Side Protocol:

A Financial Infrastructure for Bitcoin

Side Labs

v1.01, November 15th, 2024

Table of Content

Abstract

Bitcoin is widely recognized as both a store of value and a payment medium; however, its potential remains underutilized as demand for DeFi applications leveraging Bitcoin continues to grow. Much like traditional banks, which provide a spectrum of financial services based on fiat currencies and real-world assets, there is a need for decentralized infrastructure that offers diverse financial services for Bitcoin, with lending as a primary focus. Despite Bitcoin's decade-long presence, no project has yet fully realized this potential. Side Protocol addresses this gap as an extension layer designed specifically for Bitcoin, enabling a new generation of decentralized financial applications within a Bitcoin-centered internet.

1. Introduction

At the core of Side Protocol is Side Finance, a non-custodial, liquidity-based lending system for Bitcoin that eliminates the need for third-party custody of BTC used as collateral. Side Finance is a cross-chain solution that utilizes an alt-chain system, Side Chain, a fully Bitcoin-compatible appchain designed specifically as an extension layer to scale Bitcoin's programmability. In this paper, we focus primarily on introducing these two components.

2. Side Finance Preliminary

This litepaper provides technical details, but the core principles of the Side Finance lending protocol are straightforward:

- Not your keys, not your coins: While BTC can be used as collateral to borrow other assets, the collateralized BTC cannot be arbitrarily spent by lenders under any circumstances unless liquidation occurs. The protocol enforces this principle by securing all BTC collateral in 2-of-2 multi-signature addresses, which require the borrower's signature for any transaction. This ensures that the protocol offers a Bitcoin-native level of security to its users.
- Liquidity-based: A few peer-to-peer non-custodial lending protocols have already emerged and are in production. However, when comparing with Ethereum and reflecting on the history of DeFi development, P2P products prove to be inefficient, lacking product-market fit. Leading DeFi protocols are almost all liquidity-based, which are more automated and unlock new possibilities for DeFi composability.
- Alt-chain risk remains with alt-chain participants: No crypto loan system can entirely eliminate risk, especially when loan assets involve trusted entities, such as stablecoins (e.g., USDT).

However, the protocol is designed to minimize economic risks for liquidity providers, ensuring a trust-minimized environment where risks are managed within the alt-chain ecosystem without adversely affecting borrowers.

Side Finance leverages established Bitcoin native technologies, including Schnorr-based Adaptor Signatures, HTLCs, Taproot, and threshold signing, integrated with Discreet Log Contracts (DLCs) and secure distributed oracles. The combination of these technologies into Scriptless Scripts enables Side Finance to provide smart contract-like functionality on Bitcoin today, without requiring any changes to the underlying Bitcoin protocol's opcodes.

2.1 Threshold Adaptor Signatures

Adaptor signatures [1] represent a significant cryptographic advancement that enhances the efficiency and privacy of conditional transactions within the Bitcoin network. These signatures serve as a bridge between standard signatures and concealed values, offering dual functionality:

- They disclose a secret value when combined with the corresponding signature.
- They generate the complete signature when presented with the secret value.

A notable feature of adaptor signatures is their reusability: third parties can create secondary adaptors from the initial commitment, even without knowledge of the secret value. This characteristic renders them particularly effective for establishing conditional locks in Bitcoin contracts.

Traditionally, Bitcoin contracts employ hashlocks for conditional payments, ensuring atomicity across multiple transactions. While effective, hashlocks have certain limitations, including their on-chain storage footprint and the ability to link transactions that share the same hash across different blockchains.

Adaptor signatures provide developers with greater flexibility, enabling the creation of more intricate conditional structures without increasing on-chain data requirements.

2.2 Discreet Log Contracts (DLCs)

Central to Side Finance's infrastructure is the Discreet Log Contract (DLC) [2], an oracle-based Bitcoin smart contract framework designed to enhance Bitcoin's programmability. DLCs allow for conditional payments based on off-chain events, all without involving a third-party custodian to manage funds.

These contracts leverage multiple cryptographic methods, including multi-signature transactions, Hash Time-Locked Contract (HTLC) [3], Schnorr signatures [4], and adaptor signatures, ensuring the security and non-custodial nature of the system.

A DLC begins with a funding transaction that locks funds from two parties into a 2-of-2 multisignature output. Pre-signed Contract Execution Transactions (CETs) [5] are created for different potential outcomes of the external event, but these CETs remain inactive until a real event outcome occurs.

The oracle, which is independent of the contract parties, publishes a public key or nonce at the contract's initiation. Participants use this key to generate adaptor signatures for the CETs. When the oracle reveals the outcome by disclosing a signature, the participants can finalize the relevant CET and broadcast it to the Bitcoin network, redistributing the locked funds according to the contract's terms.

DLCs provide a secure, trustless method to execute conditional payments and smart contract logic, pushing Bitcoin's capabilities for DeFi applications.

3. Side Finance Architecture

3.1 Definitions

3.1.1 Participants

Borrower: The party receiving the loan by using BTC as Collateral

Liquidity Provider: Users supplying the loan through the Lending Contract

Lending Contract: A smart contract deployed on the Side Chain that automates the operations of the lending pool. This contract enables liquidity providers to supply assets for lending and earn rewards in return. While the Lending Contract itself does not have the capability to sign transactions, it delegates this function to the Distributed Collateral Agent (DCA), which signs transactions on behalf of the contract and in collaboration with the borrower.

DCA (Distributed Collateral Agent): A decentralized network of operators organized into a threshold adaptor signature scheme. DCA operators sign Bitcoin 2-of-2 multi-sig transactions on behalf of the Lending Contract, with the counterparty being the Borrower. Additionally, the DCA manages Liquidated Assets in the event of a liquidation.

Oracle Operators: Monitor the price of BTC from cryptographically signed upstream sources, signing attestations at predetermined intervals. To maintain system integrity, Side Finance implements a mechanism that discards prices falling outside a pre-established variance level. This approach ensures that in the case of problems with a specific oracle, the safety of the system is maintained. Side Finance utilizes multiple independent cryptographically signed price sources to provide outcome attestations for its DLCs.

3.1.2 Glossary

Collateral Vault: A Bitcoin Taproot address where borrowers send their BTC as collateral for lending. Each loan has its own unique vault address designated for its collateral

Repayment Escrow: An escrow vault where the borrower locks their loan repayment on the Side Chain

Collateral: The BTC assets pledged by the Borrower to borrow the loan

Principal: The original amount of assets borrowed

Maturity Time: The deadline by which the Borrower must repay the loan in full. If the Borrower fails to repay by this date, the DCA may liquidate the Collateral to recover the outstanding debt

Liquidated Assets: Collateral liquidated and sent to the DCA due to the borrower's failure to fulfill payments on the principal and interest of the loan before Maturity Time

Loan Default: The failure to repay a loan by the Maturity Time, which can result in the liquidation of Collateral

Liquidation Price: The BTC price at which a loan becomes undercollateralized, triggering the execution of CETs

Final Timeout: The specified Bitcoin blockchain height after which the Borrower is entitled to reclaim the collateral if the DCA or Side Chain becomes unresponsive.

3.2 Workflow Overview

Side Finance defines a liquidity pool-based lending protocol that provides loans to BTC holders and generates returns for loan providers. The loan assets are pooled in smart contracts on the Side Chain and offered to borrowers without involving any third party holding the collateral during the loan period. At a high level, the protocol proceeds as follows:

1. Loan Assignment
    a. The borrower sends a loan request to the Lending Contract, including a hash of `loan_secret` and the Maturity Time to initiate the loan. The Lending Contract then assigns a Collateral Vault that is specifically created for the Borrower to lock their collateral.
    b. The Lending Contract computes the interest rate and liquidation price based on the current market price, and pre-allocates the loan amount from available liquidity for the Borrower.
    c. To claim the loan, the borrower deposits the BTC collateral into the assigned Collateral Vault and submits the `loan_secret` along with pre-signed Contract Execution Transactions (CETs) and an adaptor signature to the Lending Contract. One of the CETs will be executed in the event of liquidation triggered by the depreciation of the BTC collateral value.
    d. The Lending Contract verifies the validity of the adaptor signature and checks whether the hash of `loan_secret` in the Bitcoin lock script from the previous step matches the one in the loan request to the Lending Contract. If the adaptor signature is valid and the hashes match, the borrower is able to claim the pre-allocated loan amount. The BTC collateral remains securely locked in the Collateral Vault until the loan reaches the Maturity Time.
2. Repayment
    a. At some point before the Maturity Time, the borrower initiates repayment. At that time, the Borrower generates a `repayment_secret` and sends the loan amount with interest to the Repayment Escrow, using the hash of `repayment_secret` and a Partially Signed Bitcoin Transaction (PSBT).

b. The Lending Contract verifies the repayment and produces a SignRequest for the DCA. The DCA then retrieves the required details and partially signs the adaptor signature of the Bitcoin transaction, submitting it back to the Lending Contract. The contract aggregates these partial signatures from both the Borrower and the DCA into a complete signature.
c. With the aggregated signature and the `repayment_secret`, the borrower can now claim the collateral locked in the Collateral Vault. When the borrower claims the collateral, the `repayment_secret` is revealed.
d. Once the `repayment_secret` is revealed, the DCA (or any user) can withdraw the repaid loan funds from the Repayment Escrow into the Lending Contract, finalizing the process.

3. Liquidation
    a. see Section 3.5
4. Final Timeout
    a. A `final_timeout` protects the Borrower in the event that the Lending Contract or Side Chain becomes unresponsive.

In the following sections, we will provide more details of each step involved in the lending process as outlined above.

3.3 Loan Assignment

The Borrower sends a loan request to the Lending Contract to initiate a loan.

```
Unset
{

"borrower":"bc1p5d7rjq7g6rdk2yhzks9smlaqtedr4dekq08ge8ztwac72sfr9rusxg3297",
      "maturityTime": "2024-10-10T10:10:10z",
      "hashLock": "hash of loan_secret"
}
```

The Lending Contract then assigns a unique vault, the Collateral Vault, for the loan. Expressed in a Miniscript-like pseudocode, the lock script for this vault is as follows:

```
Unset
or(
    // case 1 - the DCA and Borrower can collaborate to create CETs
    and(
        pk(borrower_pk),
        thresh(T, pk(dca0_pk),pk(dca1_pk),...pk(dcaN_pk))
    ),

    // case 2 - the Borrower can reclaim collateral after
    // loan repayment using an adapted signature.
    // the aggregated adaptor signature was created
    // by the Borrower and DCA.
    // Its revelation unlocks loan repayment acceptance
    // in the smart contract.
    and(
        pk(borrower_pk),
        adaptor(adapted(repayment_secret))
    ),

    // case 3 - DCA owns collateral after `maturityTime`
    and(
        hash256(loan_hash),
        after(maturityTime),
        thresh(T, pk(dca0_pk),pk(dca1_pk),...pk(dcaN_pk))
    ),

    // case 4 - collateral reverts to Borrower after final timeout
    and(
        pk(borrower_pk),
        after(final_timeout)
    )
)
```

The Collateral Vault serves as the foundation for all collateral-related operations. In this paper, we will examine each spending condition in detail. For now, here are some initial observations to help build understanding.

The DCA multi-sig participants are indexed from `0..N`, with a threshold `T` of signatures required to spend.

The aggregated adaptor signature setup in the script above merits further examination. We can formally describe the Adaptor Signature process as follows. Let λ represent the secret scalar, and G be the base point of the elliptic curve. The adaptor point A is computed as $A = \lambda G$. Let m denote the message, which is the SHA-256 hash of the Loan Request JSON data: $m = SHA256(LR\_JSON)$

Given a private key sk, a nonce seed η, and the adaptor point A, we define the adaptor signature $\sigma_A = SignAdaptor(sk, m, \eta, A)$.

The function $SignAdaptor$ represents the cryptographic operation to create an adaptor signature. To redeem the signature, we use the secret λ to adapt σ_A, producing the final signature $\sigma = Adapt(\sigma_A, \lambda)$.

In this formulation:

- $\lambda \in Zq$ (the scalar field of the curve)
- $A, G \in E(Fq)$ (points on the elliptic curve)
- $m \in \{0,1\}^256$ (256-bit hash output)
- σ_A, σ are elements of the signature space

In summary, from the Bitcoin chain's perspective, the resulting signature appears as a standard Schnorr signature. However, this scheme modifies the signing process to produce a special adaptor signature, which can embed a secret to be revealed at a later stage. This adaptor signature enables trustless loan repayment by leveraging *Scriptless Scripts* within the Bitcoin framework.

The Lending Contract also needs to compute the interest and liquidation price based on the current price and pre-allocate funds for this loan, representing it as the Loan Request LR_JSON:

```
Unset
{

"borrower":"bc1p5d7rjq7g6rdk2yhzks9smlaqtedr4dekq08ge8ztwac72sfr9rusxg3297",
      "maturityTime": "2025-10-10T10:10:10z",
```

```
        "hashLock": "465b1a66c9f386308e8c75acef9201f3f577811da09fc90ad",
        "borrowAmount": "20000",
        "vaultAddress":
"bc1p6r4u4qlajya6feu337gtngksgeyf4nf0mf4q35gtcj5v0ja9m00q3eagh3",
        "Oracles": ["Oracle 1", "Oracle 2", "Oracle 3"],
        "currentPrice": "50000.00",
        "collateralAmount": "1",
        "createAt": "2024-10-10T10:10:10z"
}
```

If the Borrower agrees to the terms, they can send the BTC collateral to the Collateral Vault. Once confirmed on Side Chain, the Borrower outputs and signs CETs for a Bitcoin Discreet Log Contract, exchanging them for counter-signing with the DCA. The borrower also creates an adaptor signature over the above `LR_JSON` contract message.

In code, the adaptor signature generation process is as follows:

```
Unset
let adaptor_point = secret.base_point_mul();
let message = sha265(lr_json);
let adaptor_signature = sign_adaptor(seckey, message, nonce_seed,
adaptor_point);
let redeem_signature = adaptor_signature.adapt(b"loan_secret");
```

The Borrower then submits the CETs, adaptor signature, and `redeem_signature` to the Lending Contract on the Side Chain to claim loan assets, such as USDC.

Upon verifying the adaptor signature, the `loan_secret` is revealed from the `redeem_signature`. If the hash of this secret matches the one provided in the original Loan Request, the Borrower can claim the loan previously allocated to them.

```
Unset
// Lending Contract
```

```
let verified = verify(borrower_pubkey, adaptor_signature, message,
adaptor_point);
assert_eq(sha256(b"loan_secret"), hashLock)
let revealed_secret = adaptor_signature.adapt(redeem_signature)
if sha256(revealed_secret) === <hashLock>  {
        // send USDC to the borrower
}
```

At this stage, the collateral is securely locked in the Collateral Vault, and the Borrower has received the loan. The Borrower may repay the loan at any time before the Maturity Date.

3.4 Repayment

The borrower must repay the loan before the Maturity Time to avoid liquidation by the DCA.

Recall that during Loan Assignment, BTC collateral for the loan was locked into the Collateral Vault. One of the spending conditions is a 2-of-2 multi-sig UTXO where one side is the DCA, and the other side is the Borrower, with an aggregatable adaptor signature scheme, allowing a signature from the Borrower to publicly reveal a `repayment_secret`.

Assuming the loan is denominated in USDC, the borrower must submit a transaction to the Lending Contract on the Side Chain that includes:

1. A USDC transfer from the Borrower to the Lending Contract to repay the loan principal and interest.
2. A partially signed Bitcoin transaction to withdraw the BTC collateral

The repaid USDC funds are held in a Repayment Escrow on the Side Chain, secured by a hash of a randomly generated `repayment_secret` from the Borrower. Anyone who provides the pre-image of this hash can transfer the funds from the Repayment Escrow back to the liquidity pool.

The USDC repayment is subject to a timelock (e.g., 24 hours). If the loan repayment is not completed before the timelock expires, the transaction will be automatically canceled, and the funds will be returned to the Borrower.

```
Unset
let borrower_signature = sign_partial(
        agg_pubkey,
        sec_key,
        repayment_adaptor_point,
        message
);

function lock_repayment(hash_of_repayment_secret) {
        loan.repayment_hash_lock = hash_of_repayment_secret
}
```

The Lending Contract verifies the repayment and produces a SignRequest. The DCA then fetches and partially signs the adaptor signature of the Bitcoin transaction and submits it to the Lending Contract. The contract aggregates these partial signatures from both the borrower and the DCA into an aggregated signature.

```
Unset
let valid = verify_partial(
        agg_pubkey,
        borrower_pubkey,
        borrower_signature,
        repayment_adaptor_point,
        message,
);

let dca_partial_signature = sign_partial(
        agg_pubkey,
        repayment_adaptor_point,
        message
~~~~);

~~~~let adaptor_signature = aggregate_partial_signatures(
        agg_pubkey,
        repayment_adaptor_point,
        [borrower_signature, dca_partial_signature],
        message,
)
```

The borrower can adapt the withdrawal signature from the aggregated adaptor signature with their `repayment_secret` and broadcast the withdrawal signature on the Bitcoin chain to reclaim their BTC collateral.

```
Unset
verify_single(
    agg_pubkey,
    adaptor_signature,
    message,
    adaptor_point,
)
.expect("invalid aggregated adaptor signature");

let collateral_withdrawal_signature =
adaptor_signature.adapt(repayment_secret);
```

Once the signature is broadcast on Bitcoin, anyone can extract the `repayment_secret` from the Bitcoin collateral withdrawal to move funds from the Repayment Escrow to the liquidity pool. This is then verified by the contract:

```
Unset
let repayment_secret =
adaptor_signature.reveal_secret(collateral_withdrawal_signature);
if (sha256(usdc_repayment_secret) == loan.repayment_hash_lock) {
    // send funds to liquidity pool
}
```

3.5 Liquidation

3.5.1 Liquidation Scenarios

Collateral will be liquidated in two scenarios:

1. Collateral Value Depreciation: Liquidation is governed by a DLC, which is established during the Loan Assignment process. The Oracle provides signed attestations of BTC's price at predefined intervals. If the Oracle reports a drop in the BTC price that causes the loan's Health Factor to fall below 1, collateral must be liquidated. The DCA uses the Oracle's attestation signature to unlock

the pre-signed CET and its adaptor signature, allowing the DCA to spend the BTC collateral according to the terms of the CET. This action initiates the liquidation process, with the collateral sent to the DCA for liquidation. The DCA uses the oracle attestation's signature to unlock a pre-signed CET's adaptor signature and spend the BTC collateral according to the CET's terms. This action triggers the loan collateral's liquidation. Collateral is sent to the DCA for liquidation.

2. Default: If the loan is not repaid by the Maturity Time, the borrower is considered to be in default. There is a timelock in the Collateral Vault allowing the DCA to liquidate BTC once the Maturity Time is reached. In this case, all collateral is sent directly to the DCA for liquidation.

3.5.2 Liquidation Flow

The liquidation flow involves selling liquidated assets to recover as much of the loan value as possible. The DCA plays a critical role in mitigating risks for Liquidity Providers during liquidation.

The DCA performs the following tasks:

1. Receives Liquidated Assets: When a liquidation is triggered, the DCA takes custody of the liquidated assets.
2. Auctioning Collateral: The Lending Contract lists the collateral for auction at a discount relative to the current market value. For instance, if the collateral remains unsold, the discount increases by 1% every 10 minutes until a buyer is found. This approach ensures that the collateral is sold swiftly, minimizing the DCA’s custody risk.
3. Repaying the Pool: The auction proceeds, along with a liquidity penalty, are sent to the Lending Contract to cover the principal and interest.
4. Surplus or Deficit: If there is a surplus from the auction, it is returned to the borrower to minimize the impact of liquidation. However, if the auction proceeds are insufficient to cover the loan principal, the liquidity providers incur the loss.

3.6 Loss of Liveness

A `final_timeout` timelock in the Collateral Vault safeguards the Borrower in the event that the DCA or the Side Chain becomes unresponsive. In such cases, all locked BTC collateral is returned to the Borrower, mitigating liveness risks associated with non-Bitcoin components.

The `final_timeout` must occur after the loan’s Maturity Time.

4. Side Finance Security

Side Finance operates as a non-custodial solution, meaning no third party holds the native BTC on the Bitcoin blockchain throughout the loan's duration. The main trust assumption arises during two key processes: the liquidation of collateral through auctions and the repayment settlement.

4.1 BTC Collateral Security

The Collateral Vault is secured using a 2-of-2 multi-sig combined with a HTLC, ensuring that BTC collateral cannot be moved without the Borrower’s authorization. There are four methods to spend UTXOs associated with this address, all of which adhere to Bitcoin-native security principles:

1. using CETs of a DLC in the event of collateral value depreciation, relying on Schnorr adaptor signatures
2. through a Borrower-generated `repayment_secret` and a counter-signed adapted signature from the DCA that accepts loan repayment
3. a hash timelock that activates in the case of loan default (when repayment is not made before the Maturity Time), allowing the DCA to spend the collateral
4. a final hash timelock that allows the Borrower to reclaim all collateral in the event that Side Finance stops responding

In no case does any single entity have discretion over a Borrower’s BTC collateral outside the terms specified in the “contract” encoded within the DLC.

4.2 Loan Asset Security

The Lending Contract, a smart contract deployed on the Side Chain, governs the management of loan assets according to predefined rules encoded within its framework. Liquidity Providers have the flexibility to enter or exit the liquidity pool at their discretion.

4.3 Oracle Security

In a Bitcoin DLC-based DeFi system like Side Finance, oracles must cryptographically sign periodic price attestations, enabling the liquidation of BTC when necessary. Oracle operations need to be decentralized, run by operators who have an economic stake in the system that can be slashed if they act maliciously.

21 well-known Side Chain validators, chosen through on-chain governance by stakers, also serve as Oracle Operators. These validators are selected due to their vested interest in the system's success and their role in securing value across multiple chains. Should any validator provide incorrect price outputs, a cryptographically signed proof of misbehavior is generated. This proof automatically affects their stake holdings on Side and damages their reputation on other chains they secure.

It’s important to note that Oracle Operators cannot benefit directly from any price liquidation events they sign, that they cannot make up arbitrary prices, and that a threshold set of operators must agree in order for a price to be signed.

Each Oracle Operator independently monitors BTC prices from multiple cryptographically secure upstream sources. The public keys of these approved upstream price sources, selected by on-chain governance, are stored within the Oracle smart contract on the Side Chain.

The Oracle Operators collectively produce a stream of DLC signatures for potential future price events. Later, they sign an attestation of the real BTCUSD price for each time period as it occurs. These DLC signature streams and attestations are then used by the DCA to liquidate loans that fall below the Liquidation Price.

Price attestations proceed as follows:

1. An Oracle operator proposes a price to the Oracle smart contract for a recently passed time period and signs the proposal. The Oracle smart contract verifies the validity of the upstream source’s signature. If the signature is invalid, the proposing operator’s stake is automatically slashed, and the operator is flagged for investigation.
2. Other Oracle operators independently monitor BTC prices by retrieving data from approved sources. As prices may differ slightly at any given time, if the proposed price falls within a specified tolerance (e.g., 0.5%) of an operator’s verified price, the operator signs the proposed price. If the proposed price falls outside this tolerance, the operator does not sign and instead re-proposes a new price, which is again checked for validity.
3. This process repeats until all Oracle operators reach a consensus on the price. If consensus cannot be achieved, the system halts price attestations until the issue is resolved through human

intervention. A small minority of honest operators is enough to prevent the Oracle from signing an incorrect price.

4.3.1 Oracle Operator Slashing

Oracle operators should monitor the complete set of approved upstream oracles and base their price proposals on the median price. If a proposed price deviates significantly from the median price attested by upstream sources, anyone can initiate a slashing challenge.

Since all upstream prices are cryptographically signed and the public keys of approved upstream oracles are stored in the Oracle smart contract on Side Chain, a challenger can collect the signed price data from these sources and submit it to the contract.

The contract then checks whether the proposed price falls outside a pre-set deviation from the average of all upstream price sources.

If the proposed price exceeds the allowable range, the Oracle operator's validator stake is slashed.

4.3.2 Upstream Price Source Security

If, for any reason, multiple upstream price sources begin to deviate significantly from established BTC prices available from other reliable sources, Oracle operators should temporarily halt price signings. This pause will give the governance system time to remove faulty upstream sources and propose new, reliable ones for the Oracle system.

4.4 DCA Security

The DCA acts on behalf of the Lending Contract to sign necessary transactions; however, it's important to clarify that it doesn't have control over the Lending Contract or the Collateral Vault. The funds supplied by lenders remain non-custodial and securely held within the smart contract. The DCA's role is limited to temporarily holding liquidated assets (BTC collateral) for auction purposes. To reduce risk, these liquidated assets must be sold as quickly as possible.

The DCA essentially functions as an agent managing the bad debt of liquidity providers. Allowing liquidity providers to nominate the DCA aligns interests and fosters trust. This nomination and selection

process occurs periodically, with voting power weighted by the amount and type of LP tokens held. Crucially, the DCA and the Oracle (managed by 21 Side Chain validators) must remain separate entities.

Any misconduct by DCA operators will lead to penalties, including potential removal from the DCA role.

Additionally, all DCA operators must comply with Know Your Customer (KYC) requirements. This ensures all operators are properly vetted and held accountable for their actions within the network.

5. Side Chain

Side Finance is a non-custodial lending solution for Bitcoin, designed as a cross-chain lending system. In this framework, BTC collateral is securely locked on the Bitcoin network without the need for third-party custody, while loans are issued on Lending Contract which is deployed on a separate distributed ledger, Side Chain.

Side Chain is an independent distributed ledger operated and governed by validators. It hosts the lending contract but does not hold or control the BTC collateral involved in the Side Finance protocol. Consequently, the security of Side Finance relies primarily on the trust assumptions inherent to the Bitcoin network, rather than those of Side Chain. In case of liveness failure of Side Chain, Borrowers may reclaim Collateral after `final_timeout`.

Side Chain leverages CometBFT, a high-performance consensus engine that serves as its backbone. This architecture facilitates fast transaction finality and high throughput, making it ideal for applications that require quick confirmation times. Smart contracts on Side Chain are executed in a Wasm VM, written in Rust for performance and security advantages. Rust's memory safety and Wasm's compatibility reduce vulnerabilities like re-entrancy attacks, common in Ethereum-based smart contracts.

This alt-chain system is specifically crafted with Bitcoin-centric features to provide a seamless and frictionless user experience, and it paves the way for further innovations in decentralized finance for Bitcoin:

- Bitcoin Address Compatibility: Side Chain is fully compatible with Bitcoin addresses, allowing users to interact with it without the need to create new wallets or manage multiple addresses.

- Bitcoin Wallet Compatibility: Users can access Side Chain using their existing Bitcoin wallets, simplifying the user experience and expanding access to the appchain by leveraging the Bitcoin wallet ecosystem.
- BTC as Native Gas Token: Unlike other blockchains that use their own native tokens for gas fees, Side Chain utilizes BTC as its native gas token.
- BTC Bridging: Side Bridge functions as the primary bridge for transferring native BTC and Bitcoin-based assets (like Runes) between the Side Chain and the Bitcoin network. The initial version of the bridge uses threshold signatures, making it custodial but suitable for users with a higher tolerance for trust. This bridging feature operates independently and is not integrated into the Side Finance design.
- DeFi Services: Side Chain, as a Bitcoin appchain, offers a range of decentralized financial services, including a decentralized exchange embedded within the system, allowing users to seamlessly trade BTC, Bitcoin assets, and other crypto assets.
- Interoperability: Side Chain integrates with several established cross-chain communication protocols, facilitating the bridging of assets from various blockchain systems. These assets can be injected as liquidity into Side Finance and utilized within other financial services on Side Chain.

Side Chain is a sub-protocol within the Side Protocol stack. The overview provided above offers a high-level summary. A more detailed technical review can be found in a separate document.

References

[1] Bitcoin Ops. Adaptor Signatures. https://bitcoinops.org/en/topics/adaptor-signatures/

[2] Thaddeus Dryja. Discreet Log Contracts. https://adiabat.github.io/dlc.pdf

[3] Bitcoin Wiki. Hash Time Locked Contracts. https://en.bitcoin.it/wiki/Hash_Time_Locked_Contracts

[4] Victor Shoup. The Many Faces of Schnorr. https://eprint.iacr.org/2023/1019.pdf

[5] DLC Specs. Introduction to DLCs.
https://github.com/discreetlogcontracts/dlcspecs/blob/master/Introduction.md