Cannon

Decentralised Network for Asset Interoperability

White Paper

November 15, 2020

cannon.finance

Abstract: Cannon Finance is a decentralised network that enables digital asset interoperability between various blockchains. The decentralised finance[1] (DeFi) landscape has rapidly evolved[2] over the course of the year 2020. It has shown that various financial applications can be created without centralized control or censorship. However, these applications currently have a limitation in terms of interoperability and decentralization. Most of these applications are powered on Ethereum and lack a true interoperable bridge to bring other blockchain assets onto Ethereum to enable cross-chain decentralised finance. Cannon is an interoperable decentralised bridge to wrap any blockchain based asset onto Ethereum. Cannon will operate by a basis of incentives and penalisation powered by its native Cannon token (CAN) where the protocol pays honest nodes and slashes dishonest ones while maintaining a private layer.

Introduction

Cannon is decentralised asset bridge that enables interoperability between various blockchain assets. Anyone, anywhere, can utilise Cannon to send any quantity of an asset to and from Ethereum to its native chain. Cannon is secured by Cannon tokens (CAN) that are required to be bonded with large shards that operate by shuffling Cannon to make it extremely difficult to attack. Cannon enables these bridges to become scalable as more digital assets are locked into the protocol which has an algorithmic fee adjuster that creates auto scaling to fit any capacity layer.

The importance of cross chain interoperability falls in the ways that decentralisation is achieved, how applications are operated, combined with the value of various digital assets. Interoperability is a major challenge to the belief of decentralisation and the continued scalability of blockchains as a whole. Currently Bitcoin is 10 times larger[3] than the value of Ether and Ether has a 2–4x larger value of the other top major digital assets. However, there lacks a true bridge to bring those values onto Ethereum to utilise the worlds most used decentralised smart contract platform powering an immense number of DeFi applications.

For example, for Decentralised Exchanges to grow into more viable products outside of their current asset class or synthetics, we need to create a bridge where we bring an actual bonded asset with a digital representation token (DRT) onto the network. To connect all users from various blockchains together, it is important to bring the scalability of blockchains to reality. There should be no reason, other than technical limitations, why the top digital assets can't be ported and used onto one single decentralised layer, which in this case would be Ethereum. Cannon solves this technical issue by enabling trust–less DRTs to be created and the native digital asset custodied in a method where it is secure and distributed.

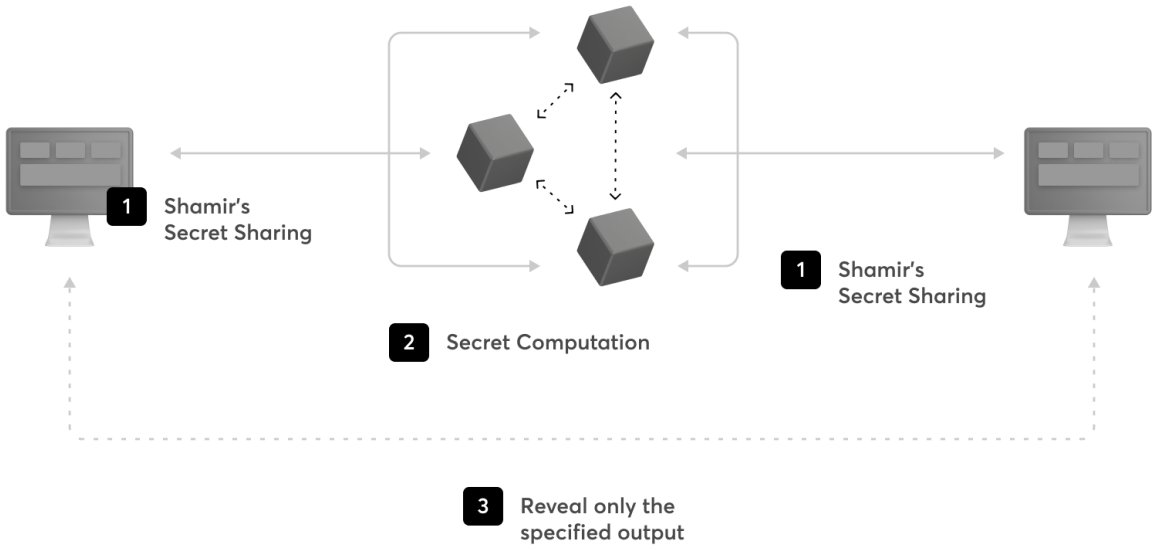
Cannon is built on a decentralised network, forked from the Republic protocol[4], that leverages zkSNARK[5] and their iteration of secure multiparty computations[6] (sMPC) to run programs in zero–knowledge[7] and validate proofs to bring cross–chain interoperability. Any application or program can be executed on Cannon while maintaining privacy by keeping its inputs, outputs, and state, completely private. Decentralised applications on Cannon can also synthesise completely new data sets, that are not known to anyone, and can still be governed by the protocol's consensus.

Consensus

Cannon operates a Byzantine Fault Tolerant[8] (BFT) network that enables decentralised interoperability between any digital asset. By utilizing this consensus mechanism with secure multi–party computation (MPC) algorithms, Cannon offers a permission–less bridge bringing other blockchain assets onto Ethereum via a locking mechanism that is trust less. This is achieved by maintaining a one–to–one of a digital representation of other chains via a decentralised custodian. Cannon will initially support a decentralised bridge between Bitcoin followed by BNB, XRP, LTC and more digital assets wrapped into a digital representation token (DRT) on the Ethereum blockchain. Additional assets can be introduced through the Cannon governance process which is powered by CAN tokens.

Secure Multiparty Computation & zkCompute

The protocol has a customized, scalable version of secure multiparty computation scripts that allow untrusted nodes to collaboratively execute an application or program without revealing the inputs or outputs of the application, not even to the other nodes in the protocol themselves. This is highly useful for executing applications or programs that require inputs from various users that want to keep this data private and share it with others without relying on a on a centralised third–party.



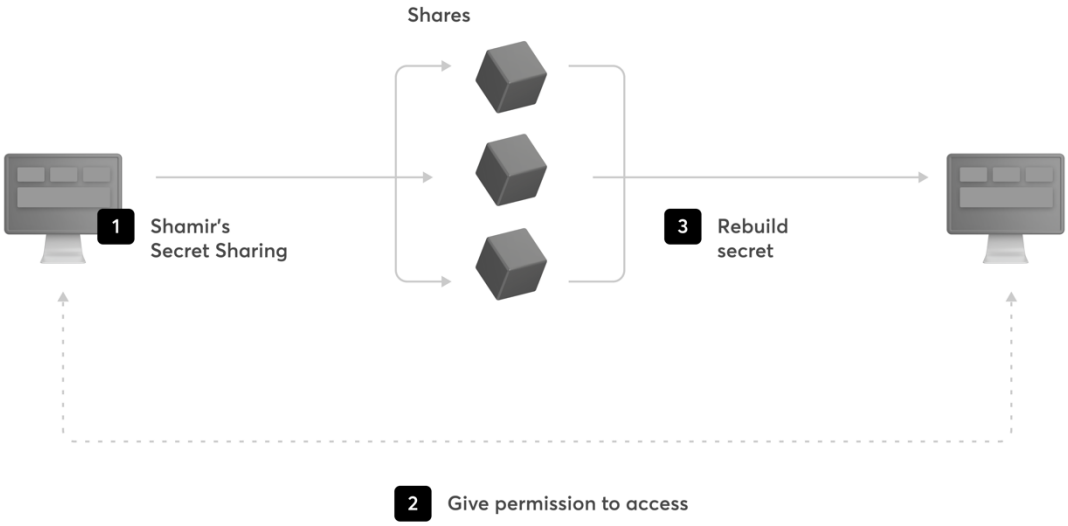
Although secure multiparty computation is general purpose, and can be used to execute any program, modern versions are inefficient and not fault tolerant. They require computationally intensive pre-processing phases that generate large amounts of data, and the entire execution fails whenever one node goes offline. This makes them impractical in decentralised networks where nodes are geographically distributed around the world and cannot be trusted to remain online even when they are operated honestly.

Cannon solves this using this new secure multiparty computation protocol which has a simpler and more efficient pre-processing phase, that does not require complex cryptographic primitives, and generates less than 1/3rd of the data compared to modern protocols. Security and liveliness are guaranteed as long as less than 1/3rd of the network is malicious. This new secure multiparty computation protocol is at the heart of Cannon, allowing a decentralised network of untrusted nodes to jointly execute general purpose programs in complete secrecy. No data is known to anyone, unless explicitly defined by the computation.

zkStorage and Permission-based Access

Secure multiparty computation is useful for more than just executing programs, it is also effective for storing sensitive information. In Cannon, data is encrypted and distributed using Shamir's secret sharing[9], a form of informational theoretically secure encryption that splits data into shares. Unless a specific threshold of shares is recovered, the original data cannot be recovered. This prevents anyone, including the nodes running Cannon, from being able to access that data.

Cannon also allow permissions to be defined that can be used to govern access to stored data. This can be done by defining a set of signatures that have read/write access, but it can also be done by defining complex zkCompute[10] programs that evaluate governance rules in secret before granting read/write access. This allows for flexible and powerful permission-based access to secret data without necessarily revealing the identity of accessor.



Using Cannon, sensitive information can be stored securely, that is trust–less and fault tolerant. Simple, or complex, rules can be defined making it flexible for all kinds of applications, but particularly well suited for financial tools and enterprises.

The combination of zkStorage[11], with permission–based access, and zkCompute can be used to build stateful applications that are persistent in Cannon over time. Such stateful applications can keep their state secret, prove properties about their state, and transition their state.

Problems

The core functionality of Cannon is to enable trust–less interoperability of digital assets to bring any asset onto Ethereum. For this to be efficient, it must happen within one transaction to eliminate multiple steps of friction. Interoperability issues have been attempted to be solved through various iterations of networks. While these existing solutions have achieved some level of interoperability, there are many disadvantages present. A high–level overview of these interoperability's is present with the last being the most effective.

A very common method of interoperability discussed are Atomic Swaps[12] which use Bitcoin scripts and Ethereum contracts to guarantee that the digital assets are swapped in full or not at all. For example, if User–1 is performing a swap of BTC for ETH with User–2, User–1 will not get their swapped ETH until User–2 is able to

completely get their BTC and vice versa. There are two issues with Atomic Swaps as described below:

1) They are only applicable for swapping and the two counterparties must agree on a price and the assets. This creates a lack of functionality where there is a clear limitation of where they can be utilised. You are unable to use Atomic Swaps for AMMs or DeFi Apps.

2) By design, Atomic Swaps require time gaps to function correctly. This could lead to the counterparties intentionally moving slow to evaluate market conditions to arbitrage their bets. One party can always cancel the swap, and this gives them the option to back out of the deal that becomes unfavourable.

Recently, projects have been using a form of synthetics to achieve interoperability to give users exposure to an underlying asset. For example, sUSD[13] is a synthetic stable coin that gives users exposure to the US Dollar. Synthetics typically operate by depositing collateral in excess or equal to the synthetic representation of the unit. This pose risks as if the collateral loses value to the point of a threshold ratio being met, you expose yourself to losing your collateral. While synthetics have their advantages, we outline below some key disadvantages:

1) Synthetics are not cross-chain compatible today. For example, sUSD cannot be transferred onto another chain with the current tools available. Therefore, a synthetic is typically only able to interact with the smart contracts of one chain.

2) Synthetic assets typically cannot withstand high market volatility as if the underlying asset has a volatile price movement, this could leave a synthetic asset with under collateralised positions which breaks price stability exposure.

3) These synthetic assets are not redeemable for the underlying asset it has exposure to. The exposure does not mean it pegged to the unit, but rather just the value. For example, if you have synthetic XRP and wanted to get a unit of XRP in exchange, you would need to either unbond that collateral and buy XRP or find a counter-party willing to do the exchange.

The most effective method for cross-chain interoperability is the use of digital representation tokens (DRT's). These are typically the most used cross-chain interoperable asset classes. For example, there are many iterations of using DRTs to

achieve cross chain compatibility such as TBTC[14] and Wrapped Bitcoin[15] (WBTC) and. These DRTs are based on when a custodian locks up the underlying asset to mint a pegged asset on another chain. This DRT can then be minted and burned behind a mechanism that relates to the units of the underlying asset. However, these mechanisms have flaws as outlined below:

1) TBTC requires a synthetic–based over–collateralization and liquidation mechanism which means the one–to–one peg can be broken with market volatility as expressed above. There are also limitations in the number of units of BTC it can handle and requires numerous transactions on both chains. Their also lacks true decentralisation as the operators and signers of the network must accept big risk for little reward.

2) WBTC requires a centralized custodian to handle the security and redemption process of BTC. It means only these custodians can mint these DRT's and also removes any decentralisation involved with this process. There are regulated risks present which undermines a permission–less system of a blockchain and requires manual work to achieve from the counter parties that you must trust.

Solutions

Cannon enables a decentralised interoperable bridge for cross–chain digital assets utilising DRTs that solves the issues at hand today. Cannon utilises the staking and bonding of CAN with automatically adjusts fees based on algorithmics to operate. This is important as it relates to protecting underlying assets from high market volatility occurrences. Cannon does not utilise a trusted execution environment but rather utilises its native core to scale its capacity and adjust its fees to enable Cannon to automatically increase its capacity for more locked underlying assets. This is a step higher than TBTC as it removes the risk and need of liquidation as well as peg losses.

The protocol replaces the need of a trusted centralised custodian and enables a permission–less decentralised custodian through Cannon Nodes. These Cannon Nodes are designed using an RZL MPC[16] algorithm, which can create and manage ECDSA[17] private keys without the risk of exposing these keys publicly or even through the Cannon protocol itself.

Cannon creates a swift and decentralised user experience by enabling the minting and

claiming of DRTs on–demand. There are no constraints to quantity, time or location. Cannon transactions occur within one transaction without the need of complex solutions. To achieve the interoperability functionality of Cannon, the core is broken into three parts: **Nodes**, **Fees**, and **Shards**.

Cannon Nodes are distributed virtual machines that together power Cannon. Each of these Cannon Nodes are required to stake and bond 10,000 CAN tokens to participate in the network. These Cannon Nodes are incentivised through fees for being an honest node with a guarantee of their stake. If a Cannon Node behaves maliciously it can have its staked CAN slashed[23].

Incentives

Fees within the protocol is the prime incentive to operate Cannon Nodes and the protocol. These node operators are rewarded with fees paid by the users. For example, if a user wants to utilise Cannon to wrap BTC to Ethereum, they are paying a small dynamic fee that is earned by the Cannon Nodes. The fees are paid natively so there is no need to maintain other assets when performing an interoperable exchange. The protocol adjusts fees algorithmically through a series of supply and demand foundations. Cannon charges fees for all interoperable cross–chain transactions which are defined below:

- **Cannon Mint**: Minting fees are collected when a user mints a DRT on Ethereum *(Sending BTC from the Bitcoin blockchain to the Ethereum blockchain)*
- **Cannon Claim**: Redemption fees collected when a user claims the underlying asset to a DRT *(Sending BTC from Ethereum back to the Bitcoin blockchain)*
- **Maintenance Fees**: These are charged per block for holding a DRT. This fee should remain at zero unless there's a protocol security risk or incident that requires Cannon to collect these fees for Cannon Nodes. This is fee is compounded to the exchange rate of the underlying asset which creates a disparity in units received when redeemed.

On default Fees for the protocol are set at 0.1% of the units transacted with. Maintenance Fees initially set to 0% and can be changed by Cannon Governance[24].

Team

Cannon's codebase is all open-source[25] and forked[26] where it's transparent for all yet also preserves the protocols privacy based on its architecture. With having an entire open-code base, we can maintain the privacy of the founding team and enable an open developer community to contribute wherever or whoever they are. There is value to the anonymity of Satoshi Nakamato[27] and we created a project where the founding team is not as important as the project itself and not rewarded as such. Given the token economics and source code, users are able to verify, instead of just trust.

Token

Cannon is secured and operated by the Cannon Token (CAN). CAN is an Ethereum based ERC-20 cryptocurrency which operates Nodes that are required to bond their CAN on the Cannon network. CAN is also used as the protocol's governance token to make changes to Cannon with sufficient consensus (>50%). The CAN a Cannon Node operator stakes/bonds to the protocol is for security of their good behaviour and can be slashed if the Cannon Nodes acts maliciously or if the Cannon Node is part of a shard that behaves maliciously. This provides an economic incentive for the Cannon Node operator not to do anything maliciously or be a part of it.

Distribution Sale

Cannon's token distribution is set to be non-inflationary[28] as all tokens are distributed unlocked on inception and there are no further token release schedules. This prevents potential dilution from inflation as time progresses. Private sale tokens will be restricted from sale for one year but may be used on decentralized finance protocols. There are also no founder, team, or advisor allocations since all members are private. This trade-off guarantees that there's no network abuse from the initial distribution. Funds raised will be used for community and development incentives. There will be a total of 1,000,000,000 CAN tokens minted and distributed in the following manner:

Amount of CAN	Category
100,000,000	Public Sale Distribution at $0.02 per CAN
900,000,000	Private Sale Distribution at $0.02 per CAN

Token-based Safety

Using Cannon's native cryptocurrency means that Cannon does not need to introduce a mechanism to actually liquidate Cannon Nodes, which would expose them and the users of Cannon to market volatility risks. It is important to note that in any trust-less protocol that has interoperability, there must be some safety mechanism of value, in which case there will be CAN bonded. If there wasn't a safety mechanism in place that had bonded value, there would be huge financial incentives for nodes to compromise the protocol when there is decentralised custody of assets. As of today, there are no known methods to completely prevent theft of assets in decentralised custody, so the only method is to make it make very difficult and unprofitable. Different trust-less cross-chain protocols often need the total value bonded to exceed the total value that is locked by various multiples.

As an example, Protocol π wants to send $X of Bitcoin to Ethereum and will nodes or validators to bond at least $Y worth of digital asset value. Suppose that π uses ETH for its staking and bonding. When someone plans to send $X of BTC to Ethereum, nodes operating π must stake/bond at least $Y of ETH. The first advantage of this method is that using ETH, π can be sure that there is $65.4B worth of ETH (market capitalization value) that may be utilised for staking/bonding and there are plenty of people able to participate as nodes. In theory, π could be holding in custody up to $65.4B in BTC at any given point.

Although this method seams ideal, it does not feasibly work as the value of ETH is not correlated with the use of π. There are market volatility conditions that would make this problematic. If demand increases, but the price of ETH remains the same of drops, it could cause security risks. Even if demand remains the same, but the price of ETH drops or vice versa if demand drops and ETH price drops, it becomes problematic. As demand for π changes, or as the price of ETH fluctuates, the π nodes/validators must constantly adjust or size of their ETH bonded/staked. In these scenarios listed above, the π validators/node would have to increase their stakes/bonds or else π would become vulnerable and pose a security risk. Furthermore, π would also need a way to make nodes to do this. The only method known for this to be executed would be to initiate a liquidation and forfeiting the bond of π nodes if they fail to maintain adequate bonds. This poses a massive risk for π nodes, but liquidation functions have been also known to fail during times of market volatility and require the use of pricing oracles which are not fully decentralised and rely on a maintainer.

Architecture

Cannon Nodes are utilising shards[18] which shuffle into random non–overlapping groups for computations. Each Cannon Node shard contains at least 100 nodes which perform the random shuffles once per day. By doing this, it makes Sybil attacks very difficult as this would require the attacker to obtain and control a large portion of the entire network to have an opportunity to do something malicious through even one shard. This makes any social attack very futile as well as it would require an attacker to locate and collude with a large number of these anonymous Cannon Nodes in a short period of time which is practically impossible. The core functionality and parameters of Cannon enable it to be attack resistant. Each Cannon Node utilises RZL MPC algorithm to create a secret ECDS private key which is unknown to the entire network including all the Cannon Nodes in that shard. This key cannot be utilised to sign any transactions or disclosed without the consensus of over one third of the Cannon Nodes. The Cannon Nodes are partitioned in format where they are randomly sampled and are continuously shuffling these shards. These shards are combined in a way where it is smaller than the entire network so that they have the capability to operate more efficiently and securely. These overall functionalities secure the protocol and its assets that are in the decentralised custody.

Validator Shards

The purpose of Validator Shards is to generate, utilise and rotate secure ECDSA keys. Validator Shards are independent from each other, so any successful attack, even though its improbable, will only remove funds from that shard. This prevents failures in individual shards from propagating throughout the Cannon network and causing massive custody losses.

The ECDSA keys that are generated utilizing z0–sMPC engines and are unable to be exposed or used maliciously. The only way an attacker would be able to is to infiltrate over one–third of the shards. These keys are utilised to interoperate assets from cross–chain blockchains.

Validator Shards must rotate these ECDSA private keys at the beginning of every epoch[19]. This occurs so that if/when Cannon Nodes leave the protocol, they are unable to know any details of the actively used ECDSA private keys. When these Cannon Nodes are no longer part of the network, they do not have a bond that disincentives

the Cannon Nodes to reveal their shares.

When these rotations occur, Validator Shards in epoch A generate new ECDSA private keys for the Validator Shards in epoch $A+1$. This enables the generation of shares that are encrypted, specifically for Cannon Nodes in $A+1$, therefore, no Cannon Node in A can know the shares that have been generated. The Validator Shards in A will then sign transactions that forwards all assets to the newly generated ECDSA keys.

Transactions and digital assets are distributed through between Validator Shards to maintain a load-balance between the shards, ensuring that all Validator Shards have an equivalent amount of assets under custody. By distributing these assets it increases the capacity of Cannon to ensure isolated failures will only impact $1/N$ of digital assets, where N is the number of Validator Shards. The current number of Cannon Nodes assigned to a Validator Shard is 100.

Traffic Shards

The Traffic Shard is responsible for the coordination of Validator Shards. It is responsible for selecting which Cannon Nodes belong to which shards (including the coordination shard itself), and it is responsible or reaching consensus on which transactions will be processed by which shards. These Traffic Shards do not validate or execute these transactions, and they do not have ECDSA private keys. Blocks in the traffic order transactions, assign those transactions to specific Validator Shards, and generate secure random numbers. These secure random numbers are generated using the z0-sMPC engine and are guaranteed to be uniformly random unless an adversary can corrupt 2/3rd+ of the Traffic Shard. Currently, the number of Cannon Nodes assigned to the Traffic Shard is 100.

Core Shard

The Core Shard is one shard comprised of Cannon Nodes that are elected through the Cannon Governance process. These Cannon Nodes that are part of the Core Shard are reputable members of the Cannon community and have a pure stake and overall protection of the Cannon ecosystem. The Core Shard is a defence line for all Validator Shards. Validator Shards themselves cannot mint or release digital assets in custody without receiving as second signature. The number of Cannon Nodes in the Core Shard depends on the Cannon Governance proposals.

Epochs

In the Cannon protocol, an epoch is the private time interval that determines when the Cannon shards will rebase. When a rebasing occurs any new Cannon Nodes that have been pending entrance into the protocol for activation, will be activated and can begin validating the network. There will be a shuffle of existing Cannon Nodes to create new shards and all network private keys are created. This will also enable the time period where any Cannon Node that wants to exit, to safely exit. Also, by going through these rebases, it enables security on the protocol to help avoid any attacks that can be infiltrated by dishonest Cannon Nodes.

When the current epoch concludes, the network private keys have to be rotated. Cannon Nodes will eventually become inactive and this will prompt them to deregister which withdraws their collateral bond. This process occurs so that no deregistered Cannon Node has a Shamir's secret shares for a participating and an active network private key. These Cannon Nodes responsible for the next shard at epoch M will create a new ECDSA utilizing the RZL sMPC algorithm. When this process occurs, it is done where public keys are known, but the Shamir secret shares are actually encrypted for the next Cannon Node epoch $M+1$. These Cannon Nodes now at epoch M will combine all UTXOs into one transaction that will forward all the BTC to the newly created ECDSA private key.

The Cannon Nodes at epoch $M+1$ will receive the public key and their part of the private key from the base block. The ECDSA public key and encrypted Shamir's secret shares are stored as a state in these base blocks. For the next duration of epoch $M+1$, the Cannon Nodes will continue to support these network scripts built at epoch M and will process all the transactions related to epoch $M+1$.

Selection

Cannon Nodes go through *selection* which is a process in which they are placed into Validator Shards and Traffic Shards. This *selection* process must be random to protect the protocols security. When there is a rebase epoch, the Traffic Shard will mine a rebase block. A rebase block has no transactions and its only purpose is to ensure that the shards rebase. For the rebase block to execute correctly, the Traffic Shard and Core Share use their secure random number set and use it as a seed to randomize the Cannon Nodes into the next Validator Shard and the Traffic Shard. This selection algorithm will ensure that there will only be one Traffic Shard and ensure as

many Validator Shards as available. It will also assign the maximum possible Cannon Nodes per shard and it will never include a Cannon Node in more than one shard.

DKG

As Cannon goes through its normal flow of operation, the deregistration of Cannon Nodes will often result in various number of shards between epochs. When the protocol proposes a rebase block, the Traffic Shard must propose the transactions from all Validator Shards in both epochs. This way it will keep the assets under custody equally distributed between all Validator Shards. The distributed key generation algorithm used in Cannon, will enable one share to generate a secure ECDSA private key from another shard, while maintaining the privacy of the resulting shares.

Security

Cannon operates under a Byzantine Fault Tolerant system which needs to assess the conditions on which it can be attacked as it will hold assets belonging to users. To start, Cannon utilises the addition of a second signature requirement on all minting and claiming transactions. The protocol adheres to strict standards to maintain its security such as **never** producing a minting signature unless it has witnessed a custodian lock on the base chain (such as on Bitcoin's blockchain), produce a claim signature unless it has witnessed a burn of the Ethereum based collateral, execute a transaction from the next block height without executing all transactions at its current block, and commit to multiple blocks at the same height.

The consensus engine that is responsible for producing blocks of transactions is built on a modified version of Tendermint[20] created by the Republic team called: Hyperdrive. This modified version of Tendermint still adheres to the security of under the one third of adversarial condition but has a process for fast forwarding over missing blocks and changing the responsible signatories for block production. This consensus algorithm

requires the existence of isValid which validates the block. This guarantees that under the Tendermint consensus, no block that is invalid is produced. For a block height to be valid, Cannon will require the state of the block is embedded for all the transactions at this block. The isValid function is locally computable, which means that adversaries and attackers cannot just imply influence the decision of honest Cannon Nodes. It is important to note that within Cannon, the output state resulting from a transaction being executed is unique with respect to the input state and the transaction. This means that no two blocks within the protocol will have the same state after an execution.

In Cannon, a cross–chain transaction involves producing a minting or claiming signature. To produce this signature, a share in Cannon will use the RZL MPC algorithm to sign a digest. This share then uses an ECDSA private key that was created using the RZL MPC algorithm at the start of the epoch and is unknown to everyone. The Shamir's secret sharing threshold used by the RZL MPC algorithm is one–third. This implies that a signature cannot be produced and the underlying ECDSA private key cannot be exposed.

Assumptions

Typically, an adversary or attacker is any participant, Cannon Node, or otherwise relevant to the consensus of Cannon, that does not conform with the rules of network. These group of malicious actors can behave arbitrarily and attempt to find a point of failure in the system in order to capitalize their attack. To ensure the adequacy of the security in the Cannon ecosystem, we assume facts to help better analyse situations such as:

1. Attackers typically are capital constraint
2. The network isn't fully asynchronous
3. Ethereum is a computation engine that is secure
4. Participants have economic benefit from behaving honestly
5. Attackers cannot move faster than the duration of an epoch

Security Incentives

To prevent malicious nodes within the Cannon protocol, there are a series of security incentives, or economic incentives, to participate in the honest operation of the network. Cannon Nodes are required to stake and bond their CAN tokens to the protocol

at 10,000 CAN per Cannon Node. This bond is then held with an Ethereum smart contract. Once a Cannon Node successfully bonds their CAN tokens to the smart contract they are now registered. There is a wait period before they can join any shards in the next epoch. To unbond their CAN stake to the network, they must wait until the beginning of the next epoch, and then a subsequent epoch before actually being able to withdraw their bond. Cannon Nodes which have not withdrawn their CAN can be slashed. Cannon Nodes that pre–commit, pre–vote, or propose two different blocks that are within the same height of a round, can have their CAN bonds slashed by submitting the details to an Etheruem based smart contract called: "Cannon Slasher"

In the Cannon protocol, shards that produce minting signatures without witnessing an actual lock transaction on the origin chain can have their bonds slashes as well. At any time, any participant can submit a challenge, which requires a bond, to the Cannon Slasher contract. The Cannon Node performing the slash (slasher) must produce an SPV[21] proof for the respective lock before the end of the subsequent epoch, or every Cannon Node, in the challenged share, will have their bond slashed. This creates economic disincentives for Cannon Nodes to act maliciously in any capacity. If this SPV proof is proven true, then the challenger will lose their bond.

In Cannon, shards that produce releasing signatures that have not witnessed a respective burn on the native chain (such as Bitcoin blockchain) can have their bonds slashed as well. As outlined in the process above, a challenger can submit a challenge that contains a SPV proof of existence for this release transaction in question. The Cannon Node must produce a SPV proof of existence of the respective burn prior to the end of the next epoch, or every Cannon Node in this share will have their bond slashed. If the Cannon Node can successfully prove via a SPV proof, the challenger will lose their bond.

Node Uptime

Cannon Nodes are required to remain online to coordinate within the protocol. There are properties in place that guarantee that no share is ever offline. With this, that means that less than one third of Cannon Nodes are unable to communication with the remaining Cannon Nodes in the share. This is calculated cumulatively across Cannon Nodes that are inadvertently unable to interact or communicate with Cannon Nodes that maliciously refusing to communicate. This is known as a Liveliness Property which is part of the Tendermint consensus algorithm. Within the Liveliness Property is a

second parameter which involves Shamir's secret sharing threshold in RZL MPC which represents 1/3rd on the Cannon Nodes which means that the underlying ECDSA private key can always be recovered as long as 1/3rd+ of Cannon Nodes are not offline. This means that only 2/3rd+ of the Cannon Nodes will be required to complete a signation and key generation transaction.

Fee Incentives

The Liveliness properties of the Cannon protocol require Cannon Nodes to contribute resources to power Cannon which does have a monetary cost such as network bandwidth, CPU/memory power, and storage space. Therefore, there must a economic reward that incentives these Cannon Nodes to operate. The protocol charges users a fee whenever they create a cross-chain transfer. Cannon Claim is a function that enables users to redeem the underlying asset in which they hold a DRT for such as cBTC for BTC which accumulates fees for Cannon Nodes. Cannon Claim Minting also accumulates when cBTC is minted on Ethereum and when cBTC is claimed for BTC. These fee structures were previously explained.

Network

Cannon has a series of Networks built into the protocol that power the interoperability between blockchains and their assets. Cannon users are able to transfer these assets through the Cannon Network from one blockchain and then mint the respective asset on another. The Network has application-specific data fields, enabling smart-contract interactions with other decentralised applications.

Cross-chain interoperability between Bitcoin and Ethereum is achieved through a tokenized Ethereum representation of Bitcoin, called Cannon BTC (cBTC). When a user sends BTC to Cannon, it is considered locked, and now Cannon will mint the exact units of cBTC, minus the protocol fees, to the requested Ethereum address. This process also occurs when cBTC wants to be redeemed and is claimed. The user then burns cBTC, and Cannon will utilise its Networks to release the respective units of BTC by transferring it to the specified Bitcoin address.

This entire process is a permission-less, decentralised, and trust-less process which sets Cannon apart from its competitors by utilizing RZL sMPC algorithms to generate, rotate, and operate ECSDA private keys in secret. The ECDSA private keys that are generated are used to transfer, receive, and authorize the minting of cBTC by using BTC. The algorithm also guarantees that ECDSA private keys are not exposed by anyone and cannot be utilised without consensus from the Cannon Network. This is a key reason why Network contracts are considered private and secret contracts.

Cannon Network enables users to perform digital asset swaps between multiple blockchains. For example, those who want to bring Bitcoin to Ethereum would follow these two functions:

- **Cannon Mint**: Send BTC from Bitcoin to Ethereum (Mint cBTC/Lock BTC)
- **Cannon Claim**: Send BTC from Ethereum back to Bitcoins blockchain (Redeem BTC/Burn cBTC)

Cannon Mint

Cannon Mint is a Network function that enables users to initiation a cross-chain transactions to bring Bitcoin, or any other supported digital asset, onto Ethereum. This is initiated in one transaction where the user will send BTC to a specified address and in return receive a DRT BTC, this this case ChainBTC (cBTC), on the Ethereum blockchain through a unique minting signature for that transaction. The units minted will be equivalent to the units locked minus fees. This asset remains locked with a decentralised custodianship where it can only be redeemed for the DRT pegged unit at any time.

To initiate this process, the user must first generate a valid Network script. The user will then transfer BTC to the network script, wait for 6 block confirmations, and then call the chain_submitTx to the Cannon Nodes. This will notify the Cannon Nodes about the transaction of the Network script and the UTXO that transferred the BTC.

A Network script must contain the following data to be considered valid:

ghash

OP_DROP
OP_DUP
OP_HASH160
pub_key_hash160
OP_EQUALVERIFY
OP_CHECKSIG

- ghash is the keccak256(..) hash of encode(phash, token, to, n), where encode(..) is the Ethereum ABI encoding function (as implemented by the Solidity abi.encode(..) returns (bytes) function).
- pub_key_hash is the hash160(..) of the Network public key. This changes every epoch.
- phash is the keccak256(encode(..)) hash of arbitrary application-specific data, encoded using the Ethereum ABI encoding function. By including phash in the Network script, the application-specific data is bound to the script. All BTC transferred to the script will be minted on Ethereum in association with this phash, allowing smart contracts to validate the application-specific data.
- token is the Ethereum address of the token being minted. In the case of BTC, this is the Ethereum address of cBTC. This is the same for all *BTCEthereum* network scripts.
- to is the Ethereum address that will receive the cBTC. This must be the msg.sender that calls the mint function on the *BTCNetwork* contract.
- n is nonce made up of 32 random bytes. By including n in the Network script, the script can have a unique address even when all other details are identical.

For a chain_submitTx transaction to be submitted, the user must broadcast a Cannon transaction. The Cannon transaction will contain the following to be considered valid:

1. the phash,
2. the application-specific data that is used to compute phash,
3. the token address,
4. the to address,
5. the n nonce, and
6. the UTXO that has transferred BTC to the Network script.

This will now initiate the Cannon Nodes to process the Cannon transactions, verify its details, independently build the Network script, ensure validity of the UTXO, check the uniqueness of the UTXO, and finally check that the UTXO has the required 6 block confirmations.

Once this transaction has been completed and fully verified, Cannon will add the transaction to the mempool for consensus and execution. The user can now poll chain_queryTx, and once it has executed, it will return:

1. the rsv minting signature that authorizes the minting of cBTC,
2. the nhash which is the keccak256(..) hash of n combined with the UTXO, and
3. the amount of cBTC that is authorized for minting. This is equal to the amount in the UTXO.

When Cannon returns these values, the user now can submit all of the input/out data to the Ethereum blockchain to mint the Cannon BTC (cBTC).

Cannon Claim

Cannon Claim is a function that enables users to redeem the underlying asset in which they hold a DRT for such as cBTC for BTC. This function executes when the user interacts with the Cannon smart contracts to burn the DRT and specifies which address to send the underlying asset back to. This occurs in one transaction and when the Cannon protocol views this burn as complete, included in the burn event, it includes the receiving address.

During any point, a user can burn an arbitrary amount of cBTC on–demand. At the same time they initiate this, they must specify a Bitcoin address. The Cannon Nodes powering Cannon will inevitably witness this burn event on Etheruem, and after 12 block confirmations, they will create a Cannon transaction to release the respective amount of BTC to the users designated Bitcoin address. The user does not have to do anything other than burning the cBTC and ensuring they specify their Bitcoin address. Cannon will automatically witness this burn event and generate a signed Bitcoin transaction to claim the BTC and release it to the specified address.

Chains protocol engages its consensus to agree on the order in which to validate and process burns, the UTXOs used when broadcasting the Bitcoin transaction, and the amount of fees used. To ensure Cannon operates in an optimized manner, Cannon will batch multipole burns into one Bitcoin transaction with multiple outs. Any unspent output will be returned to the Network public key.

Conclusion

Cannon has an architecture designed to preserve privacy and create an interoperable bridge between various digital assets and blockchains to really help unlock the billions of dollars in value between cross-chain applications. The project invites developers to contribute to the open-source project to continue to improve its security and functionality including the addition for more blockchains for cross-chain interoperability with Ethereum. Cannon will enable billions of dollars of potential value to enter the decentralised finance application landscape.

References

1. "What is DeFi", by Coindesk, September 2020, https://www.coindesk.com/what-is-defi
2. "Why Is DeFi the hottest ticket in crypto?", by The Conversation, August 2020, https://theconversation.com/what-is-defi-and-why-is-it-the-hottest-ticket-in-cryptocurrencies-144883
3. Bitcoin Market Capitalisation, https://coinmarketcap.com/currencies/bitcoin/
4. Republic Project or REN Project, https://renproject.io
5. "Non-interactive zero-knowledge proof" Wikipedia, https://en.wikipedia.org/wiki/Non-interactive_zero-knowledge_proof
6. "What is Secure Multi-Party Computation", by Inpher.io, https://www.inpher.io/technology/what-is-secure-multiparty-computation
7. "What are Zero-Knowledge Proofs?", by Wired, September 2019, https://www.wired.com/story/zero-knowledge-proofs/
8. "Byzantine Fault Tolerance Explained", by Binance Academy, https://academy.binance.com/en/articles/byzantine-fault-tolerance-explained
9. "Shamir's secret sharing — A numeric walkthrough example", by Andreas Pgoiatzis, September 2018, https://medium.com/@apogiatzis/shamirs-secret-sharing-a-numeric-example-walkthrough-a59b288c34c4
10. "Zero Knowledge from Secure Multi Party Computations", by Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai, 2007,

https://web.cs.ucla.edu/~rafail/PUBLIC/77.pdf
11. What is Zero Knowledge Storage, https://www.cloudwards.net/what-exactly-is-zero-knowledge-in-the-cloud-and-how-does-it-work/
12. "What are Atomic Swaps?", by Bisola Asolo CryptoCompare, March 2020, https://www.cryptocompare.com/coins/guides/what-are-atomic-swaps/
13. "Syntheix launches Crypto-Backed Synthetic Asset Platform", by Synthtix, December 2018, https://www.businesswire.com/news/home/20181206005225/en/Synthetix-Launches-Crypto-Backed-Synthetic-Asset-Platform-Rebranding
14. "What is tBTC", by DeFi Pulse, August 2019, https://defipulse.com/blog/what-is-tbtc/
15. "What is WBTC?" By Ki Chong Tran Decrypt, April 2020, https://decrypt.co/resources/what-is-wbtc-explained-bitcoin-ethereum-defi
16. RZL-MPC-Specification, by Republic Project, https://github.com/renproject/rzl-mpc-specification
17. Elliptic Curve Digital Signature Algorithm, Wikipedia, https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm
18. "Sharding: What is it and why are so many blockchains rely on it?", By Lucas Merian, January 2019, https://www.computerworld.com/article/3336187/sharding-what-it-is-and-why-so-many-blockchain-protocols-rely-on-it.html
19. "What is an Epoch in Blockchains?", https://www.researchgate.net/figure/Encoding-happens-in-epochs-An-epoch-is-defined-as-the-time-required-for-the-blockchain_fig1_334129900
20. "Tendermint Explained", Cosmos, May 2018, https://blog.cosmos.network/tendermint-explained-bringing-bft-based-pos-to-the-public-blockchain-domain-f22e274a0fdb?gi=813268ff38b6
21. Simplified Payment Verification, Bitcoin WiKi, https://en.bitcoinwiki.org/wiki/Simplified_Payment_Verification
22. Cannon Github, https://github.com/Cannon
23. REN Project, https://github.com/renproject
24. Satoshi Nakmato, https://en.wikipedia.org/wiki/Satoshi_Nakamoto
25. "What is a non-inflationary cryptocurrency?", https://watchcrypto.media/what-is-a-non-inflationary-cryptocurrency/