MiniMoni: Combining HLS with Payword

Otávio Vacari Martins[1], **Marcos Antonio Simplicio Junior**[1]

[1] Escola Politécnica, Universidade de São Paulo, Brazil

{ovacari,msimplicio}@larc.usp.br

Abstract. *Existing micropayment systems struggle to achieve widespread adoption for continuous video streaming due to scalability limitations, security vulnerabilities related to token management, and tight coupling with specific video players. MiniMoni addresses these challenges by integrating the widely supported HTTP Live Streaming (HLS) protocol with PayWord, embedding payments directly into segment HTTP requests for atomic payment and delivery. The solution is implemented using player-agnostic integration and a secure browser extension for isolated token management, along with a three-step relay mechanism for secure inter-component communication. As such, MiniMoni provides a scalable and secure approach to continuous streaming monetization without requiring player-specific modifications.*

1. Introduction

The evolving landscape of digital content consumption requires the exploration of novel monetization strategies. Traditional models, which often rely on subscriptions or advertisements, are unappealing to many users. A more compelling alternative is to use micropayments, defined as small-value financial exchanges made on demand [Chaudhary and DaSouza 2024]. This strategy has the potential to unlock new economic opportunities, particularly in emerging domains such as machine-to-machine transactions involving data exchange or resource utilization [Klein et al. 2022].

Albeit promising, the actual deployment of micropayment systems faces some challenges. For example, even though Blockchain networks like Ethereum [Buterin et al. 2013] and Bitcoin [Nakamoto 2009] offer the promise of decentralized micropayments, implementing them is difficult due to high transaction costs and throughput limitations of direct on-chain transactions. Payment-oriented blockchains, such as XRPL [Chase and MacBrough 2018], and instant fiat payment systems (e.g., the Brazilian PIX) overcome these limitations with lower fees and faster processing. However, scenarios involving high frequency and volume payments may still overwhelm those systems and/or incur undesirable fees. This motivates approaches that enable direct payments between vendors and users, rather than relying on payment authorities for every transaction.

Payment channels, which allow parties to conduct numerous transactions off-chain and settle only the final state on-chain, offer a pathway to enabling continuous micropayments. By aggregating numerous individual micropayments into a single on-chain transaction, fees are minimized and throughput is optimized. There are many such proposals in the literature, including the Ethereum-based implementation of PayWord [Rivest and Shamir 1996] discussed in [Chiesa et al. 2019]. Despite these advances, realizing the potential of micropayments for real-time streaming applications requires addressing significant challenges that existing solutions have only partially solved.

1.1. Motivation: challenges with video streaming micropayments.

Integrating micropayments with streaming protocols requires precise synchronization to maintain user experience, as a failed payment validation results in denied access and halted playback. This creates strict latency requirements in which token generation, transmission, and validation must occur within the segment fetch window. These technical constraints highlight the need for optimized implementations that can satisfy both payment security and performance requirements.

For real-world adoption, the implementation should also be loosely coupled to support multiple HTTP Live Streaming (HLS) protocols, a *de facto* standard for multimedia streaming in the web. That might be accomplished by modifying the video player's internals, such as altering event handlers to create customized HTTP requests. However, this approach ties the payment system to specific players, making it difficult to support different HLS implementations. Proprietary players and variations (e.g, HLS using encryption for digital rights management) further complicate integration. Tight coupling limits portability, requires constant updates, and reduces compatibility with different streaming platforms, while a decoupled approach avoids such issues.

1.2. Contributions

The proposed system, MiniMoni, integrates continuous micropayments with video streaming by adopting an approach similar to the one proposed by [Chiesa et al. 2019] for Ethereum. MiniMoni achieves three key objectives:

1. A novel integration of the PayWord protocol with HLS, ensuring atomic payment and video segment delivery without modifying existing streaming infrastructure.
2. A player-agnostic implementation using service workers, enabling compatibility with any HLS player and simplifying deployment across streaming platforms.
3. A secure browser extension architecture that isolates and protects payment tokens, mitigating the risk of theft or manipulation while maintaining user sovereignty over their payment credentials.

2. Related Work

Continuous micropayment research spans protocol design, blockchain adaptations, protocol implementations, and application systems. A pioneering work in this area is PayWord [Rivest and Shamir 1996], which uses hash chains to minimize computational overhead through single root-hash signatures. More recently, though, much of the research has focused on blockchain-oriented solutions. This is the case of EthWord [Chiesa et al. 2019], which adapts PayWord for Ethereum smart contracts to maintain off-chain efficiency. Another example is the framework proposed in [Wang et al. 2023], which uses lockable signatures and off-chain processing with continuous microtransaction hash-chains (CMHC). For streaming-specific implementations, Liu et al. [Davies and Zhang 2022] created a Bitcoin-based payment channel leveraging UTXO transaction malleability for unilateral termination and per-packet payments. VoD-Coin [Angsuchotmetee and Kaewkandee 2018], in turn, offers a comprehensive decentralized video-on-demand system combining Ethereum contracts for payment processing with IPFS-based content distribution.

MiniMoni diverges from these approaches by directly augmenting existing adaptive streaming protocols rather than reconstructing the infrastructure. It specifically addresses the synchronization requirements between micropayments and adaptive bitrate streaming protocols (HLS/DASH). As such, it offers a lightweight integration solution that maintains compatibility with established content delivery networks while enabling granular payment mechanisms that align with media consumption patterns.

3. Architecture Overview

The core mechanism of MiniMoni is the same as PayWord: it relies on a hash chain $\{c_i\}_{i=0}^{n}$ of size n. To bootstrap the payment mechanism, Alice generates a secret random number s and iteratively applies a cryptographic hash function H to generate the hash chain. The chain is defined by $c_n = H(s)$ and $c_i = H(c_{i+1})$ for $i \in \{0, 1, ..., n-1\}$. Alice then commits c_0 to the smart contract as the anchor, locking x total funds without revealing any intermediate hashes c_i to Bob upfront. The smart contract logic ensures that, upon presentation of a valid hash c_i, Bob can unlock a portion of the total funds: $\frac{n-i}{n} \cdot x$, which corresponds to the revealed fraction of the hash chain from c_n down to c_i.

This approach provides distinct advantages for each participant. For vendors, the design enables rapid payment validation through simple hash operations, avoiding any overhead or fee that might result from frequently contacting the back-end settlement system. For issuers, payments can be calculated based on the hash position during withdrawal, saving processing time and bandwidth that would otherwise be invested in handling multiple transactions. For users, the system creates a convenient and automated way to ensure that vendors can only access tokens corresponding to the content consumed.

3.1. Components

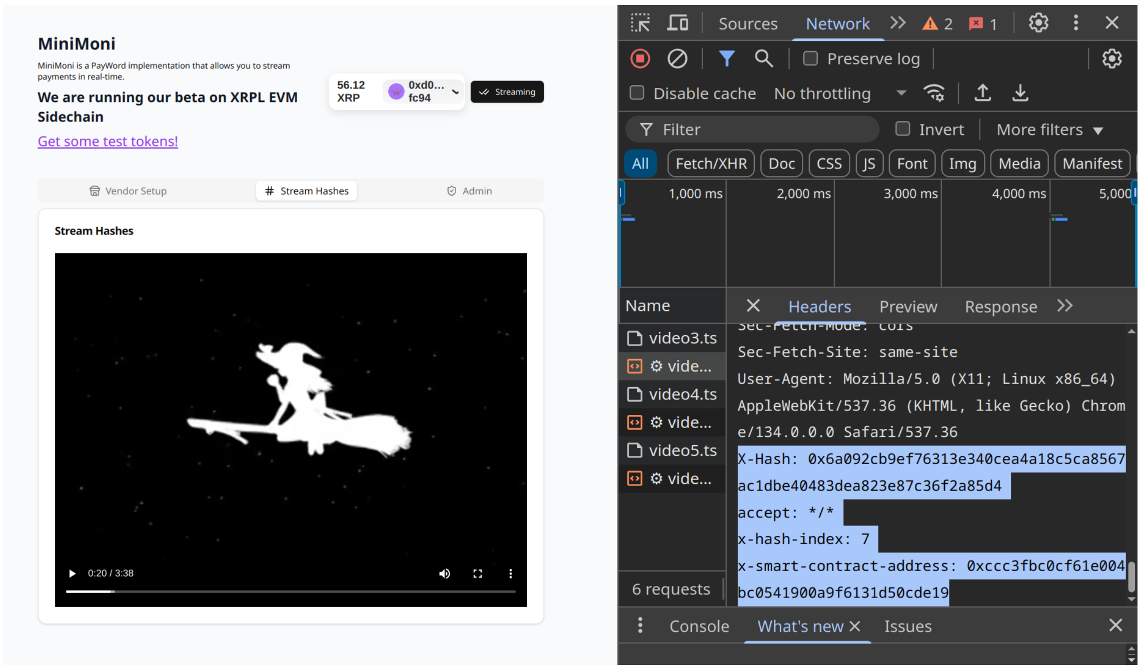
MiniMoni's architecture consists of four integrated components:

- **Web page:** Hosts the Video Player and registers a Service Worker that intercepts video segment requests, facilitating payment integration.
- **Browser Extension:** Securely stores payment tokens and manages access to them.
- **Vendor Server:** Provides video content and validates payment tokens before delivering multimedia segments.
- **Blockchain Smart Contract:** Handles payment channel initialization and final balance resolution, enforcing payment rules.

3.2. Operation

To initiate a payment channel, the user first retrieves the necessary vendor data, including the blockchain address and pricing information, from the vendor's server. Subsequently, the user compiles the smart contract directly within the browser environment. Using a commercial wallet such as MetaMask, the user signs the compiled contract, effectively locking the funds on the blockchain. Finally, the user notifies the vendor of the newly established payment channel and saves the smart contract address within the browser extension for future reference.

At the first payment, the vendor verifies the payment channel parameters by querying the blockchain state with the smart contract address provided in the request header


MiniMoni

MiniMoni is a PayWord implementation that allows you to stream payments in real-time.

We are running our beta on XRPL EVM Sidechain

Get some test tokens!

56.12 XRP

0xd0... fc94

Streaming

Stream Hashes

```
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-site
User-Agent: Mozilla/5.0 (X11; Linux x86_64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrom
e/134.0.0.0 Safari/537.36
X-Hash: 0x6a092cb9ef76313e340cea4a18c5ca8567
ac1dbe40483dea823e87c36f2a85d4
accept: */*
x-hash-index: 7
x-smart-contract-address: 0xccc3fbc0cf61e004
bc0541900a9f6131d50cde19
```

Figure 1. Video player (left) with corresponding network requests containing payment verification headers (right).

x-smart-contract-address. During this initial verification, the vendor also retrieves the tail of the hash chain, c_0, which serves as the anchor point for subsequent payment validations. MiniMoni synchronizes payments with video delivery by attaching custom HTTP headers to each segment request, as shown in Figure 1. The X-hash-index header indicates the position, j, of the incoming hash d_j within the chain.

Upon receiving a video segment request, the server validates the payment token in the X-Hash header. The vendor stores the previously accepted hash, c_i, and its index, i. For a new request with hash d_j and index j, the server retrieves c_i and i from its database. The server then computes $H^{j-i}(d_j)$. If $H^{j-i}(d_j) = c_i$, the payment is considered valid, and the requested video segment is delivered. Until the channel is closed, all micropayments can be validated without any additional blockchain query. Verifying the sequential validity of the hash chain allows this step to effectively prevent replay attacks and unauthorized token reuse, as an attacker's attempt to replay an old hash or use a future hash prematurely will result in a failed validation.

After the streaming (and payment) channel is permanently closed, the vendor initiates payment settlement by presenting the last valid hash (c_i) to the smart contract. The contract verifies that c_i is a valid predecessor of c_0 by iteratively hashing until reaching c_0. The withdrawn amount corresponds to the position of c_i in the hash chain, ensuring fair compensation proportional to the delivered content.

Usually, vendors would be the ones to initiate channel closure, aiming to redeem their due payment. That should happen after the end of the streaming, after the hash chain is fully consumed, or in other situations where the vendor-user interaction appears to have ended (e.g., if the user's media player stops requesting video segments for a long time). Nevertheless, after a reasonable validity period (e.g., 24 hours), the user can also close the channel. In either case, after channel closure, any locked funds not redeemed by the vendor are returned to the user.

4. Browser Extension

Our browser extension implements the client-side components of the hash chain payment system, managing secure token generation and communication with content providers. The extension architecture draws inspiration from Ethereum's EIP-6963[1] provider interface standard, which establishes patterns for wallet providers to communicate with web applications. While EIP-6963 uses window-based object injection for provider discovery, our approach employs message-passing. The extension implements the separation of concerns principle through three main components:

- **Content Scripts**: Executes in the context of web pages and serve as the communication channel between the page and the extension's background process.
- **Background Service**: Contain the core hash chain operations and authentication logic, handling token generation and storing hash chains in the browser's IndexedDB.
- **User Interface**: Implemented as a browser popup for hash chain administration and selection, as well as authentication status monitoring.

4.1. Security Model and Authentication

Since exposure of hash chain values or seed secrets would enable unauthorized payments and compromise the entire payment channel, the MiniMoni extension implements robust access controls. Specifically, the extension uses a two-tier access control system implemented via modal popup windows that require explicit user interaction:

- **Basic Access**: Permit non-sensitive operations including viewing available hash chains and their metadata, with a default validity period of 24 hours.
- **Secret Access**: Gate operations that require access to payment tokens, with user-selected time duration (typically minutes) during which the page may request multiple payment tokens without additional authentication prompts.

The background script is responsible for validating all authentication requests and enforcing access control policies before executing sensitive operations. For payment token requests, the extension primarily uses the **Pop Mode** method that retrieves only the specific hash needed for the current payment and automatically updates the index atomically to maintain synchronization with the provider server.

Extension components use the browser's messaging API for inter-process communication. Content scripts injected into web pages relay requests to the background service, which validates authentication status before processing operations. The extension stores hash chain data and authentication states in IndexedDB, leveraging browser data isolation. The browser's same-origin policy and extension isolation mechanisms prevent web pages from accessing the extension's IndexedDB directly, providing adequate protection for the implementation.

[1]https://eips.ethereum.org/EIPS/eip-6963

5. Bridging Isolated Contexts: A Three-Step Relay Mechanism

Although the browser's same-origin policy provides critical security isolation for our extension, it also creates communication challenges that must be overcome for the system to function. This section presents a relay mechanism that enables secure communication between a web page's service worker and an extension's privileged scripts while respecting browser-enforced security constraints. Same-Origin Policy (SOP) restricts scripts from one origin (e.g., `https://example.com`) from accessing resources or APIs in another origin (e.g., `chrome-extension://[id]`). In the MiniMoni system:

- **Service Worker**: Runs in the web page's origin (e.g., `https://example.com`) and lacks access to extension APIs or storage, since it operates in a separate execution context. It is designed to work exclusively with the web page that registered it, being SOP-compliant.
- **Web Page**: Shares the service worker's origin but cannot directly communicate with the extension's scripts. The web page exists in the website's origin context and serves as an intermediary, receiving requests from the service worker and forwarding them to the content script through DOM events like `window.postMessage`.
- **Content Script**: Operates in a hybrid context, sharing the DOM with the web page (same origin) but executing in an isolated JavaScript environment tied to the extension's privileges. This enables cross-origin message interception via DOM events like `window.postMessage` but no access to the web page's JavaScript variables, functions, or objects (including its service worker).
- **Background Script**: Executes in the extension's isolated origin (`chrome-extension://[id]`) with privileged API access, including capabilities such as secure storage (e.g., `chrome.storage`) and network requests (`chrome.runtime.sendMessage`).

To enable interaction across these isolated contexts, we design a relay mechanism that respects SOP constraints, summarized in Figure 2. The browser enforces strict origin separation, so communication must occur through APIs explicitly designed for cross-context messaging. The web page's DOM acts as a bridge, enabling messages to flow from the service worker to the content script and ultimately to the background script.

1. **Service Worker $\rightarrow$ Web Page**: To send messages to the web page, the service worker uses `clients.postMessage`. Since the service worker and web page share the same origin, this communication is allowed under SOP. The web page serves as a bridge, relaying messages to the content script.
2. **Web Page $\rightarrow$ Content Script**: The web page forwards messages to the content script via `window.postMessage`. The content script can listen to these messages because it shares the web page's DOM. This allows the content script to act as an intermediary between the web page and the extension's privileged scripts.
3. **Content Script $\rightarrow$ Background Script**: The content script uses `chrome.runtime.sendMessage`, an extension API restricted to the extension's origin. This ensures sensitive operations (e.g., token retrieval) occur only within the extension's isolated context.

The content script's dual context is essential: it listens to messages from the web page while leveraging extension APIs to bridge isolated origins. Its ability to interact with the

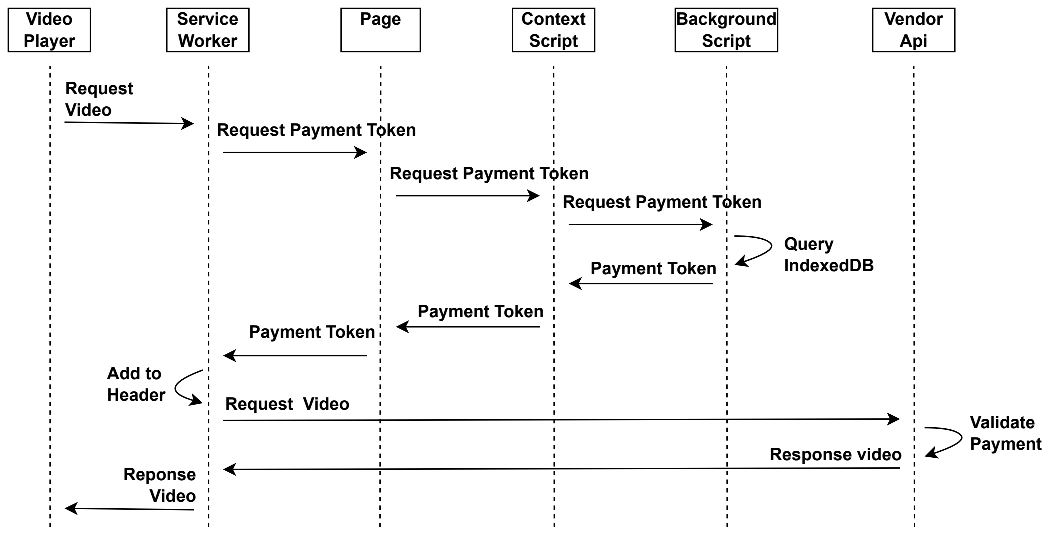


Figure 2. Sequence diagram illustrating the token-based authentication flow for video segment delivery in the MiniMoni system

DOM (unrestricted by SOP) enables it to relay messages from the web page's origin to the extension's origin, while SOP continues to enforce isolation for sensitive operations like network requests or storage access. This design ensures compliance with browser-enforced security boundaries while enabling functional interaction.

6. Performance Analysis

This section analyzes MiniMoni's effects on HLS streaming performance. To identify bottlenecks, we measured timing across client and server components, focusing on overhead patterns.

All performance measurements were conducted on a development machine equipped with a 13th Gen Intel(R) Core(TM) i7-1355U processor, 15GB of RAM, and a 476.9GB NVMe SSD running Ubuntu 24.04.2 LTS. The testing environment utilized Docker Engine 28.2.2 with Docker Compose v2.36.2 for containerized deployment of the MiniMoni system components. Client-side HLS streaming performance was measured using Google Chrome Version 137.0.7151.119.

As detailed in Table 1, the primary source of client-side delay stems from HLS fetch requests, with messaging partially contributing in part to the overall processing time. Through experimentation, we observed that pausing video playback for 60 seconds and resuming consistently triggers the higher delay range, suggesting service worker cold start issues as a key optimization target.

Category	Min (ms)	Max (ms)	Mean (ms)	Std. Dev. (ms)
Message Response Time	7.10	91.50	27.232	24.413
HLS Fetch Request Time	19.90	77.00	39.100	15.568
Total Processing Time	**36.10**	**127.70**	**66.591**	**23.314**

Table 1. Service Worker Performance Metrics

A more granular analysis of the server-side performance reveals that the main reason for delay is not the hash operations but rather the database operations, as shown in Table 2. Database interactions form the primary bottleneck, making their optimization critical for improving performance.

Category	**Min (ms)**	**Max (ms)**	**Mean (ms)**	**Std. Dev. (ms)**
Data validation	0.02	0.05	0.029	0.008
Channel query	1.51	5.52	2.768	1.054
Double-spend check	0.80	9.35	2.378	2.457
Latest payment query	2.38	5.35	3.518	1.066
Hashchain validation	0.08	0.19	0.130	0.035
Payment creation & storage	5.30	16.30	9.947	4.398
Total Validation Time	**11.39**	**25.90**	**19.387**	**5.176**

Table 2. Server-Side HLS Access Validation Performance

The overall processing overhead per segment consistently remains under 150ms. For typical 5-10 second HLS segments, the player's buffer effectively absorbs this slight delay, ensuring no noticeable effects on the user experience.

7. Demo Description

Our demonstration provides a complete local implementation of the streaming payment system, enabling users to experience the full workflow from vendor onboarding to payment settlement. The setup utilizes Hardhat[2] for blockchain simulation, MetaMask[3] for wallet integration, all integrated with our system. The demo walkthrough includes four key stages: vendor registration and subscription, payment channel establishment, real-time payment streaming, and channel closure with final settlement. A public repository[4], an installation video[5], and a demonstration video[6] are available to provide complete access to the code and guide users through the setup and usage process.

8. Conclusion

MiniMoni offers a practical and secure micropayment system for video streaming, addressing scalability, security, and player compatibility challenges. By integrating the PayWord protocol with HLS through a service worker architecture, MiniMoni achieves atomic payment and delivery with minimal latency. A secure browser extension manages tokens, while a three-step relay ensures secure communication within browser security policies. Future research includes optimizing the validation data flow on the server side and reducing messaging delays. Additionally, exploring the integration of alternative payment protocols within the architecture and evaluating the business viability of the solution are also key areas of focus.

Acknowledgment. This study was financed in part by the Brazilian FAPESP (grant 2020/09850-0), CNPq (grant 307732/2023-1), and CAPES (Finance Code 001).

[2]Hardhat is a development tool for Ethereum applications. Available at: `https://hardhat.org/`

[3]MetaMask is a commercial cryptocurrency wallet. Available at: `https://metamask.io/`

[4]Public Repository: `https://github.com/otaviootavio/sbseg-sf-minimoni`

[5]Installation Video: `https://youtu.be/TDhcyu8ikEI`

[6]Demonstration Video: `https://youtu.be/bpFZVlYnUa0`

References

[Angsuchotmetee and Kaewkandee 2018] Angsuchotmetee, C. and Kaewkandee, P. (2018). VoDCoin: a cryptocurrency-based architecture for a decentralized-based video-on-demand service. In *Proc. of the 10th Int. Conf. on Management of Digital EcoSystems*, pages 100–105, New York, USA. ACM.

[Buterin et al. 2013] Buterin, V. et al. (2013). Ethereum white paper. *GitHub repository*, 1(22-23):5–7.

[Chase and MacBrough 2018] Chase, B. and MacBrough, E. (2018). Analysis of the XRP ledger consensus protocol. Preprint: arXiv:1802.07242.

[Chaudhary and DaSouza 2024] Chaudhary, P. and DaSouza, R. O. (2024). Consumer readiness for microtransactions in digital content business models. *Businesses*, 4(3):473–490.

[Chiesa et al. 2019] Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N., and Ward, N. (2019). Deploying PayWord on Ethereum. In *Workshop on Trusted Smart Contracts (WTSC)*. Financial Cryptography and Data Security Conference.

[Davies and Zhang 2022] Davies, J. and Zhang, W. (2022). Data streaming over Bitcoin payment channels. In *2022 IEEE Future Networks World Forum (FNWF)*, pages 255–261.

[Klein et al. 2022] Klein, M., Kundisch, D., and Stummer, C. (2022). Feeless micropayments and their impact on business models. *Handbuch Digitalisierung*, pages 799–814.

[Nakamoto 2009] Nakamoto, S. (2009). Bitcoin: A peer-to-peer electronic cash system.

[Rivest and Shamir 1996] Rivest, R. L. and Shamir, A. (1996). PayWord and MicroMint: Two simple micropayment schemes. In *International Workshop on Security Protocols*, pages 69–87. Springer.

[Wang et al. 2023] Wang, W., Chen, G., Chu, C., and Lan, W. (2023). A blockchain-based continuous micropayment scheme using lockable signature. *Mathematics*, 11(16):3472.