaying logs. When PayPal notifies *CHECKOUT.ORDER.APPROVED* the handler verifies the order, authorizes the hold and converts the amount to cents. The decisive difference from Prototype B is the extra secret hash. The buyer has already sent this commitment alongside the address, so the backend can lodge the full quadruple – order Id, cents, secret hash, payout address – on-chain in one transaction:

```
const d   = await verifyOrder(orderId);              // PayPal GET
await authorizeOrder(orderId);                       // PayPal POST
const cents = Math.round(d.amount * 100);

await registerOrder(                                 // on-chain Tx
  d.orderId,
  cents,
  d.secretHash,                                      // commit
  d.ethAddress
);
```

Listing 16: Backend Flow (Prototype C)

From here the state machine is contract-driven. *OrderRegistered* triggers a Chainlink Functions call. The DON looks up PayPal with a DON-hosted OAuth token and, if the amounts and secret hash match, marks the order *Validated*. Nothing else leaves the backend, no SQL write, no escrow mutation. The listener therefore waits for three log types.

1. *OrderRegistered*: launches the oracle request
2. *OrderClaimed* (buyer revealed secret): captures the PayPal authorization and completes the swap
3. *OrderExpired* (claim window lapsed): voids the authorization so the hold falls away

A trimmed listener shows the essence:

```
escrow.on('OrderRegistered', async (_, id, cents, hash) => {
  await sendChainlinkRequest(id, cents, hash);      // oracle trigger
});

escrow.on('OrderClaimed',  async (hash) => {        // final capture
  const id  = (await escrow.getOrder(hash)).orderId;
  const auth = await getAuthorizationId(id);
  await captureAuthorization(auth);
});

escrow.on('OrderExpired', async (hash) => {         // release fiat
  const id  = (await escrow.getOrder(hash)).orderId;
  const auth = await getAuthorizationId(id);
  await voidAuthorization(auth);
});
```

Listing 17: Event Subscriptions (Prototype C)

Because neither SQLite nor in-process caches store obligation data, resilience is automatic. If the service dies during the one-hour claim window it simply re-attaches to the *OrderClaimed/OrderExpired* topics on restart and proceeds. Deduplication still matters for incoming webhooks, so a one-hour rolling Set of event IDs guards against replay attacks without persisting anything to disk. In day-to-day operations this architecture means:

1. the backend cannot release Ether, but the buyer's secret can,
2. PayPal funds are never captured unless the on-chain claim succeeded,
3. any stuck or forgotten order self-releases: the contract returns the reserved ETH to the pool, and the backend voids the fiat hold.

As a result, Prototype C achieves full end-to-end settlement with no mutable state outside Ethereum and PayPal, while keeping operational logic in a few hundred lines of auditable JavaScript.

5.4 Oracle integration & request lifecycle

Everything the escrow contracts know about PayPal comes from a single Chainlink Functions request that the backend fires after it has finished the fiat-side *authorize*.
The *sendChainlinkRequest* function therefore sits at the heart of all three prototypes. It hides the Functions-toolkit boilerplate, enforces an "either-fulfilled-or-voided" discipline and

returns only when the contract can move on. The helper is written once, imported by every backend, and parametrised so that each prototype forwards just the data its contract needs. The bare-bones flow is nearly identical across A / B / C:

```
const source = fs.readFileSync(path.join(__dirname, "source.js"), "utf8");
const args   = buildArgsForPrototypeX(...);  // differs per prototype

// Prototype A does not upload secrets; B and C use DON-hosted secrets
const tx = await escrow.sendChainlinkRequest(
  source,
  "0x",             // empty encryptedSecretsUrls (using DON-hosted or none)
  SLOT_ID,          // 0 for Prototype A; uploaded slot for B/C
  VERSION,          // 0 for Prototype A; version number for B/C
  args,
  [],               // optional bytesArgs
  SUB_ID,
  GAS_LIMIT,
  ethers.utils.formatBytes32String(DON_ID)
);

const fulfillment = await responseListener
  .listenForResponseFromTransaction(tx.hash);
```

Listing 18: Send Chainlink Request

The wrapper first runs an off-chain simulation in non-production mode, so trivial mistakes (wrong JSON key, typo in amount) never consume gas.
Before posting the request it also asks the Subscription Manager for a cost estimate, logging the expected LINK fee. The figure later printed in the fulfilment receipt should never surprise the operator. When the response comes back four outcomes are possible:

1. ***FULFILLED***: decode the tuple, log the happy path, return to caller
2. ***USER_CALLBACK_ERROR***: call *voidAuthorization* and mark the trade *FAILED*
3. ***DON_EXECUTION_ERROR***: same void-and-fail branch
4. **timeout / transport error**: treat as execution error, void immediately

A small convenience function extracts a revert reason (EIP-838) from the return-data when present. That string is logged to the console so failed requests can be quickly found during incident response.

Prototype A

Prototype A's contract needs only the PayPal order-ID and the buyer address.
The helper therefore encodes exactly those two values: *orderId* and *buyerEthAddress*.
Secrets are not required because the DON does not call PayPal directly. It hits the backend's */receipt* endpoint, which has already verified the order. Consequently, payload uses *secretsLocation: None*, and the off-chain JSON returned by the backend can be cached indefinitely in the DON. On success the escrow emits *PaymentFulfilled*, the event-listener captures the authorization and the helper's job is done. On any other code path *voidAuthorization(authorization_id)* is called, the row in *trades.db* is updated to *VOIDED / FAILED* and the promise rejects so the webhook worker can print the stack-trace.

Prototype B

The B-variant pushes verification into the DON, so an OAuth token must accompany every request. The *uploadSecrets* function copies the short-lived bearer token into the DON's encrypted store and returns a <*slot, version*> pair. The response tuple now contains *(address buyer, uint256 cents, string orderId)*. Decoding those three fields is enough for a pretty console log. If the callback reverts with *InsufficientEscrowBalance* or any other reason, the helper extracts the revert string, prints it and voids the authorization exactly as in Prototype A.

Prototype C

For the final prototype the argument list changes a little. Instead of the buyer address we are passing the secret hash. The DON script calls PayPal and checks that the *reference_id* inside the order matches the submitted commitment. The contract, in turn, stores the tuple *(cents, secretHash, orderId)* but withholds payment until the buyer supplies the pre-image inside *claimFunds*. In the success case the helper sees *FULFILLED*, prints the decoded tuple and exits. If the DON cannot fetch PayPal or the callback reverts e.g. ("*Secret hash mismatch*"), the helper voids the authorization. Later, when *OrderExpired* appears on-chain, the event listener also voids any hold that might still exist. The double-guard guarantees that PayPal never keeps money locked once the on-chain state is final.

6 Evaluation

This chapter evaluates the developed prototype against performance, reliability, and security criteria. Benchmark tests assess gas costs, transaction throughput, and latency introduced by off-chain verification. The analysis identifies strengths, such as reduced trust assumptions and transparent state tracking, and limitations, including dependency on PayPal's dispute window and oracle uptime. Security considerations and possible optimizations are discussed to inform further refinement of the system.

6.1 Evaluation Framework and Methodology

This evaluation framework provides a systematic approach to assess the fundamental trade-offs between trust-minimization and performance across the three prototype architectures. The methodology employs structured qualitative assessment supported by empirical measurements gathered from prototype testing under standardized conditions.

Evaluation Approach

The framework evaluates each prototype across eleven distinct dimensions, organized into two primary categories: Trust-Minimization Assessment (seven metrics) and Performance Assessment (four metrics). Each dimension employs a structured scoring approach from 1-10, where higher scores indicate superior performance within that specific dimension. This methodology, inspired by [24], prioritizes transparency and reproducibility while avoiding artificial mathematical precision where qualitative judgment is more appropriate.

Standardized Testing Environment

All prototypes undergo evaluation under identical conditions to ensure fair comparison:

Market Conditions: LINK $17, ETH $3,400, Gas 1.5 GWEI
Oracle Response Time: Consistent 55-second average across all prototypes using Chainlink Functions
Network Environment: Sepolia testnet with standardized gas limits and block times
Transaction Scenarios: Identical PayPal order flows with consistent USD amounts and timing

Trust-Minimization Metrics Framework

The trust-minimization test measures seven important dimensions that define the level of intermediary dependence and cryptographic security:

1. Custodial Exposure (20% conceptual weight)

Measures the period of time during which money is at risk in the process of settling transactions. Evaluation is based on the overlap in custody time against total transaction time, asking when PayPal has authorized funds, but ETH transfer is pending. The shorter the exposure, the better the trust-minimizing properties.

2. Deterministic Execution (18 percent conceptual weight)

Determines the predictability and reproducibility of transaction outcomes. Determines whether the same inputs always yield the same outputs, and the extent to which backend-controlled timing and discretionary decision-making are involved, which adds variability to the execution.

3. System Autonomy (15% conceptual weight)

Quantifies the level of autonomous functioning without human intervention or operator actions of trust. Looks at automation of state transitions, self-healing, and less reliance on external coordination to complete a transaction.

4. Fallback Robustness (12% conceptual weight)

Evaluates how the system can cope with failure conditions gracefully by using automated recovery. Tests contract-level timeout functionality, refund process and end-to-end error-handling without manual intervention.

5. Oracle Verifiability (15% conceptual weight)

Investigates the auditability and transparency of oracle behavior by cryptographic attestations. Considers SGX attestations, quorum signatures, decentralized consensus mechanisms, which allow oracle operations to be verified externally.

6. Data Integrity (10 % conceptual weight)

Evaluates the robustness of data protection systems using hash commitments, cryptographic proofs, and safe on-chain storage patterns. Looks at resistance to data manipulation and immutability of recorded transaction states.

7. Modularity (10% conceptual weight)

Assesses architectural flexibility and the separation of components that allows evolving a system without breaking fundamental security assurances. Evaluates the capability of upgrading individual components preserving trust-minimization properties

Performance Assessment Framework

The performance evaluation examines four dimensions critical to practical deployment and user adoption:

1. User Experience (35% conceptual weight)

Test the complexity of interaction and friction of the user journey. Quantifies the number of actions, waiting time, and mental complexity of transactions that a user is required to undertake. Evaluates the compatibility with the known patterns of e-commerce interaction.

2. Cost Efficiency (30 % conceptual weight)

Looks at the overall cost of transactions compared to industry benchmarks of blockchain payment processing. Takes into account gas costs, oracle costs, and overhead. Takes $4.00 as reference baseline of average fiat-to-crypto on-ramp fees. [37]

3. Time Efficiency (20 % conceptual weight)

Evaluates the speed of transaction completion with expectations of consumers on online payments. End-to-end settlement time is measured between PayPal initiation and ETH receipt with 2-minute baseline representing the general expectations of online payment confirmation. [38]

4. Scalability (15 % conceptual weight)

Assesses throughput capacity and resource capacity limits. Studies the consumption patterns of gases, dependencies between transactions in sequence and coordination overhead that affects the feasible scale of deployment.

Assessment Methodology

Each metric receives structured evaluation through:

- **Empirical Measurement**: Direct quantification of relevant parameters (timing, costs, gas consumption) under controlled testing conditions.
- **Architectural Analysis**: Systematic examination of code structures, state machines, and interaction patterns to assess inherent characteristics.
- **Scenario Testing**: Evaluation of behavior under various conditions including failure modes, edge cases, and operational stress scenarios.
- **Comparative Benchmarking**: Assessment relative to established industry practices and user expectations rather than abstract theoretical maxima.

This framework emphasizes practical deployment considerations while maintaining rigorous evaluation standards. The structured approach enables informed architectural selection based on specific use case requirements and stakeholder priorities, acknowledging that different deployment scenarios optimize for different characteristics within the trust-minimization versus performance spectrum.

6.2 Trust-Minimization Assessment

This section evaluates the degree to which each prototype reduces dependence on intermediaries and provides cryptographic guarantees for transaction security. The assessment examines seven critical dimensions that collectively determine the trust-minimization characteristics of each architectural approach.

Custodial Exposure Analysis

Prototype A illustrates a high level of custodial risk where the funds are exposed to the risk of loss during the entire 80 seconds it takes to verify the payment with Chainlink and receive the ETH. Both PayPal and blockchain coordination is under the full control of the backend, and there is no pre-commitment of funds. Users have to have faith that the backend will perform settlement correctly with no cryptographic assurances securing their transaction. **Score: 3/10**

Prototype B reduces custodial exposure through on-chain order registration that commits to transaction parameters before Chainlink validation. However, funds are not pre-reserved, meaning there remains an 80-second window where ETH availability depends on escrow balance and backend operational integrity. The two-phase architecture provides some protection but still requires trust during the settlement window. **Score: 6/10**

Prototype C achieves superior custodial protection through immediate fund reservation upon order registration. The critical innovation is that ETH funds are cryptographically locked and earmarked for the specific buyer from the moment of registration (*availableEscrow -= ethAmt*). This reduces the actual custodial risk period to just 20 seconds - the brief window between PayPal authorization and on-chain order registration. Once registered, the buyer has cryptographic guarantee that their designated ETH amount is reserved and claimable, regardless of backend availability or operational status. The system provides the strongest custodial protection through pre-commitment rather than post-validation reservation. **Score: 9/10**

Deterministic Execution Capabilities

Prototype A scores the lowest in the predictability of execution because of backend-controlled timing in which the same PayPal payments may yield different results based on operational status. The timing of Oracle requests is completely at the discretion of the user and failures in the system may lead to the incomplete settlement without automatic recovery. The architecture does not provide reproducible state transitions, and the results of transactions are contingent on the availability of the backend and its operational choices instead of cryptographic reasoning. **Score: 2/10**

Prototype B shows significant progress by event-driven architecture that generates on-chain commitments that are immutable. After *OrderRegistered* events are released, the further state transition is predictable and is controlled by the smart contract logic instead of the backend decision-making. Unsuccessful transactions may place the system in intermediate states that need manual intervention. **Score: 6/10**

Prototype C is almost perfectly deterministic in its fully autonomous state machine implementation. Each state transition is cryptographically enforced by oracle verification or user secret disclosure, which removes backend discretionary control. The commit-reveal protocol guarantees that the same inputs (PayPal payment + secret hash) will always have the same outputs (validated order + claimable funds). The results of transactions can be predicted solely based on on-chain state, which is an example of algorithmic determinism in the blockchain payment processing. **Score: 9/10**

System Autonomy Assessment

Prototype A has little autonomy, and must be able to complete transactions with constant backend access. Backend coordination between PayPal APIs and blockchain transactions is required to make all settlement decisions. System failures normally need manual intervention to fix incomplete transactions and automated mechanisms to deal with edge cases or timeouts are absent. **Score: 3/10**

Prototype B has moderate autonomy with an event driven architecture that minimizes backend state dependencies. The blockchain is used as the canonical source of truth of pending orders to allow recovery of the system after service interruptions by replaying the events. Nonetheless, the backend is still important to coordinate the oracles and complete the settlement of PayPal, which restricts the real autonomous functioning. **Score: 5/10**

Prototype C maximizes system autonomy by removing backend control of settlement decisions. After the validation of oracle, the system will be fully functioning on smart contract logic and user-initiated actions. Unclaimed funds are dealt with automatically by automated expiry mechanisms, without operator intervention. The architecture enables full backend unavailability during periods of user claiming, and proves real autonomous operation capabilities. **Score: 8/10**

Fallback Robustness Mechanisms

Prototype A has limited fallback features, with the expiry of PayPal authorization hold of 29-days being the final backup. Oracle requests or settlements that do not succeed need to be manually intervened to cancel PayPal authorizations and prevent charges to the user without getting ETH. The system does not have automated error recovery or time out management other than simple transaction reversal. **Score: 3/10**

Prototype B enhances fallback resiliency by its two-phase architecture that decouples PayPal and blockchain activities. Oracle requests that fail can be re-executed without any on-chain state change, and transactions that fail to complete have clear audit trails through event logs. Nevertheless, the two stages still have coordination problems that need operator intervention to eliminate inconsistent states. **Score: 6/10**

Prototype C illustrates fallback mechanisms in full by its expiry system based on timeouts. The *expireOrder* feature will process unclaimed validated orders, releasing the reserved funds back to the escrow pool and cancelling PayPal authorizations. The architecture has several recovery paths in case of various failure scenarios, ensuring system integrity even in case of prolonged disruptions of the operation. **Score: 9/10**

Oracle Verifiability Standards

Any prototypes can attain similar oracle verifiability using the standardized attestation mechanisms of Chainlink Functions. The SGX-secured execution environments offer cryptographic evidence of correct computation, and multi-node consensus offers no single point of oracle failure. The DON architecture allows the oracle behavior to be externally verified via transparent quorum signatures and economic staking.

Prototype A has basic oracle verification that is used to perform backend receipt validation, and Chainlink security guarantees are standard without extra verification layers. **Score: 8/10**

Prototypes B and C take advantage of improved oracle verification by querying PayPal APIs directly in SGX enclaves, which offers cryptographic evidence of payment confirmation without revealing sensitive authentication tokens. The secret management system is encrypted, allowing oracle nodes to access PayPal APIs in a security isolation that does not allow external observers to access. **Score: 8/10 each**

Data Integrity Evaluation

Prototype A has basic data integrity by transaction hashing and duplicate prevention, but does not provide any additional cryptographic guarantees beyond backend data consistency. **Score: 6/10**

Prototype B improves data integrity by storing order commitments on-chain, producing immutable records of transaction parameters prior to settlement execution. The hash-based order verification is used to make sure that PayPal and blockchain data are consistent. **Score: 7/10**

Prototype C optimizes data integrity by cryptographic hash commitments that commit transaction metadata to user secrets prior to oracle validation. The commit-reveal protocol gives a powerful cryptographic evidence of transaction validity and preserves the privacy of sensitive data until the time of claiming. **Score: 9/10**

Architectural Modularity

Prototype A demonstrates limited modularity due to tight coupling between backend components and settlement logic, making system evolution challenging without comprehensive redesign. **Score: 3/10**

Prototype B provides moderate modularity through its event-driven architecture that separates concerns between registration and fulfillment phases, enabling independent optimization of each component. **Score: 6/10**

Prototype C exhibits superior modularity through clear separation of custody, validation, and execution responsibilities. The architecture supports independent evolution of oracle verification logic, secret management systems, and settlement mechanisms without compromising core trust guarantees. **Score: 8/10**

Trust Dimension	Weight	Prototype A	Prototype B	Prototype C
Custodial Exposure	20%	3/10	6/10	9/10
Deterministic Execution	18%	2/10	6/10	9/10
System Autonomy	15%	3/10	5/10	8/10
Fallback Robustness	12%	3/10	6/10	9/10
Oracle Verifiability	15%	8/10	8/10	8/10

Data Integrity	10%	6/10	7/10	9/10
Modularity	10%	3/10	6/10	8/10
Composite Score	**100%**	**3.9/10**	**6.2/10**	**8.6/10**

Table 3: Trust-Minimization Composite Assessment

The analysis shows a definite increase in the trust-minimization abilities in the prototype spectrum. Prototype A is optimized to be simple to operate at the expense of serious assumptions of trust, and is appropriate to consumer applications where convenience is more important than security. Prototype B tries to strike a balance between security and usability but fails to achieve either of the two ideally and is a transitional design with little practical rationale. Prototype C provides maximum trust-minimization in the form of full cryptographic assurances, and is therefore suitable in institutional and high-value transactions where the complexity of security is justified by the value at stake.

6.3 Performance Assessment

This section evaluates the practical deployment characteristics that determine user adoption potential and operational viability across the three prototype architectures. The assessment examines four critical dimensions that collectively determine the performance profile of each approach.

User Experience Analysis

Prototype A offers the best user experience by having a simplified one-action workflow. Users make transactions by paying with PayPal and automatically getting ETH without any further blockchain communication. The system offers a smooth consumer-level experience similar to what is offered by conventional e-commerce sites with only a single wallet interaction needed to settle the entire transaction. The backend automation takes care of all the coordination between PayPal APIs and blockchain settlement, removing the complexity and technical overhead of the users. **Score: 10.0/10**

Prototype B has the same user experience as Prototype A although it has a more advanced backend architecture. End users do not see the dual-phase design but see a single-click buy now interface. Registration of orders on the chain is carried out transparently in the process of the transaction flow without the need to take any additional actions by the user or wait. The complexity of the architecture offers security advantages without degrading the sleek user experience that allows mass adoption. **Score: 10.0/10**

Prototype C has a considerably lower usability as it involves three-phase interaction model that involves active user involvement in the transaction lifecycle. The users have to generate cryptographic secrets, make PayPal purchases, and manually receive funds in the form of individual blockchain transactions. The commit-reveal architecture requires a 90-second delay between the completion of an order and claiming of funds, during which users are required to hold onto their secret and execute the final settlement transaction. Such complexity makes the median process time about 3.5 minutes, with human decision-making stages taking the lead over technical latency. **Score: 1.0/10**

Cost Efficiency Evaluation

Prototype A has higher cost effectiveness with overall transaction fees of 4.64, including 1.58 gas fees (309,000 gas units) and 3.06 oracle fees (0.18 LINK). The single-transaction architecture reduces the use of resources on the blockchain and is near to the industry cost benchmarks. Its simplified structure does not incur the gas overhead of multi-phase architectures, so it is cost-effective to moderate-value transactions. **Score: 8.4/10**

Prototype B shows a moderate cost efficiency of 5.95 total cost with gas costing almost twice as much, 2.89 (567,000 gas units), because of the extra order registration transaction. Oracle costs are fixed at 3.06, but the architectural separation adds 28 percent to the Prototype A costs. The marginal security gains of order hash verification have dubious value proposition compared to the extra cost overhead. **Score: 5.1/10**

Prototype C has the most expensive operational costs of $6.44 in total, and gas costs of $3.38 (663,000 gas units in three separate transactions: registration, oracle validation, and user claiming). The multi-phase architecture is a 39 percent cost premium over Prototype A, because of the computational overhead of full trust-minimization. The price difference proves the basic rule that blockchain security and decentralization involve high computational resources. **Score: 3.9/10**

Time Efficiency Assessment

Prototype A has the best time efficiency of 120 seconds total completion time, which is exactly the baseline expectation. The backend-centric architecture removes the overhead of coordination and user latency, and instead offers instant settlement after oracle validation. All the operations are automatic and do not require user interaction, which results in predictable timing properties that can be used in consumer applications. **Score: 10.0/10**

Prototype B shows a minor timing degradation of 140 seconds of total completion time because the dual-phase architecture necessitates sequential on-chain registration and oracle validation. The 20-second overhead is caused by further blockchain interactions and event coordination, yet it is not too high to be tolerated by consumers. The timing penalty is the price of the increased security in the form of immutable order commitments. **Score: 8.7/10**

Prototype C has significant timing constraints of 210-second total completion time with the 90-second user waiting time between order validation and manual claiming being its key driver. The long time is indicative of the human element of decision making associated with buyer-controlled settlement where users are required to take the final step of claiming the transaction. Technical processing is still efficient, however the user interaction requirements introduce timing delays that are not appropriate in immediate settlement cases. **Score: 4.0/10**

Scalability Assessment

Prototype A shows better operational scalability by its simplified single-transaction design that reduces complexity of coordination. Backend systems only need simple PayPal API calls and then oracle requests, which make them have linear scaling properties. There are no complex state management or user interaction dependencies that can limit high-throughput processing to PayPal API rate limits and oracle response capacity, rather than architectural limitations. **Score: 9.2/10**

Prototype B has a moderate scalability issue because of its two-phase architecture, which involves coordination of the registration and settlement processes. The system should be able to ensure state consistency over several blockchain interactions as well as oracle validation timing. Inconsistent states may also be introduced by failed intermediate transactions, causing operational overhead. The architectural isolation has security advantages, but also adds the complexity of coordination that limits practical throughput in high-load environments. **Score: 6.8/10**

Prototype C faces severe scalability limits because of a three-phase structure and user-driven claiming needs. The system should synchronize backend processes, oracle validation and user interactions over long periods of time, generating numerous failure modes and coordination bottlenecks. Claiming that is user-dependent results in unpredictable timing differences that make it difficult to batch process and plan capacity. The commit-reveal protocol necessitates a lot of order state and timeouts management, which adds a significant level of complexity to operations. **Score: 3.5/10**

Performance Dimension	Weight	Prototype A	Prototype B	Prototype C
User Experience	35%	10.0/10	10.0/10	1.0/10
Cost Efficiency	30%	8.4/10	5.1/10	3.9/10
Time Efficiency	20%	10.0/10	8.7/10	4.0/10
Scalability	15%	9.2/10	6.8/10	3.5/10
Composite Score	**100%**	**9.0/10**	**7.2/10**	**2.8/10**

Table 4: Performance Composite Assessment

The assessment shows that performance optimization is directly proportional to architectural simplicity. The backend-centric design of Prototype A removes overhead of user interaction, and reduces blockchain resource usage and coordination complexity, resulting in desirable properties of high-volume consumer applications. Prototype B tries to find a compromise between security and performance but adds costs and complexity without commensurate advantages, and is a transitional design with few practical reasons to exist. The goal of maximum trust-minimization inherent in Prototype C is a fundamental limitation on performance in all aspects, and is only appropriate in high-value transactions where security is worth the high performance costs.

6.4 Comparative Analysis and Deployment Recommendations

The empirical evaluation reveals a fundamental trade-off between trust-minimization and performance optimization, quantified through a strong inverse correlation (-0.96) between security and adoption factors. This relationship represents the central challenge facing fiat-to-crypto bridge design: cryptocurrency's core promise of eliminating intermediaries directly conflicts with the practical usability requirements for mass adoption.

Architectural Performance Spectrum

The three prototype architectures create an obvious performance continuum in which each design is optimized to different priorities. Prototype A has outstanding performance properties (9.0/10 composite score) because of backend-centric design that removes the overhead of user interaction and reduces the complexity of computations. The single-transaction architecture is streamlined to provide the most favorable environment to high-volume consumer applications where convenience is the key to adoption success.
The other extreme, Prototype C, does the opposite, and maximizes trust-minimization (8.6/10) by providing full cryptographic assurances and deterministic execution paths. The commit-reveal architecture is able to achieve a low degree of intermediary dependence at the cost of significant performance overheads (2.8/10 composite score) that limit realistic use to high-value institutional settings where the complexity of security can be justified by the value of the transactions.
Prototype B tries to compromise these conflicting goals by using dual-phase architecture and results in moderate trust-minimization (6.2/10) and reasonable performance (7.2/10). This compromise solution, however, does not perform well on either axis, resulting in a solution that is more expensive and more complex than it needs to be without corresponding gains in either security or usability.

Economic Deployment Boundaries

Transaction economics define definite limits of deployment in terms of fixed oracle cost and variable user complexity tolerance. The minimum economic viability point is at 4.1 percent oracle fee at 75 dollars transactions, which is the same on all architectures. But the overall cost difference between architectures provides a unique market positioning opportunity.
The total transaction cost of prototype A of $4.64 allows the deployment of moderate-value transactions with a reasonable ratio of fees. The low-cost structure facilitates consumer DeFi usage and speculative trading where users are not concerned with security complexity but rather the ease of use.
The total cost of Prototype C of $6.44 limits the deployment to transactions greater than 200 dollars in order to have reasonable fee ratios, which makes it specifically suited to institutional custody and high-value trade cases. The 39 percent cost premium is an expression of the overhead of computation needed to achieve full trust-minimization, and so forms a natural market segmentation by transaction value and security needs.

Evaluation Dimension	Prototype A	Prototype B	Prototype C
Trust-Minimization Score	3.9/10	6.2/10	8.6/10
Performance Score	9.0/10	7.2/10	2.8/10
Economic Viability	$75+ txn	$150+ txn	$200+ txn
Target User Base	Consumer DeFi	Moderate Trading	Enterprise Treasury
Deployment Rationale	UX Optimization	Balanced Approach	Security Maximization

Table 5: Strategic Deployment Framework

The evaluation establishes three distinct deployment contexts where each architecture provides optimal value. Consumer applications prioritizing user adoption should deploy Prototype A's streamlined design for DeFi participation, yield farming, and speculative trading activities where convenience drives market success. The single-click user experience enables broad accessibility while maintaining acceptable security for moderate-value transactions.

Enterprise applications handling institutional treasury operations should deploy Prototype C's comprehensive security model for custody operations, regulated trading, and compliance-sensitive scenarios. The cryptographic guarantees and deterministic execution characteristics justify the operational complexity and cost premiums when fiduciary responsibilities demand maximum trust-minimization.

Implementation Recommendations

Based on empirical performance data, deployment strategies should align architectural characteristics with specific use case requirements rather than attempting universal optimization. Prototype A serves consumer markets effectively where user experience determines adoption success, while Prototype C addresses institutional requirements where security complexity is acceptable given transaction stakes.

Prototype B lacks clear deployment justification given its compromise positioning that achieves neither optimal performance nor maximum security. The 28% cost increase over Prototype A provides marginal security improvements that don't justify the complexity increase for consumer applications, while falling short of the comprehensive guarantees required for institutional deployment.

The architecture selection decision should be driven by user sophistication levels, transaction value ranges, and regulatory compliance requirements rather than abstract security preferences. Consumer-focused deployments benefit from Prototype A's optimization for adoption, while enterprise deployments require Prototype C's optimization for trust-minimization.

7 Further Considerations

Multi-Asset Support and Token Integration

The existing implementation limits transactions to ETH transfers using the *convertUsdToEth()* method, which makes it less applicable to the market, even though it has sturdy architectural underpinnings. An extension to full multi-asset support necessitates radical changes to the Order struct and pricing functionality whilst maintaining the existing trust-minimization assurances.

Fungible token integration is the extension of the Order structure to contain token contract addresses and quantity specifications. The *convertUsdToEth()* function should be swapped out with a more dynamic *convertUsdToToken(address token)* implementation that makes use of Chainlink Price Feed aggregators to provide real-time valuation of a variety of asset classes. This change allows users to buy USDC, WBTC, or any ERC-20 token directly and still have the same security model as the one in the commit-reveal architecture.

Non-fungible token support is a challenge in its own right because of the need to value individual assets and verify ownership. The system should be connected to NFT marketplaces such as OpenSea or use dynamic pricing oracles that are able to evaluate floor prices and rarity scores. The Order struct needs to include new fields of token IDs and metadata verification, and the escrow mechanism should be able to transfer unique assets instead of fungible quantities atomically.

Dynamic pricing challenges are experienced when serving volatile assets where the price volatility between PayPal payment and blockchain settlement may cause arbitrage opportunities or losses to users. Price tolerance bands and time-bounded transactions are necessary, and it will be necessary to coordinate the timing of fiat payments and on-chain price feeds to fairly execute both buyers and merchants. [26]

Multi-Merchant and Multi-PSP Architecture

The existing single-owner contract paradigm limits use to single merchants, and thus does not allow the platform economics required to achieve mass adoption. To scale to multiple merchants, access control, fee distribution, and payment routing mechanisms must be fundamentally changed.

Multi-merchant integration entails substitution of the rudimentary *onlyOwner* modifier with a more complex merchant registry mechanism. Each merchant has a unique identifier and fee structure and the contract has separate escrow balances per merchant to avoid cross-contamination of funds. Commission structures should be established to facilitate the platform operations and the ability to offer competitive merchant prices.

Payment service provider expansion beyond PayPal is also possible but hard to implement. Modular attestation architecture needs to be able to verify various PSP formats and authentication mechanisms. Stripe, Apple Pay, and the conventional credit card processors have varying webhook formats and verification needs. The oracle system should be able to support pluggable verification modules capable of authenticating captures of various payment sources and still provide consistent security assurances.

The complexity of settlement and reconciliation is very high when there are many merchants and PSPs. The system must have automated fee distribution, merchant payout scheduling and

cross-PSP reconciliation. The upgrades of smart contracts are more complicated when they have to support the interests of multiple stakeholders, and governance mechanisms are needed to balance the needs of the platform operation and the autonomy of merchants.

Multi-Chain Deployment Strategies

The assessment in Chapter 6 found cost efficiency to be the main consumer barrier, with the transaction costs of $6.44 limiting its use to high value transactions of more than 200 dollars. Alternative blockchain implementations provide cost savings that are dramatic and change the economic viability calculus but need to be carefully weighed against security and user experience trade-offs.

Layer 2 network benefits consist of 90-99 percent gas cost savings that allow Prototype C to implement its full-scope security model of transactions as low as 75 dollars. Arbitrum and Optimism offer the current Chainlink oracle support to 15-minute state commitment batching, which lowers the costs of individual transactions without compromising the security guarantees that are ensured by the commit-reveal architecture. Fraud proof integration preserves the remarkable fallback robustness properties that make Prototype C so much more superior to simpler alternatives. [39]

Alternative L1 chain considerations like Polygon, BSC, and Avalanche have comparable cost advantages but different security models and chainlink oracle support levels. The choice of network is highly dependent on the destination preferences of buyer assets - users who want to use Polygon DeFi will enjoy native deployment, but users who want to hold their assets on the Ethereum mainnet will need to bridge between chains, which adds complexity and trust assumptions.

Technical implementation requirements involve the verification of Chainlink Functions availability on target networks, adjusting gas estimation algorithms to various fee structures, and network-specific optimizations. Cross-chain asset bridging will need to be incorporated with canonical token bridges to allow deposits into escrow to be performed smoothly without adding any new trust assumptions on top of the ones already accepted in the base architecture.

Regulatory Compliance and Privacy Considerations

Regulation of financial services is also becoming a factor in the operations of cryptocurrencies and the architecture of compliance must be proactive in striking a balance between regulatory demands and privacy principles and decentralization objectives.

KYC/AML integration needs identity verification functionality that is capable of meeting regulatory needs without compromising privacy of transactions where legally possible. Zero-knowledge proof systems are promising methods of proving compliance without disclosing sensitive personal information, but the complexity of implementation and regulatory acceptance are major problems.

MiCA compliance under the European Markets in crypto asset regulation mandates crypto service providers to have certain operational controls, customer protection and reporting capabilities. The systems used in the EU jurisdiction are required to have transaction monitoring, suspicious activity reporting, and customer fund segregation systems that could be incompatible with decentralized architecture principles. [40]

GDPR privacy requirements add more complexity to systems handling data of EU citizens, which necessitates explicit consent, data portability, and deletion features that conflict with blockchain immutability ideas. The implementation can be hybrid in nature with the personal data off-chain and the transaction commitment maintaining on-chain integrity.

Cross-jurisdictional challenges arise when serving international users in diverse regulatory environments, where compliance routing must be dynamic according to user location and the laws that apply. The system should strike a balance between the complexity of compliance and efficiency of operations and provide consistent security assurances across jurisdictions.

Strategic Outlook

The study shows that trust-minimized fiat-to-crypto bridges are technically feasible but economically and legally constrained in their practical applicability. The oracle cost floor of $3.06 makes fee structures prohibitive on small transactions, which is fundamentally incompatible with the democratic accessibility of cryptocurrency and limits the application to institutional use cases with high value.

Market evolution requirements involve oracle cost minimization by the way of efficiency enhancement or alternative verification systems, regulatory certainty that allows compliant decentralized operation, and user education that appreciates the complexity of security over convenience optimization. Unless these underlying limitations are overcome, fiat-crypto bridges will continue to be useful to particular market segments but not revolutionary infrastructure that can provide access to the financial system to the masses.

The technology development priorities should be to lower verification costs by batch processing, integration of zero-knowledge proofs, or other oracle mechanisms that can promise security guarantees and have lower per-transaction costs. Layer 2 deployments provide instant relief in terms of cost but need to be analyzed carefully in terms of security to make sure that trust-minimization principles are not compromised across various consensus mechanisms.

Industry impact potential is still high in the case of institutional custody, regulated trading platform and high-value individual transactions where the complexity of security warrants the cost of operations. The study presents quantified architectural selection and deployment optimization frameworks that can be used to inform practical deployments in a variety of use cases and regulatory contexts.

The inherent trade-off between trust-minimization and accessibility is an ongoing issue that needs to be addressed through further innovation of cryptographic protocols, regulation, etc.

8 Conclusion

The study solved the inherent problem of allowing safe conversion of fiat to crypto with minimal reliance on trusted intermediaries, which lies at the core of the potential promise of cryptocurrency to remove centralized financial gatekeepers. By creating and empirically testing three different prototype architecture designs, this paper presents the first quantified model of how the trade-offs between trust-minimization and practical usability in the design of fiat-to-crypto bridges can be understood.

The main novelty of the work is showing that trust-minimized fiat-to-crypto bridges can be technically implemented by integrating oracles with Chainlink Functions, and successfully reduce the trust assumptions required relative to centralized exchanges. The commit-reveal architecture used in Prototype C provides an outstanding trust-minimization (8.6/10) with extensive cryptographic assurances, deterministic execution paths, and buyer-controlled settlements that remove backend control over the allocation of funds. Nonetheless, the study shows a major economic limitation that negates the democratic accessibility promise of cryptocurrency systems. The prohibitive fee structures of transactions below 75 dollars due to the $3.06 oracle cost per transaction force the architects to decide between extensive verification and wide accessibility. This cost sensitivity is not just a technical constraint, it is a structural constraint that limits the practical deployment to high-value institutional use cases instead of a transformational financial system accessibility originally envisioned.

The assessment model provides quantitative demonstration of security-performance trade-off with a high negative correlation (-0.96) between the trust-minimization and performance optimization. This relationship plays out on many axes: Prototype A delivers high performance attributes (9.0/10) due to backend-focused design that balances on user experience, whereas Prototype C has the highest security assurances at the expense of significant performance overheads (2.8/10) that limit its deployment to enterprise use cases. The spectrum of architecture discloses three different deployment contexts as opposed to a universal solution. Consumer applications that emphasize adoption are well served by the single-transaction simplicity of Prototype A, enterprise applications that demand the utmost security are well served by the comprehensive cryptographic assurances of Prototype C, and Prototype B falls short on both axes. This result questions the premise that security incremental gains justify complexity increases in all use cases.

The economic analysis sets definite limits of deployment on the basis of transaction values. Prototype A facilitates moderate-value transactions that are greater than 75 dollars with relatively good fee ratios, which makes it suitable to be used in consumer DeFi and speculative trading. The total cost of Prototype C of 6.44 dollars limits its application to transactions of over 200 dollars, which provides natural market segmentation in the direction of institutional custody services and regulated trading environments where the complexity of security can be justified by the size of the transaction. Multi-chain deployment strategies may offer solutions to cost limitations, and Layer 2 networks can offer 90-99 percent gas cost savings that can make Prototype C a consumer-viable solution rather than the enterprise-only solution it currently is. Such deployments however need to be properly analyzed on security grounds to guarantee that the principles of trust-minimization are not compromised across

different consensus mechanisms, and depend critically on Chainlink Functions availability across target networks.

The study finds the regulatory compliance as a new limitation that can add to the accessibility issues caused by oracle costs. The need to integrate KYC/AML, the necessity to comply with MiCA, and the need to consider GDPR privacy add more complexity to the architecture that is incompatible with decentralization. These regulatory pressures indicate that even in the event that oracle costs were lowered, compliance overheads would continue to act as a barrier to mass consumer adoption.

The work is limited to PayPal integration as a representative payment service provider, and therefore is not generalizable to a variety of fiat payment ecosystems. Though comprehensive, the evaluation framework is subject to the existing market conditions and technology limitations that can change drastically. Oracle cost models, regulatory frameworks, and blockchain scaling solutions are evolving rapidly, and may change the basic trade-offs found in this work. Future work should explore batch processing protocols that may lower the cost per transaction of the oracle, the integration of zero-knowledge proofs to support privacy-preserving compliance, and alternative verification protocols that can support the same security guarantees with lower operational costs. The patterns of cross-chain interoperability and the systems of intent-based transactions are other directions in which the accessibility limitations found in the current research can be resolved.

This research shows that the key research question, i.e., how fiat payments managed by consumer online payment systems can be safely transformed into on-chain assets with minimal reliance on off-chain intermediaries, can be answered in the affirmative at the technical level but has fundamental economic and regulatory limitations in practice. The technological breakthrough of cryptographically verified off-chain payment validation via oracle integration is a real step forward to trust-minimized financial infrastructure. Nevertheless, the oracle cost floor of $3.06 (as of 2025) is a constant obstacle that goes against the democratic nature of cryptocurrency and limits its usage to certain high-value market segments.

Until architectural solutions are found that can fundamentally change these trade-offs instead of optimizing them incrementally, fiat-crypto bridges will continue to be useful in the context of institutional treasury functions, regulated trading platforms and high-value individual transactions where the complexity of security balances the cost of operational overhead. The study offers quantified architectural selection and deployment optimization frameworks that could be used to guide practical applications in these limited but important use cases. The inherent trade-off between getting rid of trusted intermediaries and making cryptocurrency systems practically usable is a long-standing issue that will need ongoing work on cryptographic protocols, oracle design, and user experience to realize the transformational promise of cryptocurrency systems as initially envisioned. This paper lays out empirical underpinnings to future study which can systematically work towards addressing these limitations without compromising the fundamental tenets of trust-minimization that make blockchain-based financial infrastructure unique compared to their centralized counterparts.

Appendix

The implementation of the three prototypes can be found on the .zip that accompanies this work.

Bibliography

[1] Skyquestt, "Cryptocurrency Market Size, Share, and Growth Analysis," 02 2025. [Online]. Available: https://www.skyquestt.com/report/crypto-currency-market. [Accessed 17 02 2025].

[2] Triple-A, "Cryptocurrency ownership data – Triple-A," 05 June 2024. [Online]. Available: https://www.triple-a.io/cryptocurrency-ownership-data. [Accessed 25 February 2025].

[3] D. Matsuoka and E. Lazzarin, "Estimating the number of real crypto users," 16 October 24. [Online]. Available: https://a16zcrypto.com/posts/article/estimating-crypto-users/. [Accessed 20 February 2025].

[4] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf. [Accessed 20 February 2025].

[5] "History of bitcoin," [Online]. Available: https://en.wikipedia.org/wiki/History_of_bitcoin. [Accessed 23 February 2025].

[6] C. R. Harvey, J. Hasbrouck and F. Saleh, "The Evolution of Decentralized Exchange: Risks, Benefits, and Oversight," July 2024. [Online]. Available: https://wifpr.wharton.upenn.edu/wp-content/uploads/2024/07/WIFPR-Decentralized-Exchange-Harvey-Hasbrouck-and-Saleh.pdf. [Accessed 24 February 2025].

[7] Moonpay, "Moonpay Docs," [Online]. Available: https://dev.moonpay.com/docs/on-ramp-architecture. [Accessed 21 02 2025].

[8] S. Hägele, "Centralized exchanges vs. decentralized exchanges in cryptocurrency markets: A systematic literature review," 18 May 2024. [Online]. Available: https://link.springer.com/article/10.1007/s12525-024-00714-2. [Accessed 28 February 2025].

[9] J. Solowey and J. J. Schulp, "Regulatory Clarity for Crypto Marketplaces Part II: Centralized Exchanges," 10 May 2023. [Online]. Available: https://www.cato.org/briefing-paper/regulatory-clarity-crypto-marketplaces-part-ii-centralized-exchanges. [Accessed 28 February 2025].

[10] V. Munene, "MoonPay Review: Ist dies die beste Fiat-zu-Krypto-Onramp-Plattform?," 23 January 2025. [Online]. Available: https://blockzeit.com/de/moonpay-review-ist-dies-die-beste-fiat-zu-krypto-onramp-plattform/. [Accessed 28 February 2025].

[11] S. Pala, A. Hillb, T. Rabehajad and M. Hitchense, "A Blockchain-Based Trust Management Framework with Verifiable Interactions," 10 June 2021. [Online]. Available: https://arxiv.org/pdf/2106.04885. [Accessed 2 March 2025].

[12] L. H. M. Hickey, "Decentralisation over Privacy: An Analysis of the Bisq Trade Protocol," Springer Nature Switzerland AG, 09 06 2022. [Online]. Available: https://doi.org/10.1007/978-3-031-06156-1_17. [Accessed 22 02 2025].

[13] T. Wang, J. Guo, S. Ai and J. Cao, "RBT: A distributed reputation system for blockchain-based peer-to-peer energy trading with fairness consideration," 7 May 2021. [Online]. Available:

https://www.sciencedirect.com/science/article/abs/pii/S0306261921005134. [Accessed 28 February 2025].

[14] K. Govindarajan, D. Vinayagamurthy, P. Jayachandran and C. Rebeiro, "Privacy-Preserving Decentralized Exchange Marketplaces," 20 December 2021. [Online]. Available: https://arxiv.org/pdf/2111.15259. [Accessed 28 February 2025].

[15] E. Budish, "Trust at Scale: The Economic Limits of Cryptocurrencies and Blockchains," 16 October 2024. [Online]. Available: https://academic.oup.com/qje/article/140/1/1/7824430?login=false. [Accessed 3 March 2025].

[16] F. H. Bappy, E. Cheon and T. Islam, "Centralized Trust in Decentralized Systems: Unveiling Hidden Contradictions in Blockchain and Cryptocurrency," 10 May 2025. [Online]. Available: https://arxiv.org/pdf/2505.06661. [Accessed 02 July 2025].

[17] M. Becker and B. Bodó, "Trust in blockchain-based systems," 20 April 2021. [Online]. Available: https://policyreview.info/pdf/policyreview-2021-2-1555.pdf. [Accessed 04 March 2025].

[18] C. Duley, L. Gambacorta, R. Garratt and P. K. Wilkens, "The oracle problem and the future of DeFi," 07 September 2023. [Online]. Available: https://www.bis.org/publ/bisbull76.pdf. [Accessed 15 March 2025].

[19] Paypal, "Paypal Developer," [Online]. Available: https://developer.paypal.com/docs/api/orders/v2/#orders_authorize. [Accessed 6 April 2025].

[20] "Stripe Docs," Stripe, [Online]. Available: https://docs.stripe.com/use-stripe-apps/netsuite/developer-hub?locale=de-DE. [Accessed 14 March 2025].

[21] "Mastercard Open Banking US," Mastercard, [Online]. Available: https://developer.mastercard.com/open-banking-us/documentation. [Accessed 17 March 2025].

[22] B. Cœuré, "Distributed ledger technology in payment, clearing and settlement," February 2017. [Online]. Available: https://www.bis.org/cpmi/publ/d157.pdf. [Accessed 19 March 2025].

[23] M. V. Nguyen, N. T. Le, D. Duong, Y. Li and M. Mukherjee, "Blockchain Oracles: Implications for Smart Contracts in Legal Reasoning and Addressing the Oracle Problem," 07 December 2023. [Online]. Available: https://dl.acm.org/doi/10.1145/3628797.3628870. [Accessed 23 March 2025].

[24] F. Zhang, E. Cecchetti, K. Croman, A. Juels and E. Shi, "Town Crier: An Authenticated Data Feed for Smart Contracts," 24 10 2016. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/2976749.2978326. [Accessed 28 2 2025].

[25] H. Xiong, Q. Wen and C. Wu, "A Privacy-Preserving Method for Blockchain Oracle Data Based on Improved Threshold Aggregated Signatures," 21 February 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10438260. [Accessed 20 March 2025].

[26] G. Caldarelli and J. Ellul, "The Blockchain Oracle Problem in Decentralized Finance—A Multivocal Approach," 09 July 2021. [Online]. Available: https://www.mdpi.com/2076-3417/11/16/7572. [Accessed 26 March 2025].

[27] Chainlink, "Chainlink Functions Documentation," 2023. [Online]. Available: https://docs.chain.link/chainlink-functions. [Accessed 1 3 2025].

[28] E. Foundation, "Ethereum Whitepaper," 12 February 2025. [Online]. Available: https://ethereum.org/de/whitepaper/. [Accessed 23 February 2025].

[29] A. M. Antonopoulos and G. Wood, Mastering Ethereum, O'Reilly Media, Inc., 2018.

[30] M. V. Brualla, "Blockchain Layer 2 scalability solutions: a framework for comparison," 2023. [Online]. Available: https://upcommons.upc.edu/server/api/core/bitstreams/94185bfa-9f6b-4c51-9c97-83874a8a2b1a/content. [Accessed 01 April 2025].

[31] R. Kapasi, "How to Improve API Security and Compliance with OAuth2.0 and mTLS," April 2025. [Online]. Available: https://www.nium.com/blog/how-to-improve-api-security-and-compliance-with-oauth20-and-mtls. [Accessed 20 April 2025].

[32] D. Malomo, "Do you Really Handle Webhooks Securely?," Flutterwave Engineering, 28 April 2025. [Online]. Available: https://dev.to/flutterwaveeng/do-you-really-handle-webhooks-securely-1hah. [Accessed 30 April 2025].

[33] Chainlink, "Introducing Chainlink Functions: Connect the World's APIs to Web3," 01 March 2023. [Online]. Available: https://blog.chain.link/introducing-chainlink-functions/. [Accessed 7 April 2025].

[34] OpenZeppelin, "Access Control," 2025. [Online]. Available: https://docs.openzeppelin.com/contracts/5.x/access-control. [Accessed 11 04 2025].

[35] Chainlink, "Data Feeds," 2025. [Online]. Available: https://data.chain.link/feeds. [Accessed 30 March 2025].

[36] ngrok, "ngrok DOCS," 2025. [Online]. Available: https://ngrok.com/docs. [Accessed 12 April 2025].

[37] Cointelegraph and Onramper, "Global Crypto Onramp Report," February 2023. [Online]. Available: https://veri-media.io/wp-content/uploads/2023/02/global-crypto-onramp-report.pdf. [Accessed 1 June 2025].

[38] D. R. Nalini and S. Yuvasri, "A Study on the Impact of Digital Transformation in the Banking Sector on Customer's Experience Dr. R. Nalini „," 2 November 2024. [Online]. Available: https://ijirem.org/view_abstract.php?title=A-Study-on-the-Impact-of-Digital-Transformation-in-the-Banking-Sector-on-Customer%E2%80%99s-Experience&year=2024&vol=11&primary=QVJULTE3Njg=. [Accessed 1 June 2025].

[39] J. Anglen, "Polygon vs Optimism vs Arbitrum: Comparing Ethereum Layer-2 Solutions," [Online]. Available: https://www.rapidinnovation.io/post/polygon-vs-optimism-vs-arbitrum-comparing-ethereum-layer-2-solutions. [Accessed 20 June 2025].

[40] "MiCA Regulation: How the EU is Shaping the Future of Crypto Asset Compliance?," 28 March 2025. [Online]. Available: https://amlwatcher.com/blog/mica-regulation-how-the-eu-is-shaping-the-future-of-crypto-asset-compliance/. [Accessed 13 June 2025].

[41] R. Zhang, J. Xie and N. Vaidya, "A Survey on Trust Management in Blockchain Systems," *Future Generation Computer Systems,* vol. 107, no. 1, pp. 841-853, 2020.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe. Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

10. August 2025, Stuttgart

Dominik Bepple Unterschrift: _____