Chain-Key Tokens

One important way how different blockchains can cooperate is by bringing tokens from one blockchain to another one, for example, bringing Bitcoin to the Ethereum chain. In traditional blockchain architectures, this is mostly accomplished through wrapping. More decentralized (and secure) alternatives to wrapping exist and can replace it: Meet ***chain-key cryptography*** and ***chain-key tokens***!

A ***wrapped token*** represents an underlying asset, which is typically native on a different blockchain than the wrapped token. For example, a wrapped bitcoin token represents real bitcoin, but is a different token than bitcoin and available on a different chain, for example, on the Ethereum blockchain. Traditional wrapping always involves intermediaries that need to be trusted.

A more secure alternative to wrapping tokens is to use advanced threshold cryptography to obtain ***chain-key tokens***. Chain-Key Bitcoin, the first major chain-key token on the Internet Computer, is a token on ICP with a 1:1 backing with real bitcoin held 100% on chain on ICP by a smart contract. Thus, it is a Bitcoin 'twin' on the Internet Computer that features low transaction fees and latency and high throughput, similar in properties to a Bitcoin Layer 2. We use Chain-Key Bitcoin as an example for chain-key tokens throughout this page. Chain-key fungible tokens should implement the ICRC-1 token standard so that they can be easily integrated by services, for example, wallets and DEXs. This helps make chain-key tokens readily available for a wide range of services.

Traditional Wrapped Tokens

In traditional blockchain architectures, token wrapping involves an off-chain trusted intermediary and a token ledger smart contract. A user who wants to have tokens of a specific type, say a wrapped bitcoin token, sends tokens of the underlying asset, such as bitcoin, to the intermediary. The intermediary, once it has confirmed the transfer of the underlying token on the token's native blockchain, keeps the received tokens in custody and instructs the token ledger to create, or ***mint***, the same amount of wrapped tokens that it has received of the underlying token.

Minting increases the supply of the wrapped token. The newly minted wrapped tokens can then be used on the token's blockchain, and the token ledger keeps track of all accounts.

If a user wants to redeem wrapped tokens for the underlying asset, this again involves the intermediary: The user sends the wrapped token to an address controlled by the intermediary and makes an unwrap request. The intermediary removes the amount of received wrapped tokens from the wrapped token's supply and returns the corresponding amount of underlying tokens to the user on the blockchain that natively hosts the underlying asset.

Regular users can just use the wrapped tokens and normally need not bother with the wrapping and unwrapping themselves, unless they own the underlying token and want to bring it to another chain or obtain the underlying token via its wrapped version. Thus, wrapped tokens are convenient for most users and as easy to use as any native token on the same blockchain.

This traditional off-chain approach of wrapping works well from a functional perspective, but has the major drawback of involving an intermediary whose integrity is crucial for the security of the wrapping and unwrapping of the token. The main problem is that the intermediary can get compromised, for example, hacked, defrauded by an insider, or go out of business, which may result in a total loss of the underlying tokens in the worst case. Strategies such as multi-signature schemes with keys held by multiple parties to try to decentralize the intermediary can provide some mitigation, but do not change the fact that wrapping is done by an off-chain entity or group of entities. In short, this architecture is not fully decentralized. Overall, this traditional way of realizing wrapped tokens is not desirable for reasons of security, risk, and its inherent centralized nature.

Another potential risk comes into play with those wrapping architectures: Ideally, a wrapped token is always backed 1:1 by the underlying token. In practice, however, the intermediary can use the tokens held in custody to create profit, for example, by using them for risky investments. In the worst case, if things go wrong, this can lead to the loss of tokens and a depegging of the wrapped token.

Chain-Key Tokens

Chain-key tokens, such as Chain-Key Bitcoin, are an advanced cryptography-based replacement for wrapped tokens offering stronger security: With chain-key tokens, all operations are performed completely on chain by smart contracts, without involving any off-chain intermediaries. This eliminates the intermediary-related risks of traditional wrapping architectures. Replacing wrapping with on-chain operations only becomes possible through the native integrations between blockchain networks and particularly the use of advanced cryptography — ***chain-key signatures***. Chain-Key Bitcoin (ckBTC) is the first chain-key token available on ICP. For each ckBTC token, a real bitcoin is held on chain by a canister smart contract.

We next give an overview on how chain-key tokens function based on a native chain integration and advanced cryptography.

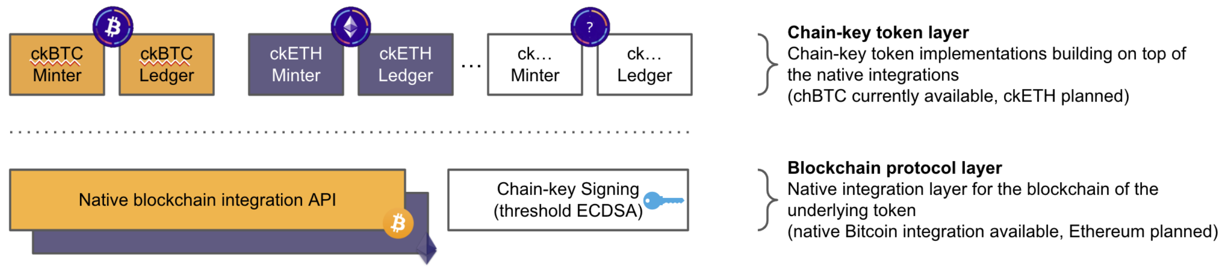
Architecture

The architecture of any chain-key token on ICP is using the following building blocks as its foundation:

1. Some form of native integration with another blockchain that hosts the underlying token, for example, the Bitcoin network and its only token, Bitcoin, must be available. This integration must allow canisters on ICP to query balances of addresses of the underlying token on its native chain as well as send transactions to the underlying chain. This integration with the native chain must be done such that no intermediaries are required, that is, is completely decentralized and realized on chain.
2. A chain-key implementation of the signature scheme used for signing transactions on the blockchain hosting the underlying asset must be available, for example, chain-key ECDSA for Bitcoin. This functionality makes it possible to sign transactions for the chain of the underlying token fully on chain using threshold cryptography without involving an intermediary.

Those two building blocks comprise the native integration with a particular blockchain. ICP has already been integrated with the Bitcoin network using this approach, and an integration with Ethereum is planned. Based on this foundation, chain-key token implementations can be built at the smart-contract layer. One possible architecture to implement a chain-key token for a specific underlying token, for example, Bitcoin, uses two canister smart contracts: A ***chain-key token ledger*** and a ***chain-key token*** smart contract. For the ***chain-key token ledger***, the ICRC-1 token standard should be used for wide compatibility in the ecosystem. The same open source ledger code base can be used and parameterized for any chain-key token to be deployed. In addition to the chain-key token ledger, a ***chain-key token smart contract*** is used in this architecture. This canister essentially replaces the off-chain intermediary of the traditional wrapping architecture with on-chain functionality for the chain-key token architecture. This canister is also called ***minter*** as it is responsible for creating (and removing) supply of the chain-key token based on in- and out-flowing underlying tokens. It keeps any underlying tokens it receives in on-chain custody as long as corresponding chain-key tokens are in circulation, thus ensuring a 1:1 backing of the chain-key token with the underlying asset. The 1:1 backing of the chain-key token by its underlying token can be verified by anyone by checking that the Bitcoin

UTXOs the minter claims to hold indeed are held by Bitcoin addresses controlled by the minter, thereby further enhancing trust.


ckBTC Minter
ckBTC Ledger
ckETH Minter
ckETH Ledger
…
ck… Minter
ck… Ledger
Chain-key token layer
Chain-key token implementations building on top of the native integrations
(chBTC currently available, ckETH planned)
Native blockchain integration API
Chain-key Signing (threshold ECDSA)
Blockchain protocol layer
Native integration layer for the blockchain of the underlying token
(native Bitcoin integration available, Ethereum planned)

High-level architecture for chain-key tokens on ICP

Note that the token ledger and minter functionalities can also be integrated into a single smart contract or more than two — this is a question of software architecture. However, splitting the functionality into two canisters as outlined is a sensible approach for the reason of modularity as the same ledger canister code can be reused for any chain-key token and only the minter needs to be adapted to the mechanics of the token of the underlying chain, particularly to account for differences between UTXO- and account-based underlying chains, like Bitcoin and Ethereum, respectively.

Creating Chain-key Tokens from Underlying Tokens

When a user wants to obtain chain-key tokens for some underlying tokens, for example, bitcoin, they send the underlying tokens to an address on the origin blockchain owned by the chain-key token smart contract (minter), instead of an intermediary as done for wrapping. The chain-key token smart contract has an address on the origin blockchain, which is made possible through chain-key cryptography, hence the name of chain-key tokens, or chain-key Bitcoin as a concrete example: In the Bitcoin example, using chain-key ECDSA signing technology — concretely, an advanced form of threshold ECDSA, the chain-key Bitcoin smart contract can have and use ECDSA key pairs, much like any user of the Bitcoin network, but fully on chain. That means the smart contract can obtain ECDSA public keys and from the public keys it derives addresses on the Bitcoin network, to which bitcoin can be sent on the Bitcoin network by anyone. Once this chain-key token smart contract has received Bitcoin from a user and the transaction has received a sufficient number of confirmations on the Bitcoin network, the canister instructs the token ledger for the chain-key token to create, or mint, an amount of chain-key Bitcoin tokens corresponding to the received amount of bitcoin. This approach leads to the chain-key token being 1:1 backed with the underlying token, which is held in on-chain custody by the minter canister smart contract.

Allowing the chain-key token smart contract to know about the balances of the addresses in the underlying chain it controls requires an integration between ICP and the Bitcoin network — the blockchain hosting the native asset to be issued a chain-key token for: ICP nodes connect to nodes of the Bitcoin network and pull in and validate Bitcoin blocks, extract the UTXOs and maintain the Bitcoin UTXO set on chain. Any smart contract can then query the balance and UTXOs of a Bitcoin address. This is what is described as (1) in the section on architecture above.

Redeeming Chain-key Tokens for Underlying Tokens

A chain-key token can circulate on the ICP as long as needed. There is no need to frequently bring in and transfer out underlying tokens, and normally there is no need for most users to do this themselves. However, a user may want to redeem chain-key tokens they hold at some point to receive the underlying asset. In this case, they send the amount of chain-key tokens to redeem, for example, ckBTC in our example, to the chain-key token canister (minter) for the token with a request to redeem them. The canister first removes the received chain-key tokens from the supply on the chain-key token's ledger. Next, it creates a transaction on the blockchain of the underlying token, the Bitcoin network in this example, to transfer the same amount of underlying tokens (modulo fees) that it has received of chain-key tokens to a user-provided address on the Bitcoin network. This step involves chain-key signatures — concretely an advanced form of threshold ECDSA signing — to authorize the transaction on the underlying blockchain (foundation (2) above). The chain-key-signed transaction is then sent out via inter-chain communication from ICP nodes to Bitcoin nodes, another crucial functionality of the native blockchain integration explained in (1) above.

Specifics of Chain-Key Bitcoin

Since Bitcoin uses the UTXO model for tracking account balances, the wrapping contract needs to implement proper handling of UTXOs, which is far from trivial. This involves, for example, a good heuristic selection of UTXOs to consume when transferring bitcoin back to users, as well as handling error cases, for example, when transactions do not get mined on the Bitcoin network due to too low fees. Implementing this properly and considering all edge cases is hard and requires well-thought-out algorithms in the implementation of the chain-key token canister.

The Future: Chain-Key ERC-20 Tokens

When the Internet Computer blockchain will integrate with additional blockchains in the future, more chain-key tokens will become available on ICP. The token ledger of a new chain-key token can use the same ICRC-1 ledger code base, parameterized for the respective new chain-key token. The chain-key token canister smart contracts, or minter, needs to be re-written for different blockchains. The next major blockchain integration being planned is ICP <> ETH, bringing Ethereum's ERC-20 tokens to the IC as chain-key tokens. This requires a new variant of the minter canister for ERC-20 tokens that is then replicated for each token.