BitcoinBridge.app: Bridging Bitcoin, Enhancing Efficiency

Electron Team

September 3, 2025

1 Introduction

Bitcoin, as the cornerstone of the crypto ecosystem, boasts the largest market capitalization and user base. However, moving BTC across different blockchains remains an expensive and inefficient process. Current bridging solutions typically incur transaction fees of approximately 1% per swap, a prohibitive cost in a maturing market. For active participants, these fees accumulate rapidly: a $\$100,000$ transfer alone loses $\$1,000$, and frequent bridging (e.g., 20 times annually) could deplete up to 20% of a user's portfolio.

Existing Bitcoin bridges face two critical limitations. First, they have slow and insecure expansion because they rely on custom smart contracts for each new chain. This leads to long rollout times, expensive audit cycles, and an expanding security attack surface. Second, they suffer from poor capital efficiency due to fragmented liquidity pools. This inefficiency forces bridges to offer inflated Annual Percentage Yields (APYs) to attract idle capital, which increases user fees and limits scalability.

BitcoinBridge.app was engineered to fundamentally transform this landscape, drawing on Electron Team's extensive, long-standing expertise expertise in Zero-Knowledge Proofs (ZKPs) and Trusted Execution Environments (TEEs). By integrating TEEs, threshold cryptography, and a proprietary dynamic liquidity engine, *BitcoinBridge.app* directly addresses the core challenges inherent in existing BTC bridges. This architecture eliminates the need for individual smart contracts on every chain by executing all bridge logic securely within TEEs, simplifying deployment across any blockchain.

2 Problem Statement

Existing cross-chain bridges face slow and costly expansion due to their reliance on custom smart contracts for each new chain. This necessitates lengthy development and audit cycles, limiting scalability and increasing security risks.

3 Bridge Protocol Overview

The bridge operates with an Externally Owned Account (EOA) on each supported blockchain. To initiate a cross-chain transfer, a user first sends their desired asset to the designated EOA on the source chain. Subsequently, they sign a message detailing the bridge specifics, such as the destination chain and asset. For Bitcoin as the source chain, this secondary signing step is not required.

While the EOA manages user assets and facilitates cross-chain transfers, relying on a single private key for transaction signing introduces an obvious security vulnerability and a central point of failure. To mitigate this risk, the bridge employs a robust combination of Trusted Execution Environments (TEEs) and a Threshold Signature Scheme.

3.1 What is a $\{t, n\}$-threshold ECDSA scheme?

A $\{t, n\}$-threshold ECDSA scheme allows a group of 'n' participants to collectively generate a valid ECDSA signature, provided that at least '$t + 1$' of these participants collaborate. This means no single party has full control over the signing key.

3.2 What is a TEE?

A Trusted Execution Environment (TEE) is like a secure, locked box inside your computer's main chip. It ensures that certain programs and their data are run exactly as intended, without any tampering. Additionally, it keeps sensitive information hidden and protected from anything else on the computer, even if other parts of the system are compromised.

4 Solution Overview

In this architecture, each of the 'n' participants operates a client inside a dedicated Trusted Execution Environment (TEE). Each client is provisioned with a unique, user-specific private key share, which remains inaccessible and unreadable by any party, including the user themselves.

When a transaction requires signing as part of the bridging process, each participant's TEE-secured client autonomously generates its individual signature share. These shares are then securely collected by one of the user's clients and combined within a secure environment to reconstruct the complete, valid transaction signature. Prior to final submission, the transaction's details are rigorously verified for correctness, ensuring integrity before broadcast.

5 *BitcoinBridge.app* Achievements

The bridge delivers unparalleled security, efficiency, and scalability through its innovative design:

- **Enhanced Security & Integrity:** Parties cannot collude to forge or send fraudulent transactions. This is guaranteed because private key shares are inaccessible within TEEs, and every transaction undergoes a rigorous correctness check before being signed and sent.
- **High Availability:** The system maintains continuous operation even if some nodes go offline, thanks to the inherent fault tolerance provided by the Threshold Signature Scheme.
- **Rapid Expansion:** The bridge can expand to support new blockchain networks in approximately 2 days, eliminating the lengthy development and audit cycles associated with traditional smart-contract-based bridges.
- **Drastically Reduced Fees:** Users benefit from transaction fees of just 0.1%, representing a 10x improvement compared to prevailing industry standards.
- **High Volume Capacity:** The platform is engineered to efficiently handle substantial demand, equipped by a proprietary liquidity engine to manage over $1.5 billion in monthly volume.

Appendix A Threshold ECDSA Mathematical Details

ECDSA is a digital signature scheme used by both Ethereum and Bitcoin for signing transactions. We here lay out the core mathematics behind a $\{t, n\}$-threshold ECDSA scheme. We first review Shamir secret sharing and multiplicative-to-additive share conversion, then describe key generation and distributed signing.

Appendix A.1 Shamir secret sharing

The underpinning concept here is that of polynomial interpolation.

A polynomial of degree t can be defined uniquely by any set of $t + 1$ points on it. For instance, $y = 3x^2 + 7x + 4$ can be recovered uniquely by three points: $(-1, 0)$, $(0, 4)$, $(2, 30)$.

Now, let there be a dealer who constructs a polynomial of degree t with random coefficients taken from some finite field.

$$f(x) = sk + a_1 x + \cdots + a_t x^t \tag{1}$$

The dealer prepares the values $(1, f(1)), (2, f(2)), \ldots, (n, f(n))$ and shares them with the n players, respectively. Each player receives their share and learns nothing about the other shares. Any t players can now jointly reconstruct the polynomial, thereby recovering the secret sk.

Appendix A.2 Feldman's verifiable secret sharing (VSS)

Feldman's Verifiable Secret Sharing (VSS) allows a dealer to distribute shares of a secret in such a way that participants can verify the integrity of their shares without revealing the secret itself. This ensures that the dealer has distributed valid shares and prevents malicious players from disrupting the secret reconstruction process.
To achieve this verifiability, the dealer first commits to the coefficients of the polynomial used for Shamir's Secret Sharing. For a polynomial $f(x) = sk + a_1 x + \cdots + a_t x^t$, the dealer computes and broadcasts a set of commitments $C_j = g^{a_j}$ for $j = 0, \ldots, t$, where $a_0 = sk$. Each participant P_i, upon receiving their share $f(i)$, can then verify its correctness by checking the following equation:

$$g^{f(i)} = \prod_{j=0}^{t} (C_j)^{i^j} \pmod{q} \tag{2}$$

Appendix A.3 Multiplicative shares to Additive shares (MtA)

Suppose Alice and Bob hold two secrets $a, b \in \mathbb{Z}_q$ respectively, which we can think of as multiplicative shares of a secret $x = ab \pmod{q}$. In the following, we will learn how both parties can get random additive shares α and β, respectively, such that $\alpha + \beta = ab \pmod{q}$.

We assume that Alice is associated with a public key E_A for an additively homomorphic scheme over an integer N (Paillier cryptosystem), where $\mathrm{E}_A(.)$ denotes the encryption algorithm using public key A.

The protocol:

1. Alice sends $c_A = \mathrm{E}_A(a)$ to Bob.
2. Bob computes the ciphertext $c_B = \mathrm{E}_A(ab + \beta')$ (due to the additive homomorphic scheme), where β' is chosen uniformly at random in $\mathbb{Z}_{q^5}$. Bob sets his share $\beta = -\beta' \pmod q$ and sends c_B to Alice.
3. Alice decrypts c_B to obtain α' and sets $\alpha = \alpha' \pmod q$.

Appendix A.4 Threshold signature scheme

Appendix A.4.1 ECDSA signing

The Public Parameters consist of a cyclic group G of prime order q, a generator g for G, a hash function $H : \{0,1\}^* \to \mathbb{Z}_q$, and another hash function $H' : G \to \mathbb{Z}_q$.

There is a private key x chosen uniformly at random in $\mathbb{Z}_q$, and a public key $y = g^x$ computed in G. To sign a message M, the signer computes $m = H(M) \in \mathbb{Z}_q$, chooses k uniformly at random in $\mathbb{Z}_q$, and computes $R = g^{k^{-1}}$ in G and $r = H'(R) \in \mathbb{Z}_q$. Then she computes $s = k(m + xr) \pmod q$. The signature on M is the pair (r, s) which is verified by computing $R' = g^{ms^{-1} \pmod q} y^{rs^{-1} \pmod q}$ in G and accepting if $H'(R') = r$.

We'll learn how to jointly compute the signature (r, s) by using additive shares of it distributed among participants. The technical complication here comes from having to jointly compute R (which requires raising g to the inverse of a secret value k) and to compute s which requires multiplying two secret values k, x.

Appendix A.4.2 Key generation protocol

Phase 1 Each Player P_i selects $k_i, \gamma_i \in \mathbb{Z}_q$ and broadcasts a commitment to g^{γ_i} along with their Paillier public key E_i.

Phase 2

1. Each P_i decommits, revealing g^{γ_i}.
2. Using Feldman's VSS, P_i splits $gamma_i$ into n (t, n) Shamir shares and sends each player their share.
3. Every player collects one share from each P_i and adds them all up to get their final share x_i.
4. The resulting values x_i are a (t, n) Shamir's secret share of the secret key $x = \sum_i \gamma_i$.
5. The public key is set to $\prod_i g^{\gamma_i} = g^x$.

Appendix A.4.3 Signature Generation

We assume that $t+1$ participants are involved in the signing protocol, as this is sufficient for it to work, although the protocol can also handle larger groups.

Each participant converts their Shamir share, x_i, to additive shares, w_i, using Lagrangian coefficients. $w_i = \lambda_{i,S}(x_i)$, i.e., $x = \sum w_i$.

Phase 1

1. Define $k = \sum k_i$ and $\gamma = \sum \gamma_i$.

2. Note, $k\gamma = \sum_{i,j} k_i\gamma_j \pmod q$ and $kx = \sum_{i,j} k_i w_j \pmod q$.

Phase 2

1. Every pair of players P_i, P_j runs MtA with shares k_i, γ_j respectively to work on $k_i\gamma_j = \alpha_{i,j} + \beta_{i,j}$. At the end, P_i ends up with $\delta_i = k_i\gamma_i + \sum_{j\neq i}\alpha_{i,j} + \sum_{j\neq i}\beta_{j,i}$, which is the additive share of $k\gamma = \sum \delta_i$.

2. Similarly, running MtA on $k_i w_j$, they end up with $\sigma_i = k_i w_i + \sum_{j\neq i}\mu_{i,j} + \sum_{j\neq i}\nu_{j,i}$, which is the additive share of $kx = \sum \sigma_i$.

Phase 3 P_i broadcasts δ_i and the players reconstruct $\delta = \sum \delta_i = k\gamma$ and compute $\delta^{-1} \pmod q$.

Phase 4 Using every player's decommitment, the players compute $R = [\prod_i g^{\gamma_i}]^{\delta^{-1}} = g^{\gamma k^{-1}\gamma^{-1}} = g^{k^{-1}}$ and $r = H'(R)$.

Phase 5 Each player P_i sets $s_i = mk_i + r\sigma_i$. Then $s = \sum s_i = \sum(mk_i + r\sigma_i) = m\sum k_i + r\sum \sigma_i = mk + rkx = k(m + xr)$.

The signature (r, s) is generated without exposing either the private key x or the nonce k to any single party, and without reconstructing those values. From this point, the signature is complete and can be used in the standard way.

Appendix A.5 Note

Additional security measures like range proofs, extended Phase 5 for checking invalid signatures from any adversary, and some more ZK proofs are implemented in the full protocol but are beyond the scope of this document.