



# Chameleon Whitepaper

Privacy Mode for Blockchain and Crypto



## Introduction

1. [Introduction: A Platform of Decentralized Privacy Coins](#)
2. [Shielding Cryptocurrencies: Turning Any Cryptocurrency Into a Privacy Coin](#)
3. [Trustless Custodians: A Decentralized Approach to Cryptocurrency Custodianship](#)
4. [Sending Cryptocurrencies Confidentially: Ring Signature, Homomorphic Commitment, and Zero-Knowledge Range Proofs](#)
5. [Privacy at Scale with Dynamic Sharding](#)
6. [Consensus: A Combination of iPoS, Multiview-PBFT, and BLS](#)
7. [Multiview PBFT](#)
8. [Chameleon Software Stack: Navigating the Chameleon Source Code](#)
9. [Chameleon Performance](#)
10. [Network Incentive: CHML](#)
11. [User-Created Privacy Coins](#)
12. [Use Cases: Privacy Stablecoins, Privacy DEX, Confidential Crypto Payroll, and more](#)
13. [Highway: an Upgrade to Chameleon Network Topology](#)
14. [Privacy Mode for dApps on Ethereum](#)
15. [Future Work: Smart Contracts, Confidential Assets, Confidential IP, and more](#)
16. [Conclusions, Acknowledgments, and References](#)



## Introduction: A Platform of Decentralized Privacy Coins

In the near future, anyone will be able to send BTC, ETH, and thousands of other cryptocurrencies to another party without going through a financial institution [Nakamoto, 2008; Buterin et al., 2014]. However, for those who value privacy, these cryptocurrencies will still come with a significant tradeoff. Transactions will continue to be recorded on public ledgers, displaying transaction amounts and inscribing the virtual identities of their senders and receivers. Given the choice, we strongly believe that very few people will willingly disclose their crypto financials to the entire world.

The inherent lack of privacy in cryptonetworks will remain a real and present threat to the entire crypto space.

Existing solutions like Monero, Zcash, and Grin introduced their own version of cryptocurrencies that focus on privacy, based on CryptoNote, Zerocash [[Sasson et al., 2014](#)], and Mimblewimble [[Jedusor, 2016](#)], respectively.

Chameleon will take a different approach, grounded in the idea that people won't want an entirely new cryptocurrency with privacy. What they will truly want is privacy for their existing cryptocurrencies: a "**privacy mode**" for any cryptocurrency.

The Chameleon Network will be designed so that users won't have to choose between their favorite cryptocurrencies and privacy coins. They will be able to have both. Users will be able to hold any cryptocurrency and still use it confidentially whenever they desire. Privacy will be ubiquitous, inclusive, and accessible.




**Figure 1: Chameleon Network as a privacy hub.**

The Chameleon Network will act as a privacy hub, interoperable with other cryptonetworks through [shielding and unshielding processes](#). These processes will allow cryptocurrencies like BTC and ETH to go private and then return to their original form.

Initially, Chameleon will propose a solution to [shield any cryptocurrency](#), such as BTC, ETH, and USDT. Essentially, any cryptocurrency will be able to become a privacy coin. Both shielding and unshielding will be carried out via a decentralized group of [trustless custodians](#). Once shielded, transactions will become confidential and untraceable. To ensure [privacy](#), Chameleon will employ linkable ring signatures, homomorphic commitment schemes, and zero-knowledge range proofs.

Secondly, Chameleon will present a solution for scaling out a privacy-focused cryptonetwork by implementing [sharding](#) on privacy transactions, along with a new [consensus](#) model based on proof-of-stake, pBFT, and BLS. Transaction throughput will scale linearly with the number of shards.

At launch, with 8 shards active, Chameleon Network will aim to achieve a throughput of 100 TPS. Once fully deployed with 64 shards, it will be capable of handling up to 800 TPS – a significantly higher throughput than most other privacy blockchains, which usually handle less than 10 TPS.



This approach will be inspired by Incognito's architecture, which has demonstrated similar scalability and performance.

Chameleon Network will launch its mainnet as a privacy-protecting, high-performance cryptonetwork, delivering "**privacy mode**" for other cryptonetworks like Bitcoin and Ethereum. Based on Incognito's stats from February 2020, Incognito had 8 shards, powered by over 1,000 validators, and had confidentially processed over \$1.4M worth of crypto across 74 different currencies like BTC, ETH, and USDT. Chameleon Network will aspire to achieve similar milestones and extend these capabilities even further.

# Shielding Cryptocurrencies: Turning Any Cryptocurrency Into a Privacy Coin

## Shielding any cryptocurrency into a privacy coin

Shielding will be the process of transforming cryptocurrencies on other cryptonetworks (or “public coins”) into privacy coins on Chameleon.

### Privacy coins

Through the Chameleon Network, a public coin will be shielded to obtain its privacy coin counterpart of the same value. For example, BTC will be shielded to obtain the privacy coin pBTC. pBTC will have the same value as BTC, so 1 pBTC can always be redeemed for 1 BTC and vice versa.

Once shielded, privacy coin transactions will be confidential and untraceable. A privacy coin on the Chameleon Network will offer the best of both worlds: it will retain the value of its original counterpart while being transacted confidentially on the Chameleon Network.

PRIVACY COINS	COUNTERPART	NUMBER OF TRANSACTIONS
pBTC	BTC	432,755
pUSDT	USDT	385,451
pETH	ETH	361,648

**Table 1:** The top three privacy coins on the previous network from 2019 to 2024.

### Shielding

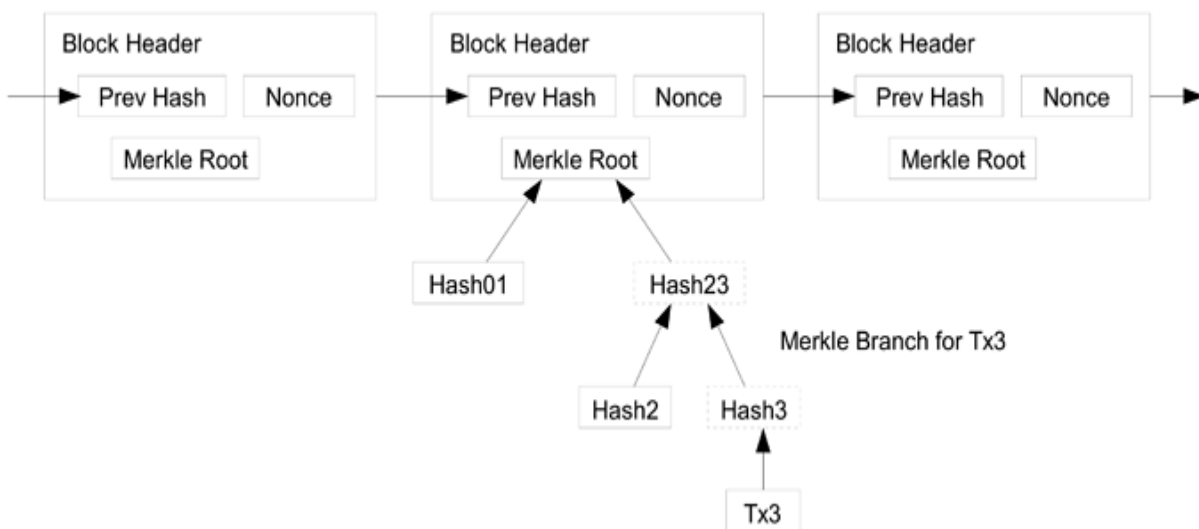
The shielding mechanism will be based on the experience of building the first-generation trustless bridge, inspired by the framework used between Chameleon and Ethereum. It will be generalized to enable a broader range of cryptonetworks to be interoperable with the Chameleon Network.

Current blockchain interoperability solutions will mostly involve building ad-hoc bridges. For example, BTC Relay [BTC Relay, 2019], WBTC [WBTC, 2019], and TBTC [TBTC, 2019] create specific bridges between Bitcoin and Ethereum, while Kyber Network builds Waterloo [Baneth, 2019], an ad-hoc bridge between Ethereum and EOS. For Chameleon, doing it ad hoc—one bridge for every cryptonetwork—will not be a scalable option.

Chameleon Network will take a different approach: build once, and work with any cryptonetwork. The shielding mechanism will operate via a general bridge design that connects Chameleon to a wide range of cryptonetworks, enabling secure **bi-directional** transfers of cryptocurrencies whenever privacy is required. This means any coin will be able to transform into a privacy coin. This approach will be particularly valuable for creating interoperability with cryptonetworks that do not support smart contracts, such as Bitcoin and Binance Chain.

To obtain privacy coins, the user will first submit a shielding request to the Bond smart contract with details about which public coins they wish to shield and the amount. The Bond smart contract will then select [trustless custodians](#) for the public coins and provide the user with the custodians' deposit addresses. Once the deposit is confirmed on the cryptonetwork of the public coins, the user will initiate a shielding transaction on Chameleon along with the deposit proof. A deposit proof on a cryptonetwork will typically be a **Merkle branch** that links the deposit transaction to the block in which it is time-stamped, ensuring that the deposit transaction has been accepted by that cryptonetwork.

Longest Proof-of-Work Chain

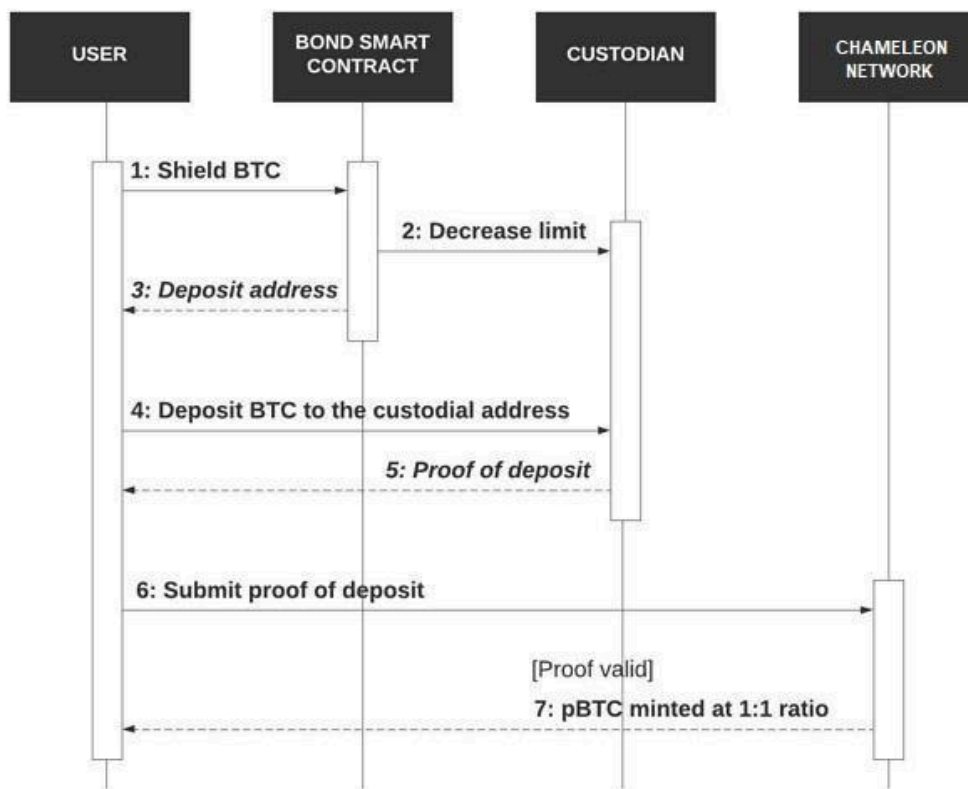


**Figure 1.** SPV in Bitcoin [Nakamoto, 2008]. Other cryptonetworks employ similar SPV methods.

Note that while we have designed a general bi-directional bridge between Chameleon and other cryptonetworks, we still need to implement the specific SPV logic for each cryptonetwork we add to the bridge. This includes relaying block headers from those cryptonetworks to Chameleon and performing SPV on deposit proofs.


Chameleon validators will verify the shielding transaction and the deposit proof inside it, specifically using Simplified Payment Verification [Nakamoto, 2008]. Most cryptonetworks, including the Chameleon Network, will support Simplified Payment Verification with a few small differences in the underlying data structures. For example, Bitcoin and Binance will implement Merkle Tree [Merkle, 1980], while Ethereum will implement a modified Merkle Patricia Tree [Wood, 2014].

Once the deposit proof is verified, new privacy coins will be minted at a 1:1 ratio.



**Figure 2.** Shielding BTC and minting pBTC in the Chameleon Network.





*Other public coins follow the same shielding process. Note that we simplify step 5 to make it easier for readers to follow the main logic: the proof of deposit is not generated by the custodian but by the miners of the underlying crypto network.*

## Unshielding

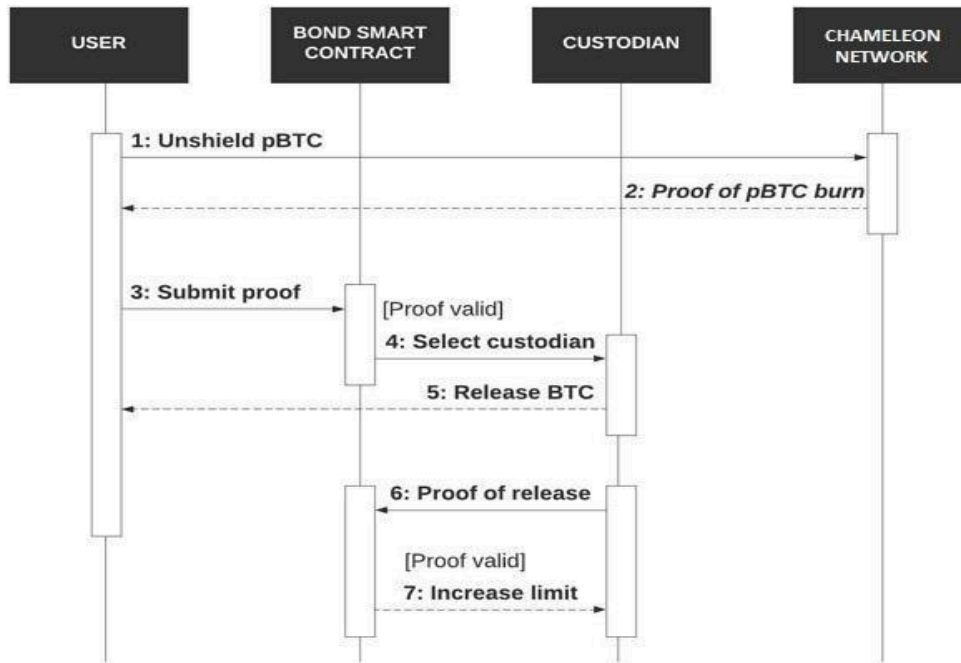
**Unshielding will be the reverse process of shielding:** turning privacy coins back into public coins.

The user will initiate an unshielding transaction on Chameleon with information about which privacy coins they want to unshield and the amount.

Chameleon validators will verify the unshielding transaction, burn the privacy coins, and issue a burn-proof. A burn-proof on Chameleon will be cryptographic proof. When signed by more than  $\frac{2}{3}$  of Chameleon validators, it will prove that the privacy coins have been burned on the Chameleon network.

The user will then submit the burn-proof to the Bond smart contract, which will verify the burn-proof and instruct a custodian to release the public coins that back those privacy coins at a 1:1 ratio. Once the release is confirmed on its respective crypto network, the custodian will submit the released proof to the Bond smart contract. Similar to the deposit proof, a release proof will be a Merkle branch linking the release transaction to the block it is time-stamped in, proving that the release transaction has been accepted by that crypto network.

After verifying the released proof, the Bond smart contract will free up the custodian's collateral. Custodians will be able to withdraw their collateral or start taking new user deposits.



**Figure 3. Unshielding pBTC and releasing BTC.** Other public coins follow the same unshielding process.

We will propose a mechanism for turning cryptocurrencies on other cryptonetworks (or “public coins”) into privacy coins, based on a set of [trustless custodians](#). Once shielded, privacy coin transactions will be confidential and untraceable. A privacy coin will enjoy the best of both worlds: it will retain the value of its original counterpart and can be transacted confidentially on the Chameleon Network.

An exception will be addressed in the Auto-Liquidation section in the [Trustless Custodians](#).

# Trustless Custodians: A Decentralized Approach to Cryptocurrency Custodianship

## A Decentralized Approach to Cryptocurrency Custodianship

Custodians will play a crucial role in the **shielding** mechanism that will turn cryptocurrencies—like BTC, ETH, and USDT—into [privacy coins](#).

Existing custodian solutions, such as Bitgo and Coinbase Custody, will remain centralized and expensive. Trusted third parties will continue to pose security vulnerabilities [\[Szabo, 2001\]](#). Moreover, centralized custody will require users to share their private information with third parties.

**Chameleon** will take a decentralized approach to custodianship. Instead of relying on a single centralized authority like Bitgo or Coinbase Custody, the Chameleon Network will utilize multiple custodians. Anyone will be able to become a custodian by simply supplying a bond.

A smart contract on Ethereum will be implemented to manage bonds and ensure that the system operates exactly as programmed. Not only will the Bond smart contract be trustless, but it will also provide real-time processing, in contrast to the multi-day manual processes used by centralized custodian companies.

Initially, a fixed custodian fee structure will be implemented for simplicity. However, this could be further enhanced by adopting a market-driven custodian fee model, where users will set their own fees, and custodians will compete for user deposits.

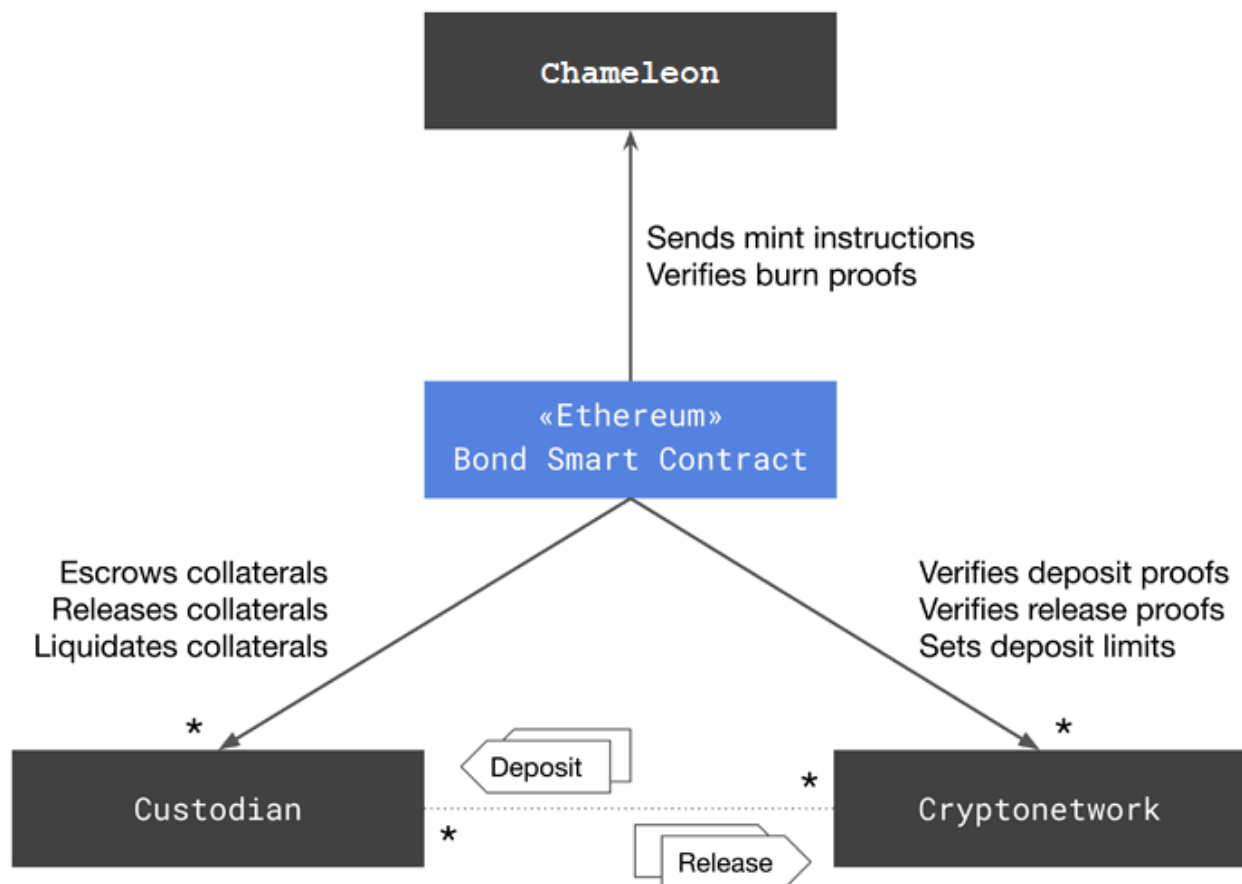
Feature	Chameleon	BITGO
Privacy	✓	✗
Trustless	✓	✗
Insured by collateral	✓	✗
Processing time	Instant	Days
Fees	Low	High

**Table 1:** A comparison between Chameleon and centralized custodians like Bitgo and Coinbase Custody.

## The Bond Smart Contract

The Bond smart contract will serve as the bridge between Chameleon, custodians, and other cryptonetworks like Bitcoin and Binance Chain. There will be multiple implementations of the Bond smart contract on different cryptonetworks, including Chameleon itself, using their respective cryptoassets as collateral.

The first implementation will be programmed as an Ethereum smart contract [Wood, 2014]. Ethereum will be chosen because its smart contract platform is battle-tested, and it offers a wide range of liquid cryptoassets suitable as collateral.



**Figure 1.** The Bond smart contract is programmed to glue together custodians, Chameleon, and other cryptonetworks like Bitcoin and Binance Chain.



The Bond smart contract will be programmed to:

1. **Escrow collateral** in ETH and liquid ERC20 tokens bonded by custodians.
2. **Set the maximum total amount** of user deposits a custodian can accept based on the Collateral-to-Deposit ratio.
3. **Verify deposit proofs** on other cryptonetworks and submit valid proofs to Chameleon for minting privacy coins.
4. **Verify burn proofs** of privacy coins on Chameleon and instruct custodians to release public coins.
5. **Verify custodians' release proofs** on other cryptonetworks and free up their collateral, enabling custodians to withdraw their collateral tokens or take new user deposits.
6. **Liquidate collateral** if necessary, in accordance with the protocol's rules for protecting user funds.

## Over-Collateralized Bonds

Custodians will be required to bond collateral into the Bond smart contract. Bonded collateral tokens will serve as a safeguard, providing recourse when custodians misbehave. The Bond smart contract will only accept ETH and liquid ERC20 tokens as collateral.

Because cryptocurrency prices will remain volatile, the value of bonds will also fluctuate. To address this, custodians will need to overbond, ensuring that the total amount of user deposits to a single custodian will not exceed the total value of the custodian's bonded collateral.

A parameter  $\alpha$  will be introduced, initially set at 200%. This **Collateral-to-Deposit** ratio will ensure that user deposits are always fully backed, even in the event of a significant drop in collateral value.

For example, as a custodian, Alice will need to bond at least \$2,000 worth of ETH and ERC20 tokens into the Bond smart contract if she wishes to accept \$1,000 worth of BTC user deposits.

## Auto-Liquidation

Over-collateralization will ensure that custodians do not misbehave.

During the **unshielding** process, if Alice fails to send the public coins back to Bob in full, Alice's bonded collateral will be used to repay Bob. In this case, the public coins Bob receives—Alice's collateral, to be precise—may differ from Bob's original public coin, but their total value will be the same or greater than the value of Bob's original deposit.

Auto-liquidation also triggers if the value of bonded collateral drops significantly. Custodians must add more collateral to avoid auto-liquidation. We introduce a parameter  $\beta$ , initially set at **120%**. If  $\alpha$  is the upper bound,  $\beta$  is the lower bound or the liquidation threshold.  $\beta$  ensures that total custodian collateral amounts do not fall below total user deposits.

A future improvement could involve automatically liquidating collateral on a decentralized exchange such as **Kyber** [Luu and Yaron, 2017], **Uniswap** [Adam, 2018], or **Chameleon pDEX**.

## Incentives

First, custodians will earn shielding fees and unshielding fees. The initial fee structure will be straightforward, with a fixed shielding fee of **0.1%** and an unshielding fee of **0.1%**.

In the future, as part of a market-driven pricing structure, users could set their own fees, and custodians could prioritize processing transactions with the highest fees. A more advanced fee structure might also consider shielding, unshielding, and custodial times.

Second, custodians will earn **CHML**, the native coin of **Chameleon**, through **shield mining**. In traditional cryptonetworks, mining rewards come solely from block mining, where miners earn rewards for producing new blocks. In Chameleon, custodians will participate in shield mining alongside block mining, earning **CHML** for shielding public coins. The more a custodian shields, the greater the **CHML** rewards they earn. The Chameleon DAO funds shield mining rewards.

The shield mining reward  $r_i$  for a custodian  $c_i$  at block height  $k$  is computed as follows, where  $R_k$  is the total shield mining reward for that block,  $n$  is the number of custodians, and  $b_i$  is the bonded collateral value from custodian  $c_i$ .

$$r_i = \frac{b_i}{\sum_{j=0}^{n-1} b_j} * R_k$$

This design outlines a decentralized approach to custodianship. While this mechanism is designed specifically for **Chameleon**, it is hoped that the community will find it valuable and expand upon it to develop more fully decentralized systems of custodians.

---

# Sending Cryptocurrencies Confidentially: Ring Signature, Homomorphic Commitment, and Zero-Knowledge Range Proofs

## Sending Cryptocurrencies Confidentially

Once public coins are shielded, they will be confidentially sent, received, stored, and traded as privacy coins. Custodians may retain records of the total number of privacy coins minted but will have no visibility into how these privacy coins are utilized afterward. Every privacy coin transaction will remain confidential and untraceable, even to custodians and validators. Chameleon will employ advanced cryptographic primitives, including linkable ring signature schemes, homomorphic commitment schemes, and zero-knowledge range proofs, to obscure sending addresses, receiving addresses, and transacted amounts.

## Fungibility: The Basis of Monetary Privacy

All privacy coins issued on the **Chameleon** network will be fungible—fulfilling one of the fundamental requirements of money. A unit of currency must be identical and interchangeable with another.

## Ring Signatures: Shielding Sending Addresses

A ring signature scheme will enable a member of a group to sign a message on behalf of the group without revealing the signer's identity [Chaum and Van Heyst, 1991; Fujisaki and Suzuki, 2007; Van Saberhagen, 2013]. Signer anonymity is maintained by ensuring each group member has an equal probability of being the actual signer.



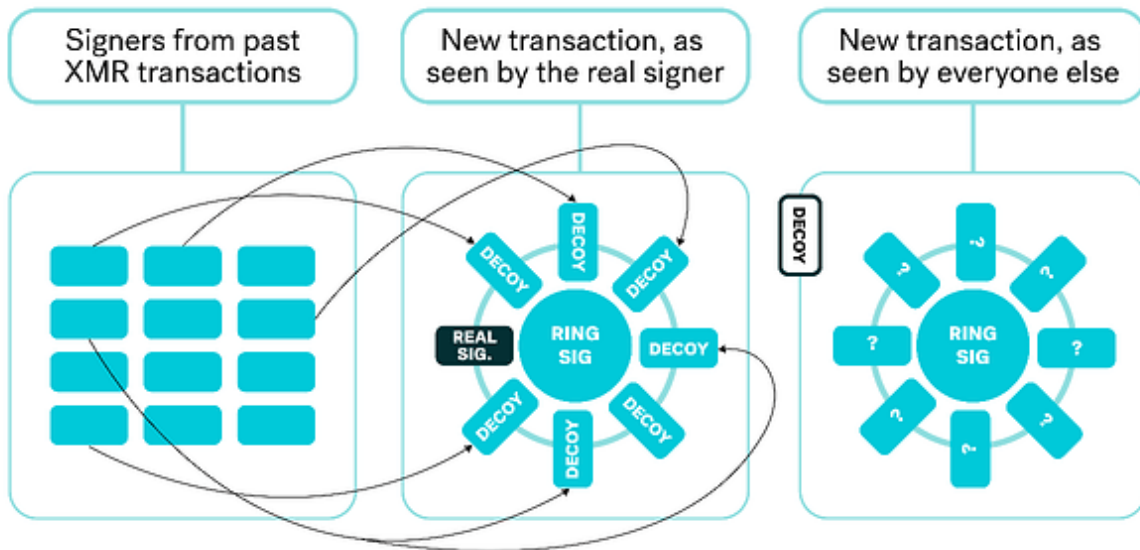
**Figure 1. The identity of the signer is obscured.** For example, if you encounter a ring signature with the public keys of Annie, Bob, John, and Peter, you will be able to claim that one of these users is the signer, but not pinpoint him or her.

Group formation in a ring signature scheme will be **spontaneous**, with no centralized manager to uncover the identity of the true signer. Due to these characteristics, such groups are referred to as **ad hoc groups** or **rings**. A signer will form a group by collecting public keys from other group members. These additional group members, known as **decoys** or **mixins**, will be drawn from historical transactions. The unified signature generated by the ring provides anonymity for the true signer.

In **Chameleon**, ring signatures will be utilized to authorize the spending of an **Unspent Transaction Output (UTXO)** [Nakamoto, 2008], without exposing the identity of the spender. Each ring will comprise the actual UTXO being spent along with its decoys, which will consist of random outputs from historical transactions. Together, the actual UTXO and its decoys will form the transaction inputs.

From a public perspective, any of these inputs could plausibly be the actual output being spent. This uncertainty ensures that the true spender remains indistinguishable within the group, thereby safeguarding the privacy of the transaction.





**Figure 2. Visualization of ring signature.** The notion of ring signature was first proposed as a way of whistleblowing [Rivest et al., 2001]

Since it will be impossible to determine which UTXO is being spent in a ring signature scheme, there is the potential for a **double-spending problem** [Finney, 1993]. To mitigate this, Chameleon will implement a variant of the ring signature called **Linkable Ring Signature** [Liu et al., 2004]. This approach adds a crucial property: **linkability**.

With linkability, any signature issued under the same public key—whether for the same or different messages—will have a unique identifier called a **serial number**. These serial numbers enable verification of whether two signatures originate from the same group member without disclosing the signer’s identity.

In Chameleon's implementation, a **serial number** will be derived from each UTXO being spent and included in every ring signature. To prevent double-spending, a **list of used serial numbers** will be permanently stored as part of the transaction data. If a new ring signature attempts to reuse an existing serial number, it will be automatically rejected, ensuring robust protection against double-spending.

This combination of anonymity and accountability ensures that privacy is preserved while maintaining the integrity of the Chameleon network.

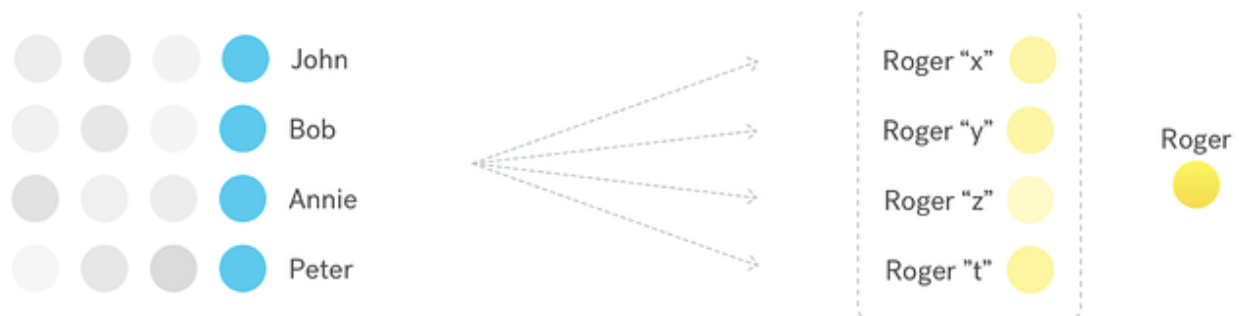
## Stealth Addresses: Shielding Receiving Addresses

In traditional cryptonetworks like Bitcoin or Ethereum, a **public address** is all that is required to view the complete history of incoming and outgoing transactions associated with that address [Reid and Harrigan, 2013]. This level of transparency can reveal **total balances, spending patterns**, and other details, making it easy to **link transactions**.


To prevent such linking and ensure privacy, **Chameleon** introduces **stealth addresses**—a type of one-time public key. For every incoming transaction, Chameleon automatically generates a unique one-time public key. These stealth addresses act as **one-time deposit boxes**, ensuring that:

1. **Only the intended receiver** can access the contents of the deposit box.
2. Each incoming transaction appears independent, making it impossible to correlate transactions or infer the receiver's total balance.

By using stealth addresses, Chameleon ensures that the **sender, receiver, and transacted amount remain confidential**. This cryptographic mechanism significantly enhances **user privacy** while maintaining the flexibility and functionality of a decentralized network.



**Figure 3.** Creating multiple unique one-time keys



Stealth addresses in Chameleon are built on the **Diffie-Hellman key exchange protocol** [Diffie and Hellman, 1976], a cryptographic method enabling two users to create a shared secret, even in the presence of an eavesdropper monitoring all communications.

A **Chameleon address** is composed of:

- A **public view key**, which is used to receive transactions.
- A **public spend key**, which is paired with a **private spend key** to authorize outgoing transactions.

### How Stealth Addresses Work

#### 1. Transaction Setup

When Alice wants to send privacy coins to Bob, she uses:

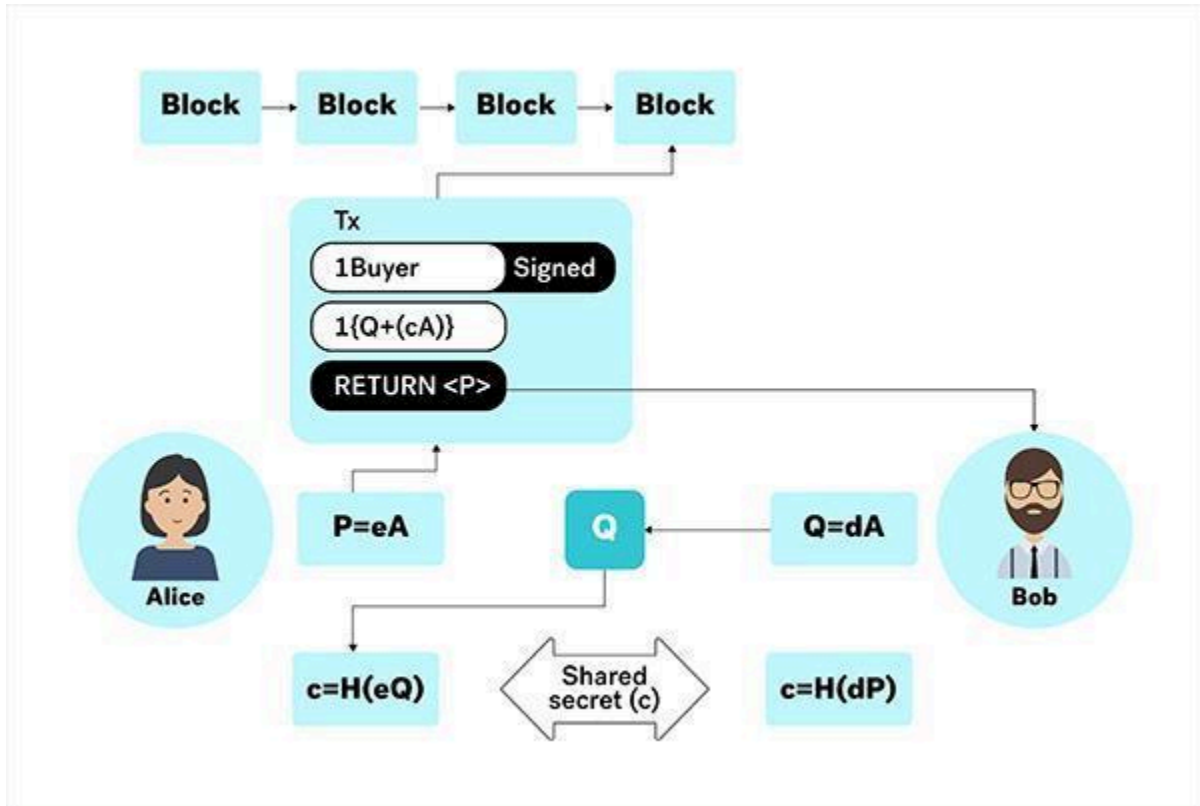
- Bob's **public view key** and **public spend key**.
  - A piece of **fresh randomness**.
2. Using these inputs, Alice derives a **one-time public key** for Bob's new **UTXO**. This derivation is done such that **only Bob** can compute the corresponding one-time private key.
- #### 3. Receiving the UTXO
- Bob scans all incoming transactions using his **private view key** to identify the UTXO intended for him.
  - Once identified, Bob computes the **one-time private key** corresponding to the one-time public key.
- #### 4. Spending the UTXO
- Bob can spend the UTXO using his **private spend key**, maintaining control over his funds.

### Privacy Benefits

- The **transaction data** is recorded on the Chameleon public ledger, allowing anyone to see that a transaction occurred.
- However, the **one-time public key** in the transaction cannot be linked to Bob or his Chameleon address.

For example, if Bob is a merchant, **observers cannot determine that he and Alice are conducting business**, preserving their privacy.

Stealth addresses ensure that Chameleon transactions remain unlinkable and confidential, even in a fully transparent public ledger environment.



**Figure 4. Stealth addresses**

The transaction data is on the **Chameleon** public ledger. Anyone can see that a new transaction has occurred, but cannot link the one-time public key in the transaction to Bob. If Bob were a merchant, for example, no one would be able to determine that he and Alice are doing business together.

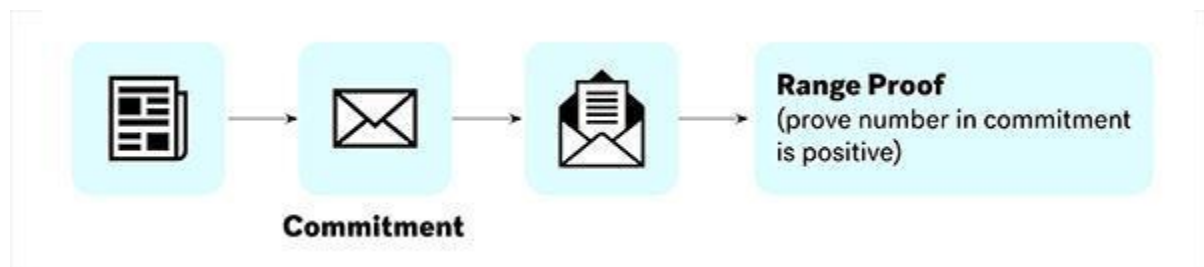
## Confidential Transactions: Shielding Transacted Amounts

**Confidential transactions** are employed to shield the transacted amounts on the Chameleon public ledger [Maxwell, 2015]. While the occurrence of privacy coin transactions remains visible, the exact amounts involved are concealed.



**Figure 5. Confidential transaction amount**

The fundamental approach involves committing both input and output amounts of a transaction using **Pedersen commitments** [Pedersen, 1991]. A commitment combines the value of a transaction with a **blinding factor**, which serves as randomness that prevents others from determining the value. The value and blinding factor can later be revealed by the committer, allowing others to verify the validity of the commitment.



**Figure 6. Commitments are equipped with zero-knowledge range proofs to prove their validity**



## Addressing Validation Challenges


The first challenge arises when validators can no longer verify the transaction due to the inability to confirm that the sum of inputs matches the sum of outputs. To resolve this, **zero-knowledge proofs** [Goldreich et al., 1991] are integrated into each transaction, enabling the prover to demonstrate knowledge of a statement's truth without revealing any additional details beyond its validity.

Thanks to the **homomorphic** property [Gentry and Boneh, 2009], all input commitments can be aggregated into a single input commitment, and all output commitments can be aggregated into a single output commitment. The sum of these commitments represents a commitment to the total value, with the blinding factor serving as the sum of individual blinding factors in the commitments.

A **commitment to zero** emerges as a valid public key, with its corresponding private key being the blinding factor. The sender then signs the difference between two such commitments, proving that the balance has been maintained. By including this commitment in the ring signature, the sender can demonstrate the validity of the transaction while using the blinding factor as one of the private spend keys.

## Preventing Inflation through Range Proofs

The second issue is the potential for an attacker to generate coins arbitrarily and inflate the supply of privacy coins by committing to negative amounts. To mitigate this, each output commitment is paired with a **range proof** [Boudot, 2000; Morais et al., 2019]. A range proof verifies that the output amounts lie within a positive range, specifically within the interval  $[0, 2^{64})$ , without disclosing the actual amounts.



Validators can now confirm the legitimacy of a transaction without needing to know the exact amounts being transferred. To implement these range proofs efficiently, **Bulletproofs** [Bünz et al., 2018] are utilized. Bulletproofs are short, non-interactive zero-knowledge proofs designed to enable confidential transactions without requiring a trusted setup. The size of the range proof is significantly reduced from approximately 5KB to just 700 bytes, enhancing efficiency. Furthermore, Bulletproofs support aggregation, meaning that combining several range proofs results in only a minimal increase in size.

## Scaling Blockchain Privacy with Dynamic Sharding

Low throughput is a significant barrier to the widespread adoption of cryptocurrencies. Throughput, measured in transactions per second (TPS), reflects the network's capacity to process and confirm transactions. For instance, Bitcoin can handle only 7–10 TPS [Croman et al., 2016; Li et al., 2018], while Visa processes up to 24,000 TPS [Visa, 2018].

Privacy-focused transactions, however, face even greater challenges due to the additional computational overhead required for proof generation and verification. These transactions also tend to be larger due to the inclusion of extra privacy-related data. Zcash, a privacy-focused Bitcoin fork, can process only 6 TPS due to its 2 MB block size, a 150-second target block interval, and average transaction sizes of 2000 bytes. Such constraints severely limit scalability.

### Chameleon's Solution: Sharding for Scalability

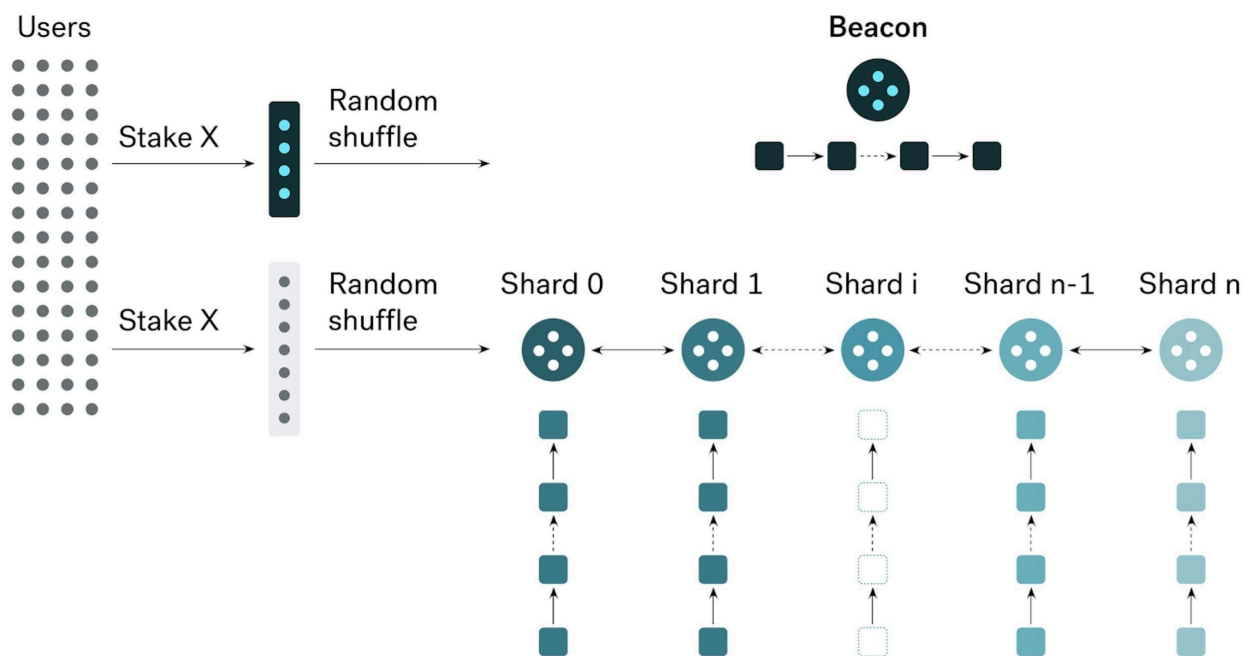
To address these throughput issues, the **Chameleon Network** implements **sharding** for privacy transactions, drawing inspiration from systems like **Omniledger** [Kokoris-Kogias et al., 2018], **RapidChain** [Zamani et al., 2018], and **Zilliqa** [Zilliqa, 2017]. Sharding allows for **linear scalability**, increasing throughput proportionally as more shards are added to the network.

The Chameleon Network is architected as a network of blockchains, consisting of:

1. **A Beacon Chain** (acting as the "coordinator")
2. **N Shard Chains** (serving as the "workers")

The shard chains operate in parallel, independently producing blocks at the same time. The beacon chain plays a crucial role in synchronizing these shard chains,

ensuring consistency and coordination across the network. The block production process is divided into equal **epochs**, with each epoch ensuring the synchronization of all shard chains. This design enables Chameleon to achieve high throughput while maintaining strong privacy guarantees for its transactions.



**Figure 1. Sharding on privacy transactions.** Chameleon throughput scales out linearly with the number of shards.

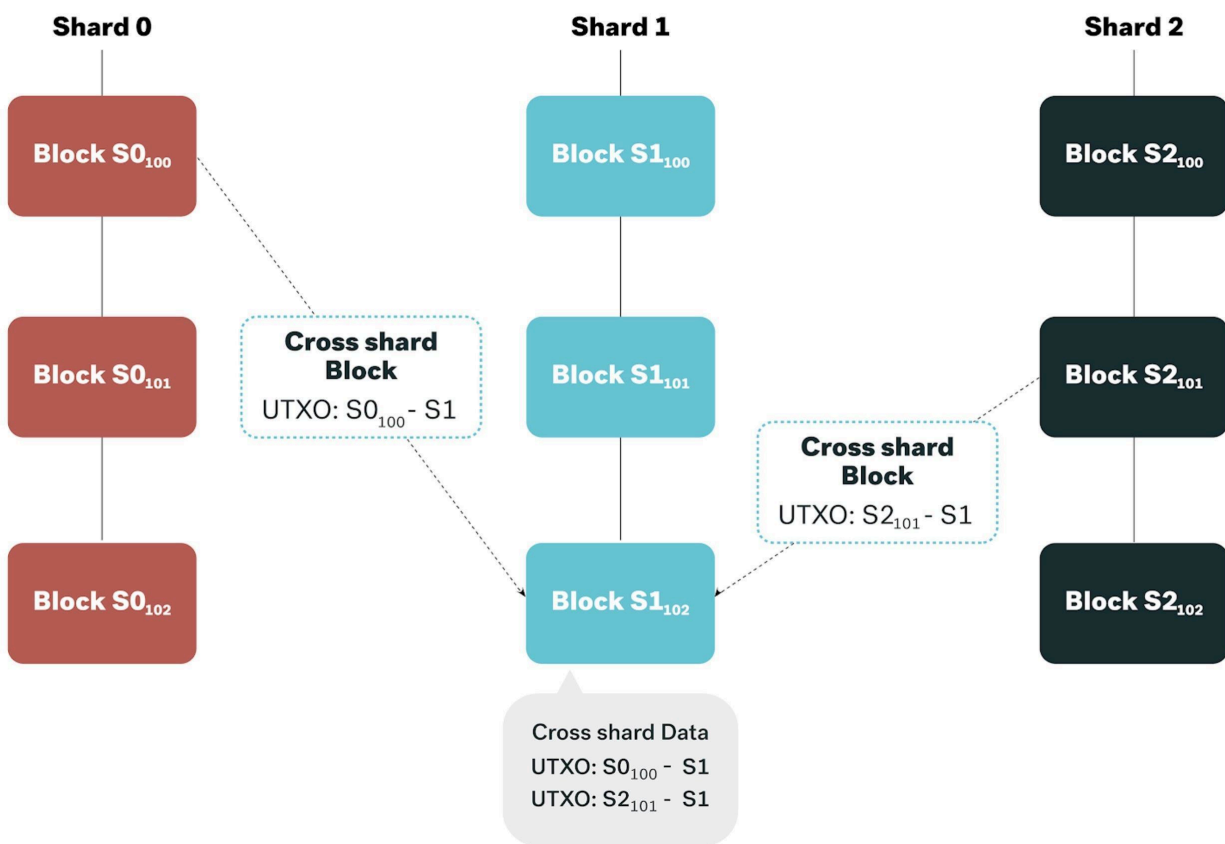
## Shard Chains

Shards will be organized based on the last byte of sender addresses. Each shard will have its own committee, which will be randomly assigned by the beacon chain. This shard committee will be responsible for validating and confirming transactions within its shard.



Whenever a shard block is created, the beacon committee will verify the block's validity and insert the valid block header into the beacon chain.

If the block is found to be invalid, the beacon chain will send proof to all other shards, prompting them to vote on slashing the misbehaving shard committee.



**Figure 2. Shard Chains**

## Beacon Chain

The responsibility of the beacon chain will be to verify shard blocks and coordinate **shard** chains, acting as the global state of the entire network.

- The beacon chain will verify the **validity** of shard blocks.
- It will confirm **cross-shard** information. Each shard block header will include cross-shard information, indicating which shard this block has interacted with. This will also include the height of each shard chain, which will be part of the block body.
- The beacon chain will manage the candidate and validator list. Whenever a user stakes **CHML** to become a validator, this action will be recorded in the block header.
- The beacon chain will **shuffle committees**. When a new random number is generated, it will be recorded in the beacon block header.
- The beacon block will store the Merkle root of the candidate list and the validator list, both for the beacon chain and the shard chains.

## Cross shard transaction

For cross-shard transactions, the sender shard will generate a receipt detailing all transactions directed to the receiver shard and will forward this receipt to the receiver shard. A summary of the cross-shard transactions will also be sent to the beacon chain. To prevent double-spending, the **UTXOs** in the sender shard will be locked. The receiver shard will validate the receipt and will wait for confirmation of cross-shard information from the beacon chain before marking the corresponding UTXOs as spendable.

## Dynamic Committee Size

At the start of an epoch:

- **Substitute List at 100-400%:** Committee size remains the same, but 10% of members are replaced in a round-robin manner.
- **Substitute List > 400%:** Committee size increases by 15% (adjusted for previously slashed members).
- **Substitute List < 100%:** Committee size decreases by 10%.

## Dynamic Sharding

The **Chameleon chain** will initially be implemented with 8 shards. The number of shards could dynamically increase, up to 256 shards. Let  $X$  be the last byte of the sender's public keys. The shard with  $id = X \% \text{number\_of\_shards}$  will handle the transaction.

Each shard will have a maximum ( $M$ ) of committee members. When the substitute list of all shards exceeds  $5M$ , the chain will double the number of shards. In this case, each shard will be split into two new shards.

### Example:

In the case of 8 shards, public keys with  $lastbyte = 0, 8, 16, 24, 32, 40 \dots 240, 248$  belong to shard 0. I.e.,  $shard\_id = lastbyte \% \text{number\_of\_shards}$ .

If doubled to 16 shards, shard 0 will be split into shard 0 and shard 8, which are called sibling shards. The public keys with  $lastbyte = 0, 16, 32, \dots 240$  are in shard 0, and public keys with  $lastbyte = 8, 24, 40, \dots 248$  are in shard 8.

The current committee and substitute nodes in shard 0 will be split equally into shards 0 and shard 8. This way, all committee members already have a full database to confirm any new transactions belonging to them.

If the committee size is smaller than the minimum committee size threshold, two sibling shards will be merged into one shard.

### Example Scenario:

Let's look at a scenario where there are 8 shards, a committee size of 50, and 200 nodes in the substitute list. Table 1 summarizes the probability of conquering the chain, given the percentage of nodes owned by the attacker.

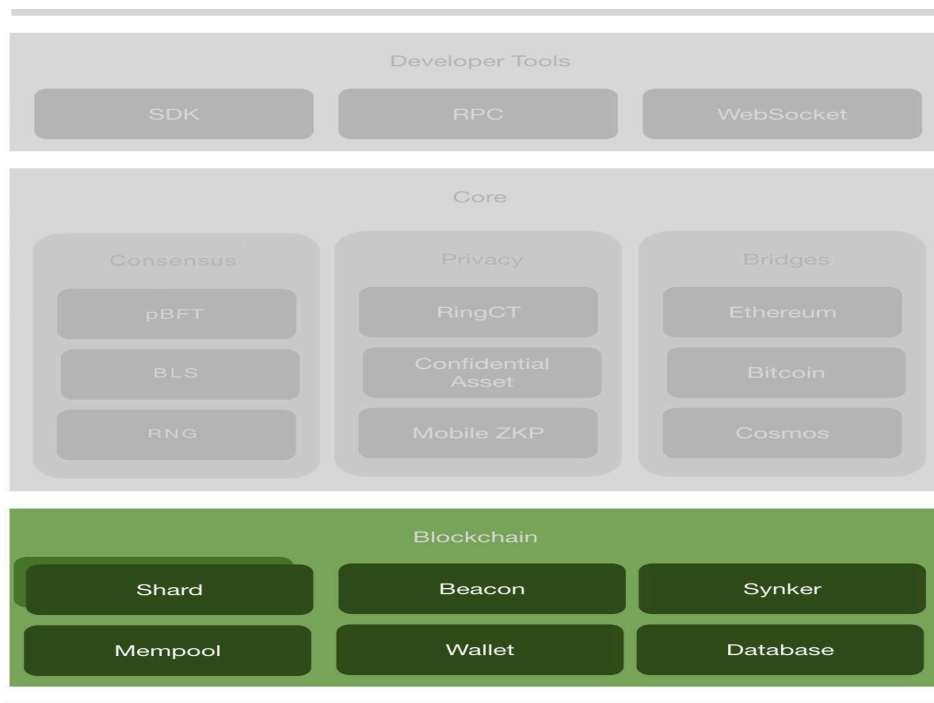
Percentage of byzantine validators	20%	25%	30%	35%	40%
Probability of conquering the chain (8 shards)	$2.04 \times 10^{-107}$	$2.4439 \times 10^{-76}$	$9.6104 \times 10^{-56}$	$9.6104 \times 10^{-41}$	$2.4814 \times 10^{-29}$
Probability of conquering the chain (16 shards)	$4.163 \times 10^{-214}$	$5.973 \times 10^{-152}$	$9.236 \times 10^{-111}$	$9.1834 \times 10^{-81}$	$6.1575 \times 10^{-81}$

**Table 1. Probability of the attacker conquering the chain.**


The probability of conquering the chain is extremely low, even if an individual owns 40% of validators.

## Implementation

The implementation is mainly in the Blockchain component of the Chameleon architecture.



**Figure 14. The layered Chameleon architecture.**



The code is going to be open-source on GitHub, with links to specific packages provided below.

- **Shards:** Shards are subchains. A subchain is a Proof-of-Stake blockchain with its own committee of **N** nodes. A shard's job is to produce new blocks via the Multiview Practical Byzantine Fault Tolerance (pBFT) consensus algorithm.
- **Beacon:** Beacon is also a subchain. A beacon's job is to coordinate the shards and maintain the global state of the network.
- **Synker:** Synker ensures the node stays up to date among its peers and broadcasts the node status to its peers.
- **Mempool:** Mempool (memory pool) is a collection of transactions and blocks that are verified but not yet confirmed.
- **Wallet:** Software that will hold all your Chameleon keys. It will be used to send and receive Chameleon tokens.
- **Database:** Chameleon will use LevelDB to store block data.

## Consensus: A Combination of iPoS, Multiview-pBFT, and BLS

Both beacon chains and shard chains will use the same consensus mechanism to produce new blocks.


### Chameleon Proof-of-Stake (iPoS)

Chameleon will implement the more **energy-efficient Proof-of-Stake (PoS)** [King and Nadal, 2012] instead of **Proof-of-Work (PoW)** [Dwork and Naor, 1992; Juels, 1999]. In the first generation of **PoS**, each **staking node (validator)** will have one vote. The more **validators** there are, the more secure the chain becomes. However, **PoS** will not be a scalable consensus; communication overhead will increase linearly with the **committee size**. Research will show that **PoS** can only achieve a reasonable level of security when there are **600 validators** or more in the committee [Luu et al., 2016]. A **committee** of 600 nodes will face real challenges in syncing statuses and exchanging messages to create a new block.

To overcome the scaling-related communication problems, **Delegate Proof of Stake (DPoS)** will be proposed, initially by **Bitshares** (<https://bitshares.org/>). This will be considered the **second generation of PoS**. In **DPoS**, only a small group of **validators** will be selected into the **committee**, giving them the right to **propose** and **verify new blocks**. This approach will solve the communication problem of **PoS**, but it will sacrifice some **decentralization** and **trustlessness** properties, specifically:

- **Staking nodes** will have to trust **delegated nodes**.
- The smaller the **committee size**, the easier it can be compromised.
- Only **delegated nodes** will work (i.e., **propose & verify blocks**), while **staking nodes** will not participate in block creation.

**Chameleon** will propose **iPoS (interactive Proof-of-Stake)**, which will enable both **scalability** and **trustlessness**. Anyone will be able to become a **validator candidate** by staking **CHML**. The **beacon chain** will randomly assign **validators** to each **shard**. At any given moment, only a small group of **validators** in a **shard** will be selected to form the **committee**. The **committee** for each



**shard** will be responsible for **proposing** and **verifying blocks**. The **committee** will rotate periodically, ensuring every **validator** contributes equally.

As for **decentralization**, each **shard block** signed by the **shard committee** will be verified by the **beacon committee**. If any incorrect transaction, such as an **airdrop** or **double-spending**, is detected, the **beacon chain** will inform all other **shards** using **proof**. All honest **shards** will vote to **slash** the **byzantine shard's committee**.

## Multiview Practical Byzantine Fault Tolerance (M\_PBFT)

**Chameleon Network** will propose and implement a variant of **pBFT (practical Byzantine Fault Tolerance)** [Castro et al., 1999] at the **consensus layer**. We will further improve its efficiency by employing the **BLS-based aggregate multi-signature scheme (AMSP)** [Boneh et al., 2018].

**Tendermint**, a popular implementation of **pBFT**, requires participants to have the **same view** for every block minted. **Nodes** must sync their **status** at every block, which causes **communication overhead**. **Chameleon** will propose **multiview pBFT**, whereby a **node** makes decisions based on its **best view** and does not require the syncing status of other **nodes** in the **committee**. Find more details on **multiview pBFT**.

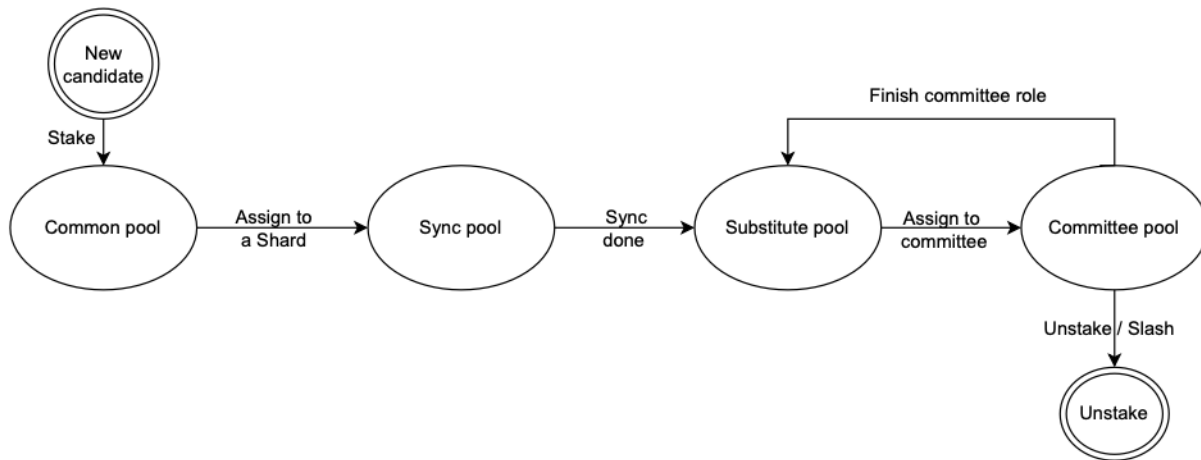
## Validator Life Cycle

**Common pool:** new staking node, waiting to be assigned to a particular shard  
Sync pool: sync the assigned shard's data

**Substitute pool:** node finished sync its data, queueing in the substitute list of its shard

**Committee pool:** validate and vote blocks

4 states: new (in common pool), candidate (in sync pool), substitute (in substitute pool), committee (in committee).



**Figure 1. Life cycle**

## Slashing Rules

For **Chameleon Network**, we will prefer a **light punitive approach** when it comes to **slashing** — i.e., a **slashing policy** that does not deduct any **CHML** from the **stake** or **rewards** of the **validators**. However, the policy must effectively prevent **misbehavior** and **unreliability**.

For each shard:

1. If  $\text{MissingSig} \leq \text{ExpectedShardBlock} / 2$ , then the node will be forced to unstake.
2. If  $\text{MissingSig} > \text{ExpectedShardBlock} / 2$ , then the node will not receive any penalty.

The **ExpectedShardBlock** is calculated as follows:

Let **M** be the mean number of blocks created by the shards in an epoch. For each shard:

- If the number of blocks confirmed is smaller than **M**, then **M** becomes the **ExpectedShardBlock** for that shard.
- Otherwise, the **ExpectedShardBlock** is the number of blocks confirmed by the **Beacon chain**.

If a node is slashed, the **1750 staked CHML** would be returned to the node operator, who may choose to **re-stake** at a later time. **Rewards** from slashed nodes will be distributed evenly to the remaining **validators** in the **committee**.



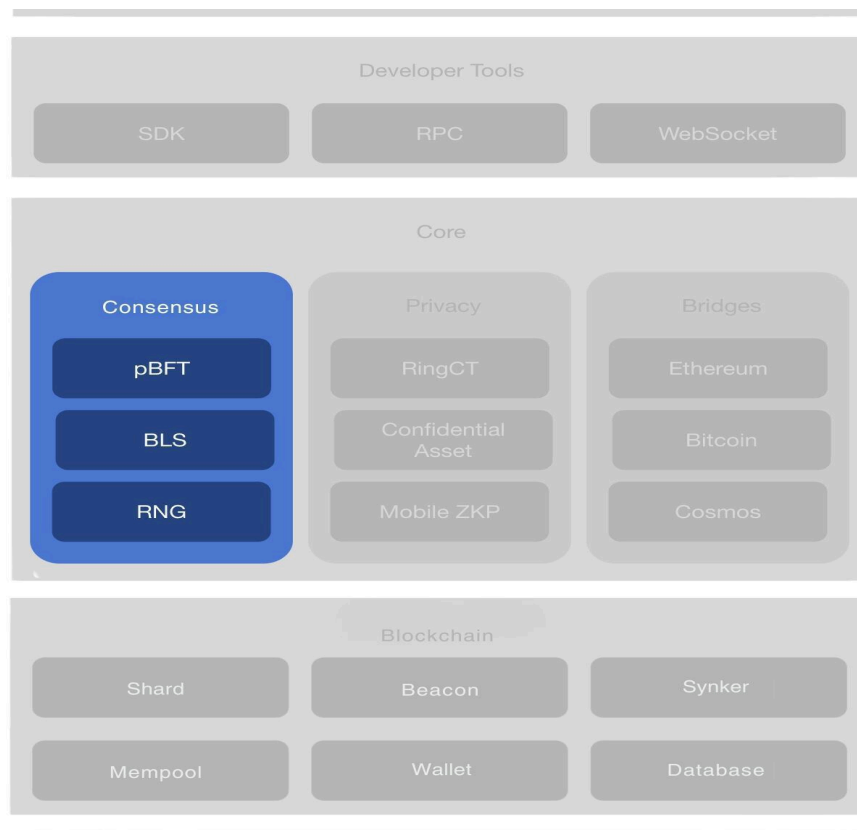
## BLS-based Aggregate Multi-Signatures from Pairing

As the number of **validators** grows, the total size of all **validator signatures** also increases, impacting the **block size**. To solve this problem, we implement the **BLS-based aggregate multi-signature scheme (AMSP)** [Boneh et al., 2018].


When the **block proposer** proposes a new block, all the **validators** in the current **committee** verify the block and broadcast their signatures. All of these signatures are then aggregated into a single **aggregate signature**. Regardless of the number of **validators**, the size of the **aggregate signature** remains only **32 bytes**.

### Implementation

The implementation is mainly in the Consensus component in the **Chameleon** architecture.



**Figure 2.** The layered Chameleon architecture.



The code will be open-source on GitHub with links to specific packages to be provided in the future.

- **Multiview-pBFT:** For the consensus algorithm, **Chameleon** will implement the Multiview-pBFT (Practical Byzantine Fault Tolerance).
- **BLS:** For multi-signature aggregation, **Chameleon** will implement BLS Multi-Signatures.
- **RNG:** For random number generation, **Chameleon** will currently use Bitcoin block hash. Other RNG solutions will be explored in the future.

We are going to propose a new approach to scale privacy on cryptonetworks by applying sharding on privacy transactions to increase throughput for **Chameleon**. **Chameleon's** throughput will scale out linearly with the number of shards. The more shards we add, the more transactions it will be able to handle.

## Multiview: A New Approach for PBFT Implementation

### Problem

Many well-known approaches to implementing the **Proof of Stake** consensus, such as **Tendermint** [1], **Hotstuff** [2], and **PBFT** [3], are **bi-modal**. These protocols typically consist of a simple normal path where a **leader** makes proposals and everyone **votes**. When the normal path fails, the protocol switches to a much more complicated fall-back mode, typically called a “**view change**.” During the **view change** phase, participants use the same communication process to reach an agreement on the new view before returning to the normal path of producing blocks.

**Chameleon Network's BFT (Byzantine Fault Tolerance)** proposes a simpler approach. Instead of reaching an agreement on a single view, participants can observe **multiple views**; the communication process used during **view changes** in existing approaches is now leveraged to propose the new block. Participants may have multiple different views, but when the majority (more than  $\frac{2}{3}$  of participants) commits a new block, it indicates that they have achieved **consensus** on that view. If they continuously produce blocks based on this view, it becomes dominant and achieves **finality**.

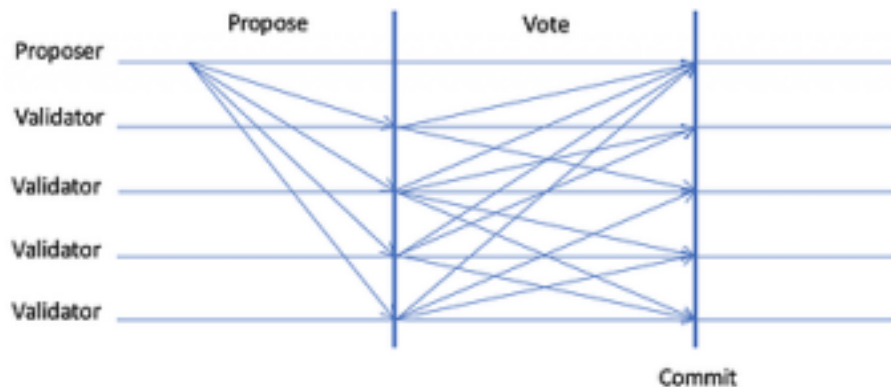
## Preliminaries

- The number of **malicious nodes** in the network cannot simultaneously equal or exceed  $\frac{1}{3}$  of the overall nodes in the system during a given **window of vulnerability**.
- Nodes must be **deterministic** and begin in the same state.
- The final result ensures that all **honest nodes** agree on the correct and longest chain.

## Protocol

**Multiview PBFT** details:

The **Multiview PBFT** protocol used by **Chameleon Network** has the following phases:



**Figure 3. Multiview PBFT.**

### PROPOSE PHASE

- The **Block Proposer** broadcasts the **PROPOSE\_MESSAGE** along with the proposed block to all validators in the committee.

### VOTE PHASE

- Validators broadcast the **VOTE\_MESSAGE** and collect valid **VOTE\_MESSAGE(s)**.
- After a bounded time **T**, if the number of valid votes ( $|VOTE\_MESSAGE(s)|$ ) exceeds  $\frac{2}{3}$  of the committee size, the process moves to the **Commit Phase**.

- If the required threshold is not met, the system waits for a new **Propose Phase** to begin.

## COMMIT PHASE

- Validators combine the valid **VALIDATOR\_SIGNATURE(s)** and include it in the block.
- The block is then committed to the chain.
- After committing, the protocol transitions to a new **Propose Phase**.

In a typical scenario, **Chameleon Network's** PBFT protocol behaves similarly to other PBFT implementations. A node is selected as the proposer and will propose a block; other committee members vote to decide whether the block should be appended to the chain. Proposers are chosen in a round-robin fashion based on their ID in the committee.

If a failure occurs in the normal case (for example, due to:

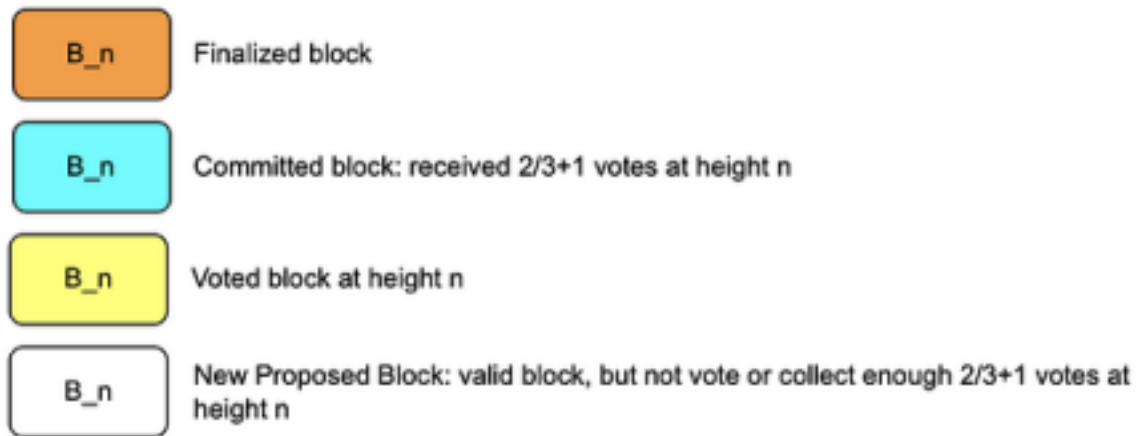
- A Byzantine proposer not proposing a new block or proposing multiple blocks within the same round, or
- A failure to collect enough votes due to delayed vote messages),

the committee members may see multiple different views. To restore consensus, the following general rules are applied:

1. **Do not propose multiple different blocks** at the same block height.
2. **Follow the majority group** to ensure a common view.

## Vote and Propose Rules

To achieve consensus without requiring agreement on a view change, nodes in **Chameleon Network's** committee use the following **Vote Rules** and **Propose Rules**:



Two vote rules for:

1. Branches with the same height
2. Branches with different heights

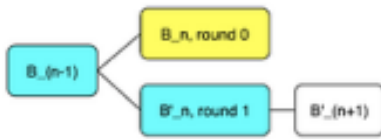


**Vote rule 1 (same height)**

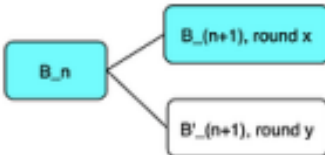
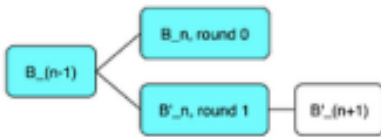
IF receiving multiple B<sub>(n+1)</sub>, vote for B<sub>(n+1)</sub> with the smallest round.

Later on, receiving block B<sub>(n+1)</sub> with round y

- IF  $y > x$ : ignore
- IF  $y \leq x$ : vote it again



**Vote rule 2 (different height)**  
 ONLY vote for the longest branch



**Note:** DO NOT vote for any shorter branch, e.g. ignore block  $B'_{(n+1)}, \text{round } y$ .

Two Propose rules for:

1. Branches with the same height
2. Branches with different height



**Propose rule 1 (same height)**

- IF branches are at the same height, propose block with smallest round

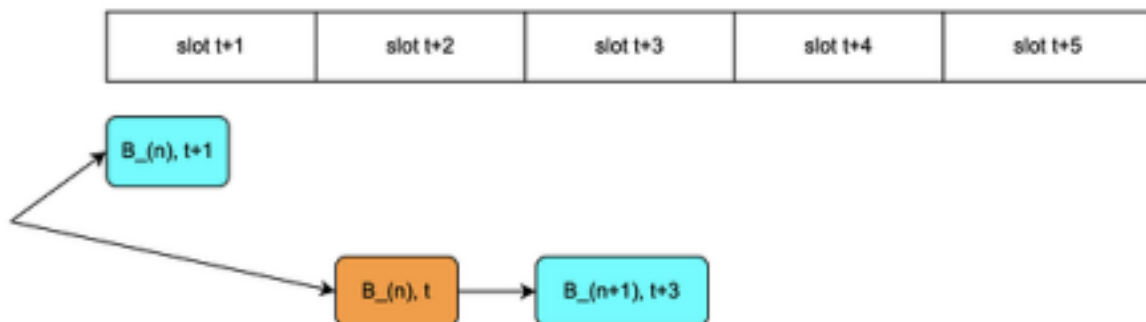


**Propose rule 2 (different height)**

- Propose new block from the longest chain



**Lemma 1.** (Finality 1) If two consecutive blocks  $B_{(n)}$  &  $B_{(n+1)}$  on the same branch are committed in two consecutive time slots, then block  $B_{(n)}$  is finality.



**Figure 4.** Example Finality by Lemma 1

**Proof:**

When block  $n$  is committed at time slot  $t$ , and block  $(n+1)$  is proposed at time slot  $(t+1)$ , this implies that block  $(n+1)$  is proposed for the first time. This also implies that more than  $2/3$  of the committee members have received, agreed upon, and voted for it. Therefore, any further proposed block with height  $(n+1)$  will not receive enough votes to be committed, in accordance with **Vote Rule 1**. Furthermore, following **Vote Rule 2**, no branch can grow longer than the one containing block  $n$ .

**Lemma 2. (Finality 2)**

If two consecutive blocks,  $B_n$  and  $B_{(n+1)}$ , on the same branch are committed at time slots  $t$  and  $(t+i)$ , respectively, where  $B_{(n+1)}$  is first proposed at time slot  $(t+1)$ , and this block is re-proposed at every time slot from  $(t+2)$  to  $(t+i)$ , then **block  $B_n$**  is considered final.



**Figure 5. Example Finality by Lemma 2**

**Proof:**

Since block  $n$  is committed at time  $t$ , and block  $(n+1)$  is committed at  $(t+i)$ , where block  $(n+1)$  is first proposed at time slot  $(t+1)$ , this implies that block  $(n+1)$  with time slot  $(t+1)$  is the latest one. Furthermore, more than  $2/3$  of the committee members have received, agreed upon, and voted for it. This means that any further proposed block  $(n+1)$  will not receive enough votes to be committed, in accordance with **Vote Rule 1**. Additionally, during the time slots  $(t+1)$ ,  $(t+2)$ , ...,  $(t+i)$ , no other blocks except for **block  $(n+1)$**  at time slot  $(t+1)$  are proposed or committed. Due to **Vote Rule 2**, no branch can grow any longer than this one.



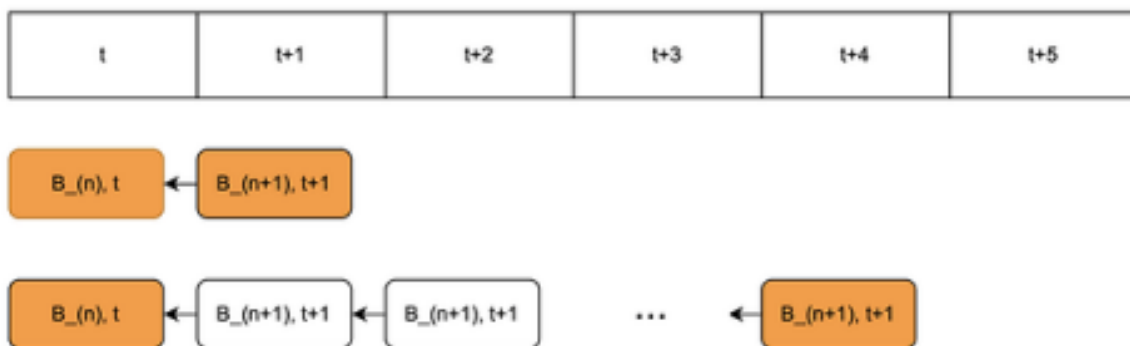
## Finality Theorem

- **IF** a block at height **h** is first proposed and committed in the same time slot, **THEN** it is finalized.
- **IF** a block at height **h** is first proposed in the time slot **t**, it is committed in time slot **(t+n)**, for **n > 1**, and is re-proposed in the consecutive time slots: **t+1, t+2, ..., t+n**, **THEN** it is finalized.

### Proof:

- **(i)** By **Proposing Rules 1 and 2**, the proposed block at height **h** is always part of the longest chain.
- **(ii)** Following **Vote Rule 1**, any other block at height **h** cannot be committed because it will not be able to collect enough votes. In other words, no multiple branches can be created at height **h**.
- **(iii)** Therefore, in order to be committed, any newly proposed block must append to this block. No branch can grow any longer than this one.

Thus, **(i)**, **(ii)**, and **(iii)** imply the **Finality Theorem**.



**Figure 6.** Example finality cases by the finality theorem.

## ANALYSIS

### Observation 1:

If block  $\mathbf{bn}$  is finalized, then further blocks will be appended to the branch containing  $\mathbf{bn}$ , and any other branch  $\mathbf{b'n}$  becomes obsolete. If a new block is successfully appended to another branch, say  $\mathbf{b'n}$ , this implies that more than  $2/3$  of the participants do not agree that  $\mathbf{bn}$  is finalized. This leads to a contradiction.

### Theorem 1 (Consistency Proof):

Let  $\mathbf{ch}$  be the chain:

$\mathbf{ch} := \mathbf{b}_1\mathbf{b}_2\dots\mathbf{b}_i\mathbf{b}_{i+1}$

and  $\mathbf{ch}'$  be the chain:

$\mathbf{ch}' := \mathbf{b}'_1\mathbf{b}'_2\dots\mathbf{b}'_i\mathbf{b}'_{i+1}$

where  $\mathbf{bn}_{i+1}$  and  $\mathbf{b'n}_{i+1}$  are finalized, and if  $\mathbf{bn}_{i+1} = \mathbf{b'n}_{i+1}$ , then  $\mathbf{b}_i = \mathbf{b}'_i$  for all  $i \in [n]$ .

### Proof:

Since  $\mathbf{bn}_{i+1}$  and  $\mathbf{b'n}_{i+1}$  are finalized, it means  $\mathbf{bn}$  and  $\mathbf{b'n}$  are also finalized. If  $\mathbf{bn} \neq \mathbf{b'n}$ , this either violates **Observation 1** or the assumption that more than  $2/3$  of the participants are honest.

### Theorem 2 (Liveness Proof):

If an honest participant receives some transactions, those transactions will eventually be included in all honest participants' finalized chains.

### Proof:

- **Observation 2:** The proposer is selected in a round-robin fashion. Eventually, every participant will become a proposer. The proposer can then include the transaction in the proposed blocks.
- **Observation 3:** If two blocks at the same height are committed, the block proposed earlier must be committed later. Following **Propose Rule 1**, nodes will vote for the block with the smaller round number only. If more than  $2/3$  of the nodes vote for the first proposed block, they will not vote for the later block.

### Worst-Case Scenario:

Consider the worst-case scenario where two chains grow indefinitely. For this to happen, the following conditions must be satisfied:

1. More than  $\frac{2}{3}$  of the participants (the "weak participants") do not collect enough votes to commit any block but may vote for both chains, as shown in **Fig 4**.
2. The remaining  $\frac{1}{3}$  of participants (the "power participants") could propose and commit new blocks.

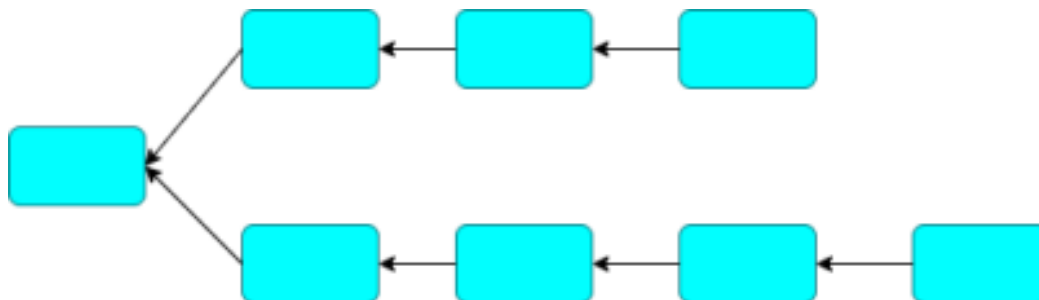
Proposers in the  $\frac{1}{3}$  power participants group are divided into two subgroups, with each subgroup alternately selected to propose blocks on **chain 1** and **chain 2**.

When a participant proposes a block, the next proposer in the round may not receive any messages from other participants, allowing them to propose a block on the other chain.

Let  $N$  be the number of participants. In cases of peak network traffic, assume the probability of successfully transmitting a message between two participants is **0.5**. The probability of a

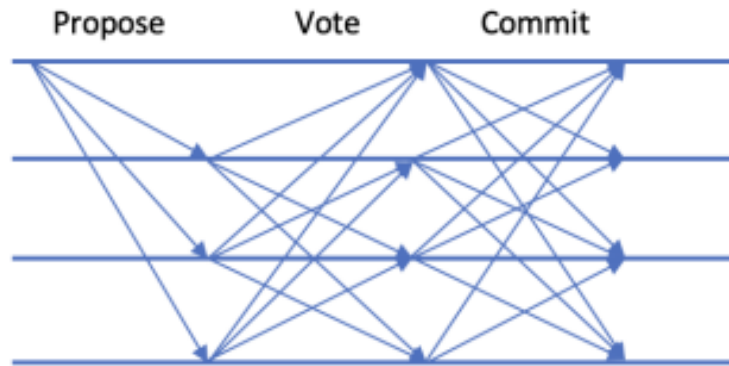
participant not receiving any messages in a single round is then  $(0.5^N)^2$ . This probability becomes negligible as  $N$  grows. Furthermore, the probability of a participant failing to receive any messages over  $x$  rounds is  $(0.5^N)^{(2x)}$ , which decreases exponentially as the number of rounds increases.

Therefore, the above conditions are practically impossible to maintain over multiple rounds, guaranteeing the **liveness** property.

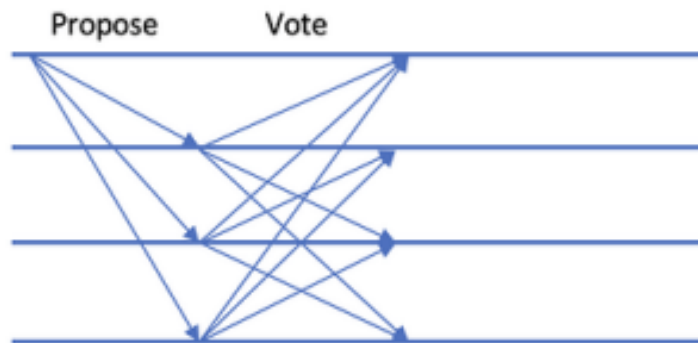


**Figure 7. Fork case.**

Multiview PBFT vs Tendermint [1]



**Figure 8.** Tendermint operation in normal cases.



**Figure 9.** Multiview PBFT operation in normal cases.

**Tendermint Multiview**

Aspect	Tendermint	Multiview
Total messages	$n+2n^2$	$n+n^2$
Number of phases to commit	3	2
Throughput (T being the delay in transferring a message)	$1/(3T)$	$1/(2T)$



## Conclusion

The **Multiview PBFT** approach is simple but offers many advantages over **Tendermint**:

1. **In normal cases, Multiview PBFT** increases throughput by 33% compared to Tendermint, and the total number of exchanged messages is reduced by nearly 50%. Both Tendermint and Multiview PBFT can achieve **instant finality** in a single block.
2. **Network Peaking**: In the Tendermint approach, if more than 1/3 of validator vote messages arrive late, participants will continuously communicate to switch to a new view, and no block can be committed. In contrast, **Multiview PBFT** allows a block to be committed and appended to the chain despite such issues.

The **Multiview PBFT** approach has a natural philosophy: it respects the majority group. The powerful node—capable of committing blocks during bursts of network traffic—can advance the chain to new heights, even during periods of heavy network traffic.

## References

1. [Tendermint Documentation](#)
2. [Paper on Multiview PBFT](#)
3. [MIT's Paper on PBFT](#)

# Chameleon Software Stack: Navigating the Chameleon Source Code

## Chameleon Network Software Stack

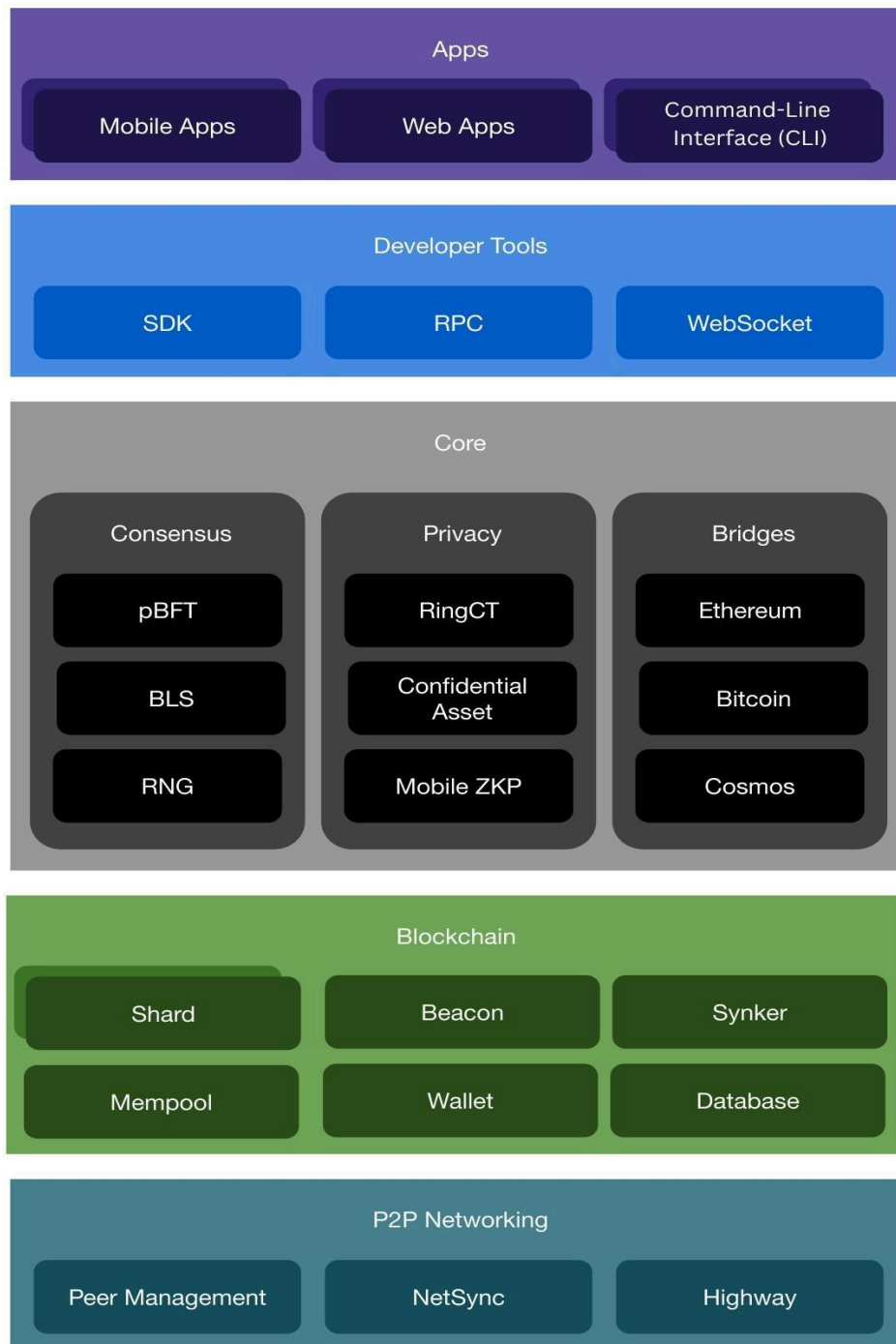
It could be a little overwhelming to read the Chameleon Network source code. There will be more than 1 million lines of code in the Chameleon Network codebase. This topic will provide an overview of the Chameleon Network architecture and a guide to navigating the Chameleon Network source code.

## Network Architecture

The Chameleon Network software stack will be designed in five layers:

1. **P2P Networking layer:** This will be implemented on top of libp2p [Benet and Dias, 2019] to handle peer-to-peer communications such as finding peers, connecting to them, sending and receiving transactions, blocks, and messages. In the future, we plan to introduce changes to further enhance privacy by masking IPs, making the network more secure and private.
2. **Blockchain layer (Data layer):** This will implement data storage for shards and the beacon, and ensure data synchronization among nodes.
3. **Core layer:** This will implement the consensus mechanism, privacy, and bridges to other cryptonetworks.
4. **Developer Tools layer:** This will provide a few different options for developers to work with Chameleon Network, including an SDK, RPC, and Websocket.
5. **Application layer:** This will ship a reference mobile wallet, some reference privacy apps, and a reference hardware full node.

Additionally, several infrastructure tools will be built to support network monitoring, visualization, and deployment.



**Figure 1.** The layered Chameleon architecture.

## Navigating the Chameleon Network Source Code

Chameleon Network will be implemented in Go for a balance of portability, performance, and the development efficiency it will bring to a large-scale, open-source project like Chameleon Network.

### [P2P Networking](#)

- **Peer Management:** Peer management will handle peer-to-peer communications such as finding peers, connecting to them, sending and receiving transactions, blocks, and messages.
- **NetSync:** NetSync will be a mediator that receives incoming messages, parses them, and routes the messages to the appropriate components.
- **Highway:** Highway will be a new network topology design that speeds up P2P communications. It will be under development and merged into the main codebase in the future.

### [Blockchain](#)

- **Shards:** Shards will be subchains. A subchain will be a Proof-of-Stake blockchain with its own committee of N nodes. A shard's job will be to produce new blocks via a Practical Byzantine Fault Tolerance (pBFT) consensus algorithm.
- **Beacon:** Beacon will also be a subchain. A beacon's job will be to coordinate the shards and maintain the global state of the network.
- **Synker:** Synker will ensure that the node is up to date among its peers and will also broadcast the node's status to its peers.
- **Mempool:** Mempool (memory pool) will be a collection of transactions and blocks that have been verified but are not yet confirmed.
- **Wallet:** The wallet will hold all your Chameleon Network keys. Users will be able to use it to send and receive Chameleon Network tokens.
- **Database:** Chameleon Network will use LevelDB to store block data.

### [Core](#)

#### [Consensus](#)

- **pBFT:** Chameleon Network will implement pBFT (Practical Byzantine Fault Tolerance) as the consensus algorithm.
- **BLS:** For multi-signature aggregation, Chameleon Network will implement BLS Multi-Signatures.
- **RNG:** For random number generation, Chameleon Network will use Bitcoin block hash initially, but will explore other RNG solutions in the future.



## Privacy

- **RingCT:** Chameleon Network will implement RingCT (Ring Confidential Transaction) with ring signatures, stealth addresses, and confidential transactions for privacy.
- **Confidential Asset:** RingCT will hide the amount of the transaction, but it won't hide the type of asset being sent. Confidential Asset will solve that.
- **Mobile ZKP:** Chameleon Network will implement Zero-Knowledge Proofs (ZKP) Generation on mobile. Private transactions will be able to be sent on any regular phone in under 15 seconds.

## Bridges

- **Ethereum:** Chameleon Network will implement a trustless two-way bridge between Chameleon Network and Ethereum, allowing users to send and receive ETH & ERC20 tokens privately.
- **Bitcoin:** Chameleon Network will work on a trustless two-way bridge between Chameleon Network and Bitcoin, allowing users to send and receive BTC privately.
- **Cosmos:** Chameleon Network will explore Cosmos and hopes to build a trustless two-way bridge between Chameleon Network and Cosmos.

## Developer Tools

- **RPC:** RPC will allow developers to interact with Chameleon Network via their own programs.
- **WebSocket:** WebSocket will provide another way for developers to interact with Chameleon Network via their own programs.
- **SDK:** Chameleon Network will be working on Developer SDKs to make it easier to build on top of Chameleon Network.

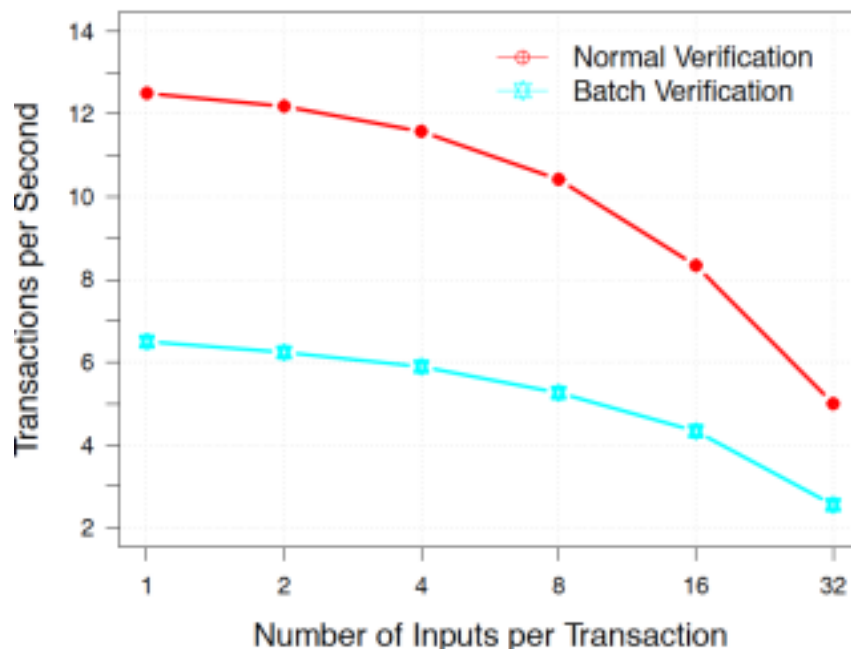
## Apps

- **Mobile Apps:** Developers will be able to easily build mobile apps on top of Chameleon Network once the SDK is available. An example could be a mobile wallet.
- **Web Apps:** Developers will be able to build web apps on top of Chameleon Network once the SDK is available. Examples could include a web wallet or a desktop network monitor.
- **Hardware Devices:** Developers will also be able to build hardware on top of Chameleon Network once the SDK is available. An example could be a node device.
- **CHML-CLI:** A command-line interface (CLI) tool will be available for interacting with Chameleon Network. This will allow developers to interact with the network and manage nodes from the terminal.

## Chameleon Performance

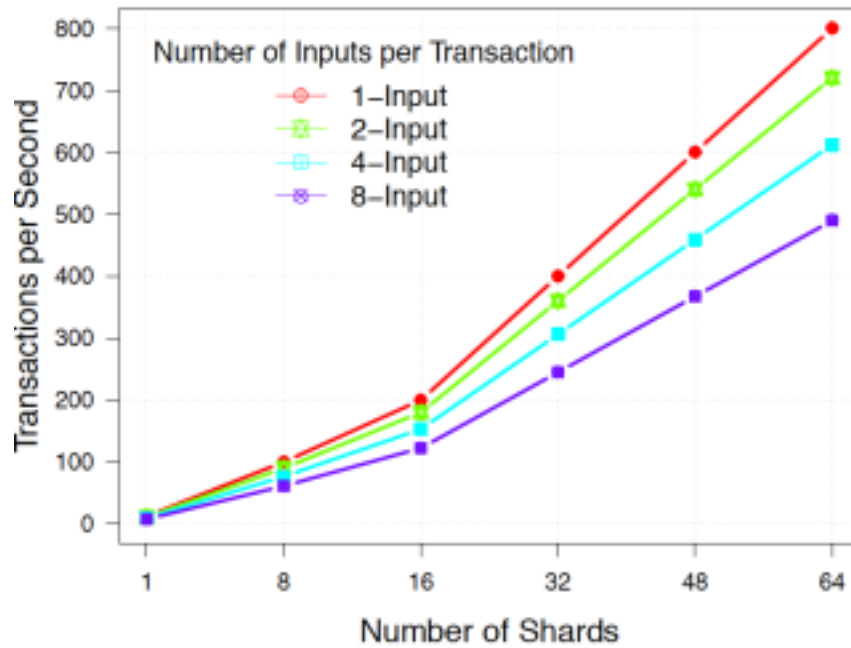
We will evaluate the performance of Chameleon Network based on its real workload on the mainnet, as well as on simulated workloads.

In evaluating the performance of the Chameleon Network, we will focus on the privacy transaction throughput, more specifically the metric of transactions per second (TPS). Note that Chameleon Network will use a UTXO-based ledger. In most cases, fewer than 32 existing UTXOs will be used as inputs, and 2 new UTXOs will be produced as outputs.

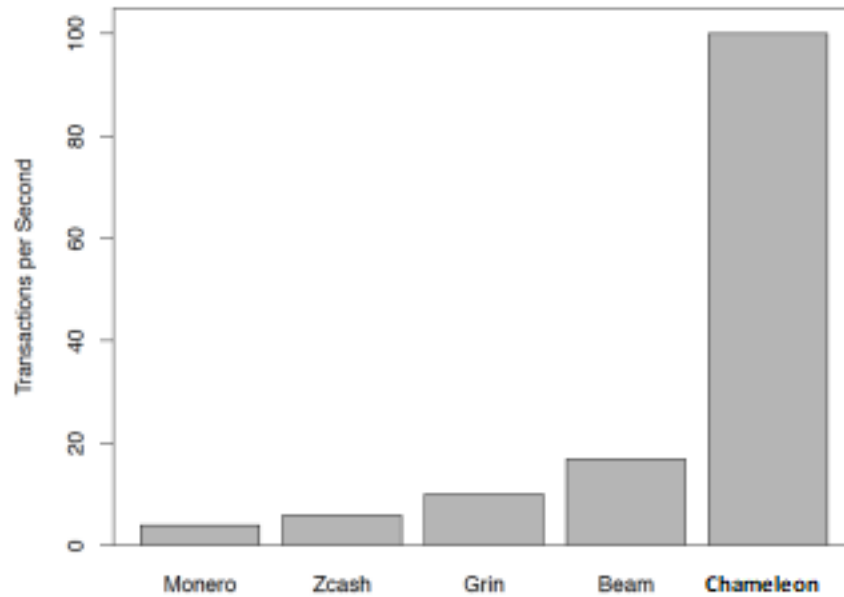


**Figure 2.** Transactions per second based on the number of inputs per transaction on a shard .


In Figure 2, we will illustrate the TPS metric for one shard and two fixed outputs. In the best case, the network will achieve 6.5 TPS with only one input, and in the worst case, approximately 2.5 TPS with 32 inputs. In the future, we will introduce a batch verification feature to verify transactions in a batch, which will significantly enhance transaction throughput. This feature is expected to improve transaction throughput by twofold. Specifically, Chameleon Network will achieve 12.5 TPS and 5 TPS for one input and 32 inputs, respectively.



**Figure 3.** Transactions per second based on the number of shards .



**Figure 4.** TPS comparison among Chameleon, Monero, Zcash, Grin, and Beam .



Transaction throughput will scale linearly with the number of shards. Initially, with 8 shards active, Chameleon Network will handle 90–100 TPS in the most common case (two inputs and two outputs per transaction). We are developing a new network topology to scale the network to 64 shards. With this enhancement, Chameleon Network will achieve 800 TPS, which will be significantly higher than other privacy blockchains. For example, Monero achieves approximately 4 TPS, Zcash 6 TPS, Grin 10 TPS, and Beam 17 TPS. Details will be shown in Figures 2 and 3.

## Network Incentive: CHML Mining & Distribution

**CHML** is the native token of the Chameleon Network, with a fixed supply of 100 million tokens. It will play a pivotal role in the ecosystem by enabling staking, governance, transaction fee payments, and liquidity mining.

### Key Features of CHML

1. **Fixed Supply:** Total supply will be capped at 100,000,000 CHML, ensuring scarcity and value preservation.
2. **Multiple Use Cases:**
  - **Staking:** Validators (V-Nodes/P-Nodes) will stake CHML to secure the network and earn rewards.
  - **Transaction Fees:** CHML will be used to pay network fees for transactions and shielding/unshielding.
  - **Governance:** Token holders will participate in decision-making by voting on network proposals.
  - **Liquidity Mining:** CHML will incentivize liquidity providers on the privacy decentralized exchange (pDEX).

### Token Allocation

The total supply of 100 million CHML tokens will be allocated as follows:

#### 1. Community Airdrop (5%)

**Allocation:** 5,000,000 CHML

- **4% (4,000,000 CHML):** Allocated to Privacy token (PRV) holders from a previous privacy blockchain project. A 1:1 swap mechanism will allow eligible PRV holders to claim CHML via a vesting schedule.
- **1% (1,000,000 CHML):** Reserved for newly acquired early community supporters who actively contribute to the ecosystem.



### **Airdrop Vesting Schedule:**

- 15% unlock at the conclusion of presale rounds.
- 25% unlock at testnet launch.
- 60% unlock through linear vesting over six months post-mainnet launch.

### **2. Presale (20%)**

**Allocation:** 20,000,000 CHML

**Purpose:** Raise funds to support development, marketing, operations, and infrastructure for the Chameleon Network.

- **Round 1:** 12,000,000 CHML at \$0.45 per token.
- **Round 2:** 8,000,000 CHML at \$0.60 per token.

### **Vesting Schedule:**

- 15% tokens unlock at creation of an interim pool at Uniswap.
- 25% token unlock at testnet launch.
- 60% tokens unlock in linear vesting over 6 months post-mainnet.

### **3. Reward Pool (65%)**

**Allocation:** 65,000,000 CHML

**Purpose:** Incentivize participation through staking rewards and liquidity mining.

### **Breakdown:**

- **Validator Rewards:** 45,500,000 CHML (70% of the Reward Pool).
- **Liquidity Provider Rewards:** 19,500,000 CHML (30% of the Reward Pool).

**Emission Schedule:** Gradual rewards decline over 20 years, with higher emissions in the early years to encourage participation.



#### 4. Initial Liquidity Pool (2%)

**Allocation:** 2,000,000 CHML

- **2,000,000 CHML** paired with **2,000,000 USDT** (raised during the presale) to create a liquidity pool on **Uniswap**.

**Purpose:**

- **Enable early trading** for CHML tokens to foster confidence and flexibility during development.
- **Allow new investors** to join the ecosystem at the initial listing price.

**Breakdown:**

- Liquidity paired with **USDT raised during the presale** to establish a trading pool on **Uniswap**.
- Liquidity will remain **locked until the mainnet launch** to ensure stability and prevent manipulation.
- Post-mainnet, **all liquidity will migrate to the Chameleon pDEX**, enabling **private trading** and **staking opportunities** exclusive to the pDEX.

#### 5. Staking Infrastructure (5%)

**Allocation:** 5,000,000 CHML

- Reserved to support the **fixed/highway node infrastructure** during Chameleon's early stages.
- Tokens will unlock progressively based on network decentralization milestones and community governance approval.

**Purpose:**

- Provide operational support for the **Chameleon network's fixed nodes** during its initial phases.
- Ensure a smooth transition to **community-managed Validator Nodes (V-Nodes)** and **P-Nodes** as the network moves toward **100% decentralization**.



### Vesting Schedule:

- Tokens will remain **locked** until the network reduces reliance on fixed nodes and demonstrates significant progress toward decentralization.
- The core team will control the allocation and propose token usage based on **project milestones** and the community's approval.

### 6. Ecosystem Development (3%)

**Allocation:** 3,000,000 CHML

- Supports **community-driven development** and incentivizes **open-source collaboration** through a milestone-based reward system governed by **DAO proposals**.

#### Purpose:

- **Community Developer Recruitment:**
  - Tokens allocated as rewards for **community developers** contributing to Chameleon's technical ecosystem.
  - Rewards are **milestone-based**, ensuring compensation for achieving predefined technical deliverables.
  - Decisions on reward allocations will be **governed by DAO voting** for transparency and fairness.

#### Speeding Up Core Development:

- Engages talented developers worldwide to accelerate **feature releases** and ensure high-quality technical contributions.

#### Supporting Open-Source Collaboration:

- Encourages developers to build **tools, libraries, and applications** that integrate with or enhance the Chameleon ecosystem.

### 7. Reward Emission Schedule

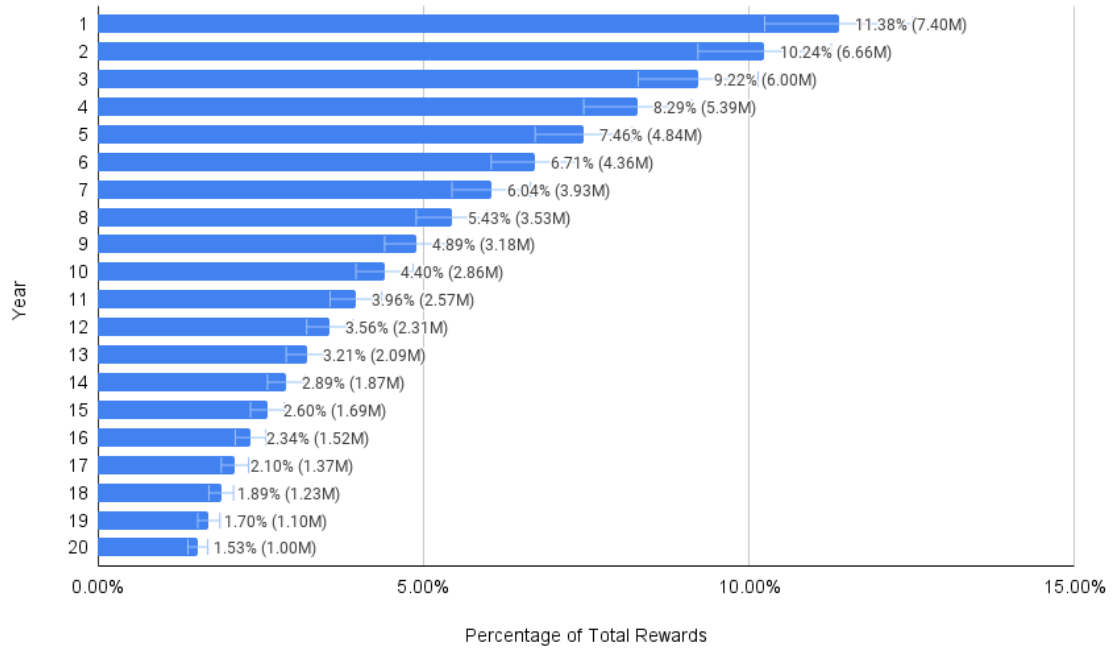
**Total Reward Pool:** 65,000,000 CHML

Total Reward Pool (65M) emission schedule



<b>Year</b>	<b>Percentage of Total Rewards</b>	<b>CHML Distributed</b>
1	11.38%	7.40M
2	10.24%	6.66M
3	9.22%	6.00M
4	8.29%	5.39M
5	7.46%	4.84M
6	6.71%	4.36M
7	6.04%	3.93M
8	5.43%	3.53M
9	4.89%	3.18M
10	4.40%	2.86M
11	3.96%	2.57M
12	3.56%	2.31M
13	3.21%	2.09M
14	2.89%	1.87M
15	2.60%	1.69M
16	2.34%	1.52M
17	2.10%	1.37M
18	1.89%	1.23M
19	1.70%	1.10M
20	1.53%	1.00M
<b>Total</b>	<b>100%</b>	<b>65M CHML</b>

### Reward Pool (65M CHML Allocation)



The Reward Pool of 65M is further split into

- **Validator Rewards:** 45.5M CHML (70% of Rewards Pool).
- **Liquidity Provider Rewards:** 19.5M CHML (30% of Rewards Pool)

**Validator Rewards** 45.5M CHML (70% of Rewards Pool).

**Total Allocation:** 45.5M CHML

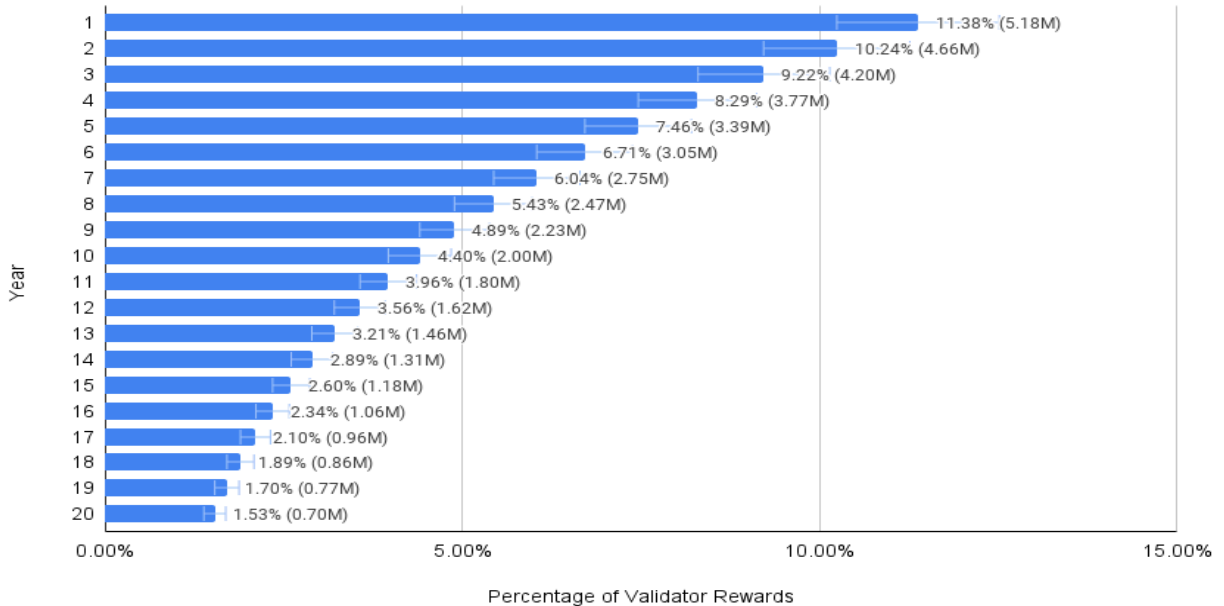
- Emission is structured to gradually decrease over 20 years, incentivizing long-term participation while ensuring sustainability.

### Validator Rewards Emission Schedule

Year	Percentage of Validator Rewards	CHML Distributed
1	11.38%	5.18M
2	10.24%	4.66M
3	9.22%	4.20M
4	8.29%	3.77M
5	7.46%	3.39M

6	6.71%	3.05M
7	6.04%	2.75M
8	5.43%	2.47M
9	4.89%	2.23M
10	4.40%	2.00M
11	3.96%	1.80M
12	3.56%	1.62M
13	3.21%	1.46M
14	2.89%	1.31M
15	2.60%	1.18M
16	2.34%	1.06M
17	2.10%	0.96M
18	1.89%	0.86M
19	1.70%	0.77M
20	1.53%	0.70M
Total	100%	45.5M CHML

### Validator Rewards (45.5M CHML Allocation)



Liquidity Provider Rewards 19.5M CHML (30% of Rewards Pool)

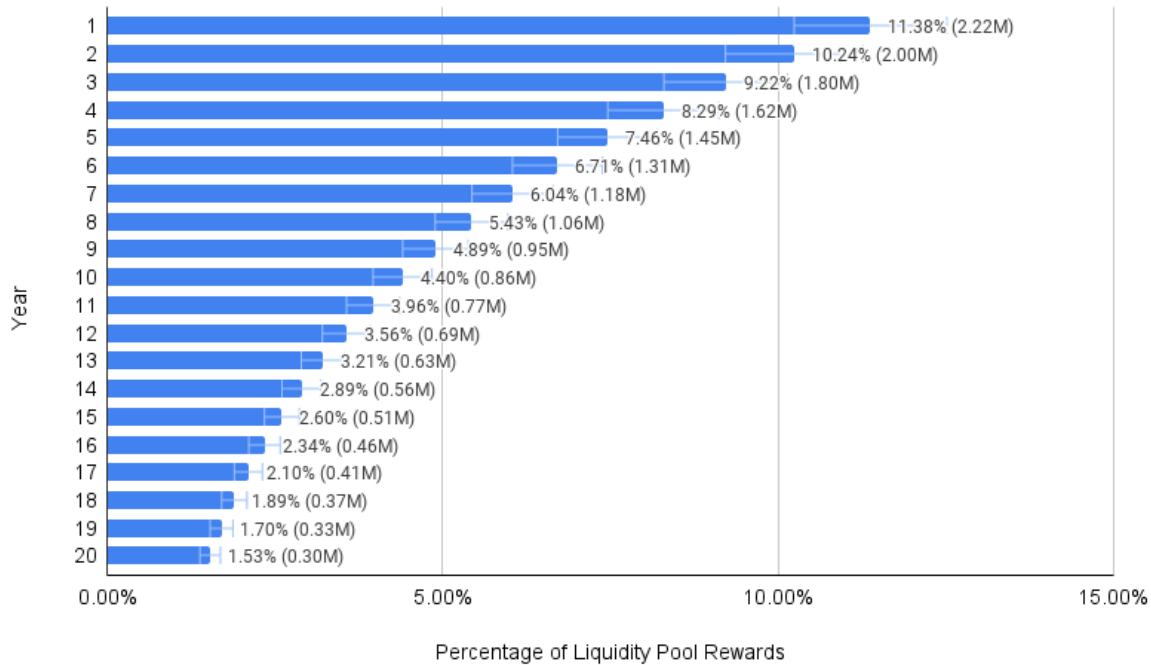
Total Allocation: 19.5M CHML

- Emission follows a similar declining pattern to Validator Rewards over 20 years.

Liquidity Pool Rewards Emission schedule

Year	Percentage of Liquidity Pool Rewards	CHML Distributed
1	11.38%	2.22M
2	10.24%	2.00M
3	9.22%	1.80M
4	8.29%	1.62M
5	7.46%	1.45M
6	6.71%	1.31M
7	6.04%	1.18M
8	5.43%	1.06M
9	4.89%	0.95M
10	4.40%	0.86M
11	3.96%	0.77M
12	3.56%	0.69M
13	3.21%	0.63M
14	2.89%	0.56M
15	2.60%	0.51M
16	2.34%	0.46M
17	2.10%	0.41M
18	1.89%	0.37M
19	1.70%	0.33M
20	1.53%	0.30M
Total	100%	19.5M CHML

## Liquidity Provider Rewards (19.5M CHML Allocation)



## Token System

There are 3 types of tokens:

- **CHML.** will be Chameleon Network’s native coin — a work token. Users will stake CHML to become validators. Validators will earn block rewards in CHML and transaction fees in various cryptoassets (i.e. pBTC, pETH, etc.). This model ensures that only those truly invested in the growth of the network participate, avoiding speculative activity. As demand for private transactions grows, validators will earn more revenue, leading to an increase in the value of CHML.
- **Bridged Privacy Coins.** Users will be able to convert cryptocurrencies (or “public coins”) on other blockchains (i.e. BTC, ETH, USDT) to privacy coins on Chameleon Network (i.e. pBTC, pETH, pDAI). These privacy coins will maintain a 1:1 peg and be fully confidential. As a result, users can store, send, and receive any cryptoassets with complete privacy. Private coins will also be usable for transaction fees.
- **Issue-Created Privacy Coins.** Users will have the ability to issue their own privacy coins on Chameleon Network, providing further flexibility and privacy options within the ecosystem.

## Transaction Fees

Users will be able to pay transaction fees in their cryptocurrency of choice (CHML, pBTC, pETH, pDAI, etc.).

## User-Created Privacy Coins

**Chameleon Network** will offer users and developers an easy way to create their own privacy coins. At the time of writing, numerous user-created privacy coins will have been issued within the Chameleon ecosystem.

We believe that in the near future, tokens will increasingly represent everyday assets, including but not limited to stocks, fiat currencies, gold, real estate, and other forms of ownership. We also strongly believe that very few people will willingly disclose their token holdings to the entire world.

### PRIVACY COINS

Description	# of TXs
Hedus (HEDUS)	Rewards for student behavior
Tipcoin (TIP)	A privacy coin for tipping on social media
PERKS (PERKS)	Tradable coupons redeemable for fiat by merchants

**Table 1.** *Some privacy coins created by the community.*

---

## Use Cases: Privacy Stablecoins, Privacy DEX, Confidential Crypto Payroll, and more

### Privacy Stablecoins as P2P Digital Cash

The lack of privacy remains a significant obstacle for stablecoins to function effectively as digital cash. With current blockchain systems like Ethereum, it is too easy to analyze the exact amount of money individuals and businesses hold, as well as how they spend it, based on the public ledger. Chameleon Network enables the creation of privacy stablecoins, such as pDAI or pUSDT. These privacy stablecoins will offer both stability and privacy, much like physical cash. They will be ideal for cross-border business payments and offer a secure, familiar way for users to store personal savings.

### Anonymous Cross-Chain Decentralized Exchanges

Existing decentralized exchanges (DEX) are pseudonymous, meaning they provide limited privacy. Chameleon Network's **pDEX** (privacy decentralized exchange) will be a new type of exchange that is not only decentralized but also fully privacy-protecting.

Thanks to Chameleon's interoperability with various cryptonetworks, the pDEX will facilitate anonymous cross-chain trading—for instance, trading pBTC for pDAI. In the first two months since its launch, Chameleon's **pDEX** is expected to facilitate nearly 16,000 anonymous trades across 150 pairs.

### Buy & Sell Crypto Anonymously

Chameleon Network will provide a way for users to buy and sell crypto **anonymously**. On platforms like LocalBitcoins or through Bitcoin ATMs, users will be able to perform transactions with privacy. The seller will simply need to make a deposit to the buyer's Chameleon wallet address. Although the seller will not need a **Chameleon wallet**, the buyer will enjoy full privacy. Similarly, sellers will be able to sell crypto anonymously, making transfers from their Chameleon wallet to the buyer's address, who may or may not need a Chameleon wallet if they don't mind exposure.



## Pay Anonymously Online

Many distributed teams and companies are turning to crypto payroll to cut down on international transfer fees, save time, and, in some cases, pay employees in a more desirable currency.

However, the major downside of traditional crypto payroll is that salary and payment details are publicly visible to anyone who examines the blockchain.

With Chameleon's **anonymous** payroll system, users will be able to make payments in privacy-preserving stablecoins like privacy USDT, privacy BTC, or another chosen privacy coin.

This system will eliminate the issue of public exposure while still retaining all the benefits of digital payments.



# Highway: An Upgrade to Chameleon's Network Topology

## Chameleon highway

Chameleon Network, will use a **pBFT-based Proof-of-Stake consensus protocol** combined with **state sharding** to maintain high transactions per second (TPS) while preserving the security of the network. However, during testing, two key challenges were identified that could hinder future scalability:

1. **Physical Node Connectivity:** A significant portion of Chameleon validators are anticipated to run on **Chameleon Physical Node devices**, typically set up in home environments behind NAT (Network Address Translation) or firewalls. This prevents these devices from connecting to each other, especially when all validators in a shard are on such devices, which would leave them unable to communicate effectively.
2. **Excessive Connection Management:** Validators' clients need to manage too many connections to other validators within the same shard, across other shards, and to the beacon chain. Maintaining this level of data availability and preventing message-filtering-attacks require broadcasting messages to all nodes, consuming significant bandwidth and processing time.

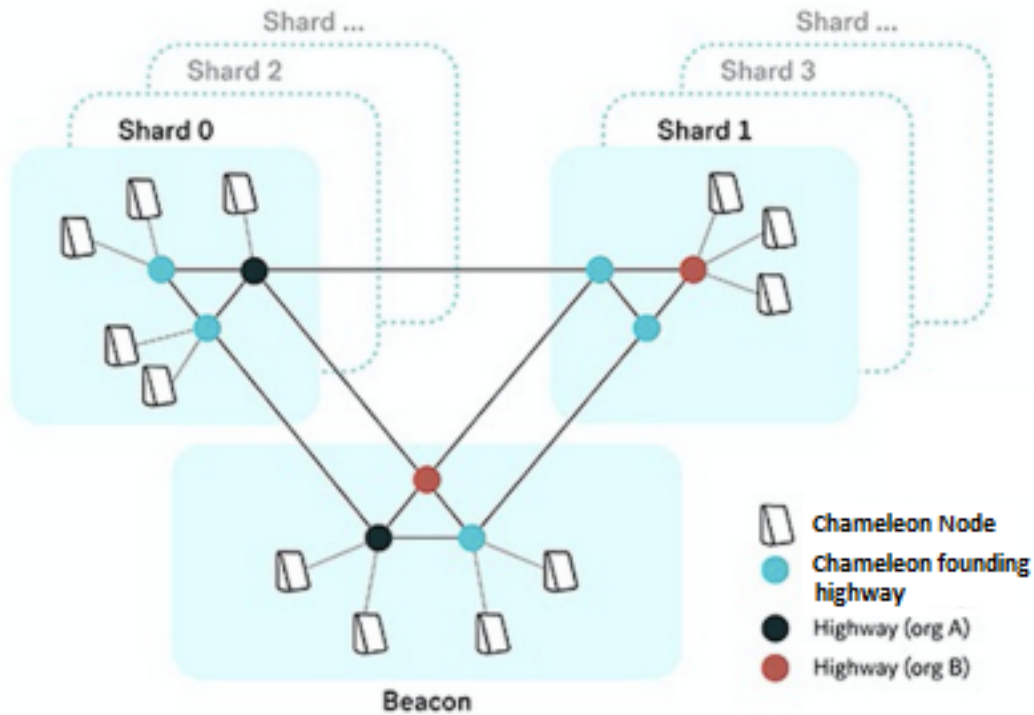
To resolve these problems, Chameleon Network introduces the **Highway** system—highly dedicated nodes that function as proxies to forward messages efficiently across the network.

## Highway Design Principles

The **Highway** design embraces four key principles:

1. **Highly Available:** Highways are always available to serve node requests, prioritizing availability over consistency in the event of network partitioning.
2. **Incremental Scalability:** Highways can be scaled one by one based on the network's needs, ensuring flexibility and efficient resource management.
3. **Heterogeneity:** Each highway's workload is distributed based on its capabilities, ensuring that more powerful highways handle heavier loads.
4. **Symmetry:** Every highway performs the same functions, creating an equal and balanced system for easier provisioning and maintenance in the long term.

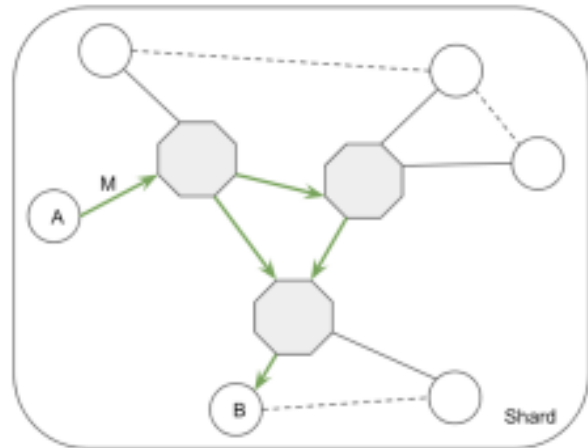
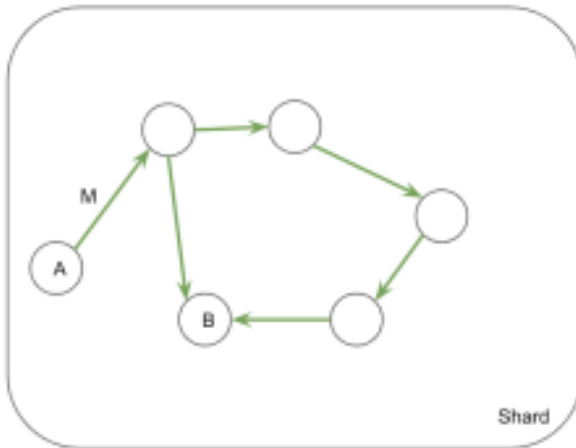
## New network topology



The figure above gives a simplified view of Chameleon's network topology. Highways manage each shard and beacon. Instead of connecting directly, each node connects to a highway for the shard it needs. Highways are interconnected (using mesh/ring or fully-connected topology) to ensure message flow. Each highway must have a static public IP to allow node connections.

When a node broadcasts data, it sends it to the highway. The highway then forwards the data to other highways and connected peers. This reduces redundant data and cuts down on processing. Latency is also minimized, as messages pass through at most 3 hops.

This setup improves network efficiency and performance.



## API

Chameleon Network's highway code will be open-source and serve as a reference implementation. The goal is for developers to contribute or implement their own versions in their preferred programming language.

A highway node provides three main functions for the Chameleon client:

1. **Broadcast and Listen for New Blocks:** Using a **Publish/Subscribe model** to share new blocks as they are generated.
2. **Request and Provide Old Blocks:** Using the **gRPC framework** for requesting and providing historical data.
3. **Broadcast and Listen to the Current State of the Network:** To keep the network updated and in sync.

To put it more concretely, a highway satisfies this interface:

```
type Highway interface {
    BroadcastBlock(blockHeight int, blockData []byte, shardID int) error
    ListenToBlock(shardID int) (channel []byte, error)
    RequestBlock(blockHeight int, shardID int) ([]byte, error)
    PublishState(pubkey []byte, state []byte, shardID int) error
}
```

## Scalability of the Highway System

Chameleon aims to maintain a network with **64 shards**, each supported by **256 validators**. For the purpose of this calculation, we assume a blocktime of **40 seconds/block** and a maximum **block size of 2MB**.

The system needs to handle three types of messages:

1. **pBFT messages**
2. **Block broadcasting messages**
3. **Cross-shard messages**

### Bandwidth Calculation

1. **pBFT Messages:** The required bandwidth for each highway to handle pBFT messages is calculated based on the number of validators **N** and the number of validators supported by each highway **K**. The formula for this type is **12.8×KMB/s**.
2. **Block Broadcasting:** Assuming each validator has 10 substitutes, the bandwidth required for block broadcasting is **0.5×KMB/s**.
3. **Cross-shard Messages:** Each highway receives blocks from 63 other shards (and the beacon), which contributes a bandwidth requirement of **3.2×KMB/s**.


Thus, for a highway with **K=32K**, the total bandwidth requirement is approximately **528 MB/s** to maintain the network at full load.

### Memory Requirements

To ensure low latency, highways must store frequently accessed blocks in memory. The memory required to store **M epochs of blocks** is **M×700MBMB**. Since validators are generally up-to-date with the latest blocks, the memory requirement is manageable.

### Design Choices

1. **Availability:** Highways must be crash-tolerant, which is achieved through **replication**. Each shard is supported by at least two highways, preferably located in different data centers. In the event of a network partition, highways act independently to serve client requests.

- 
2. **Scalability and Stability: Consistent hashing** is used to assign highways to nodes, ensuring that highways can be added or removed without disrupting the network. This method also ensures the topology remains stable, even when nodes frequently go offline and come online.
  3. **Functionality:**
    - For **block publishing** and **subscribing**, Chameleon uses the modular **libp2p** network stack for compatibility with older versions and other blockchains.
    - For **requesting old blocks**, instead of broadcasting requests, highways directly contact a suitable peer using **gRPC** for high-performance communication.
    - The **network membership** is maintained via a **publish/subscribe pattern** to gossip about highway additions and removals.

## Challenges and Ongoing Research

1. **Security:** All messages are cryptographically signed by validators. Highways are mere messengers and do not interfere with the consensus process.
2. **Availability of Highways:** If a significant number of highways go down, block generation could slow until enough highways are available to cover more than  $\frac{2}{3}$  of validators.
3. **Centralization:** Currently, highways are run by the **Core Development Team**, but the goal is to enable others to run and connect highways in the future.

# Privacy Mode for dApps on Ethereum

## pEthereum Specifications

Building privacy-protecting decentralized applications, smart contracts, and crypto-assets.

### What is pEthereum?

The Ethereum smart contract platform offers an entirely new programming paradigm. It enables developers worldwide to collaboratively build a new type of global financial infrastructure without relying on central authorities. However, privacy concerns often discourage broader adoption beyond the crypto niche. Everyday users may hesitate to disclose how much DAI they are saving, the profitability of their **Uniswap** trades, or the frequency of their borrowing on Compound.

What is needed is **Privacy Mode** for Ethereum smart contracts.

**pEthereum** is an extension of Ethereum that enables privacy-protecting Ethereum transactions and decentralized applications like Uniswap and Compound. Transactions are encrypted using zero-knowledge proofs, allowing users to maintain their privacy.

With pEthereum, developers can program smart contracts that are not just decentralized but also privacy-protecting.

### Core Concepts

- **Cross-Chain Instruction:** Chameleon communicates with Ethereum via instructions. These instructions are high-level, cross-chain opcodes specifying operations to be performed by the other chain. There are five instructions: **SHIELD**, **UNSHIELD**, **DEPLOY**, **UNDEPLOY**, and **EXECUTE**.
- **Bridge:** The bridge is a two-way trustless bridge between Chameleon and Ethereum, responsible for forwarding instructions between the two chains. It consists of multiple relayers. Relayers cannot forge or corrupt instruction content because each instruction is cryptographically signed by users and verified on both ends of the bridge.
- **Broker:** The broker is a smart contract on Ethereum that receives instructions from Chameleon, verifies them, and redirects them to appropriate dApps on Ethereum.
- **dApp:** A decentralized application (or "dApp") lives on Ethereum. It consists of one or more smart contracts that execute exactly as programmed.

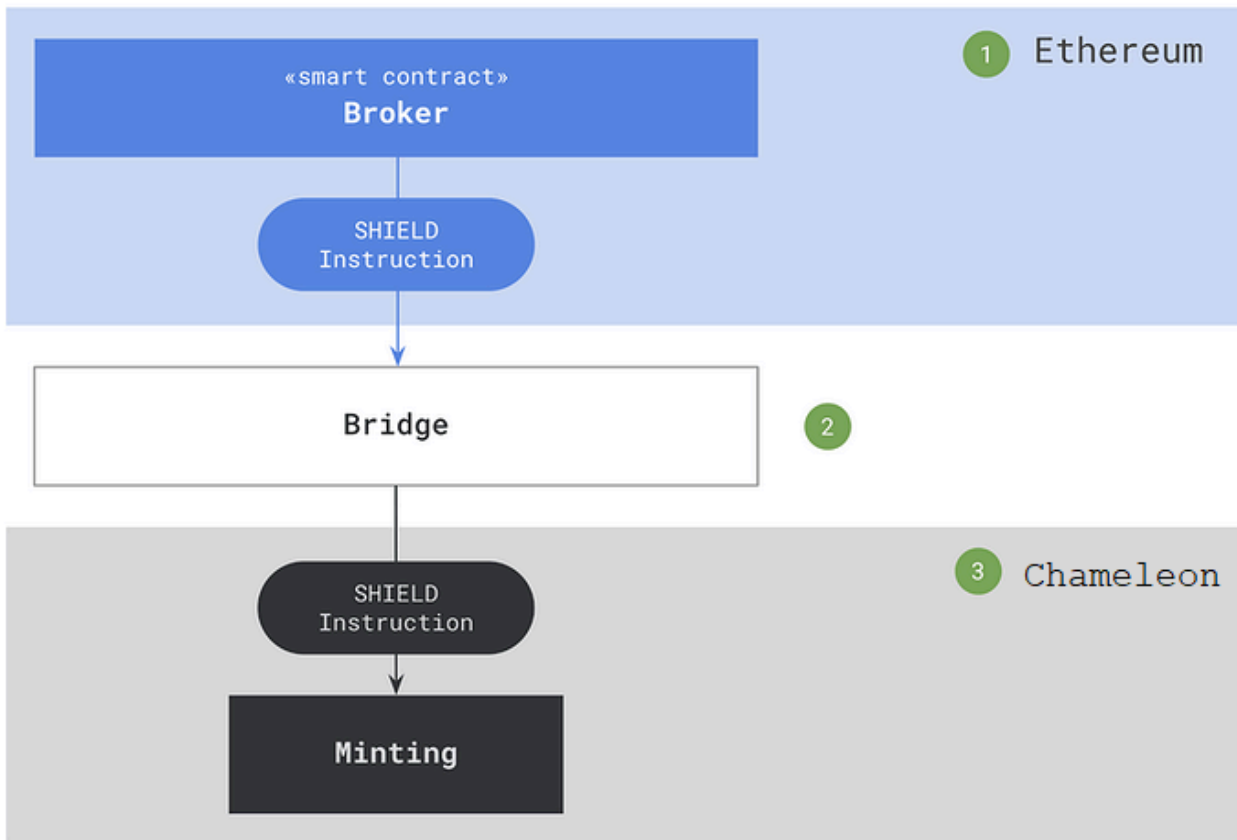
- **pApp**: A privacy-protecting decentralized application (or "pApp") lives on Chameleon. It is the privacy-protecting counterpart of an existing dApp on Ethereum.

## Cross-Chain Instructions

### SHIELD instruction

Shielding is the process of converting a public ERC20 token into its privacy counterpart of the same value. For example, DAI can be shielded to obtain the privacy coin **pDAI**. pDAI holds the same value as DAI, ensuring that 1 pDAI can always be redeemed for 1 DAI and vice versa. Once shielded, privacy coin transactions are confidential and untraceable.

Following is an overview of the **SHIELD** instruction flow:

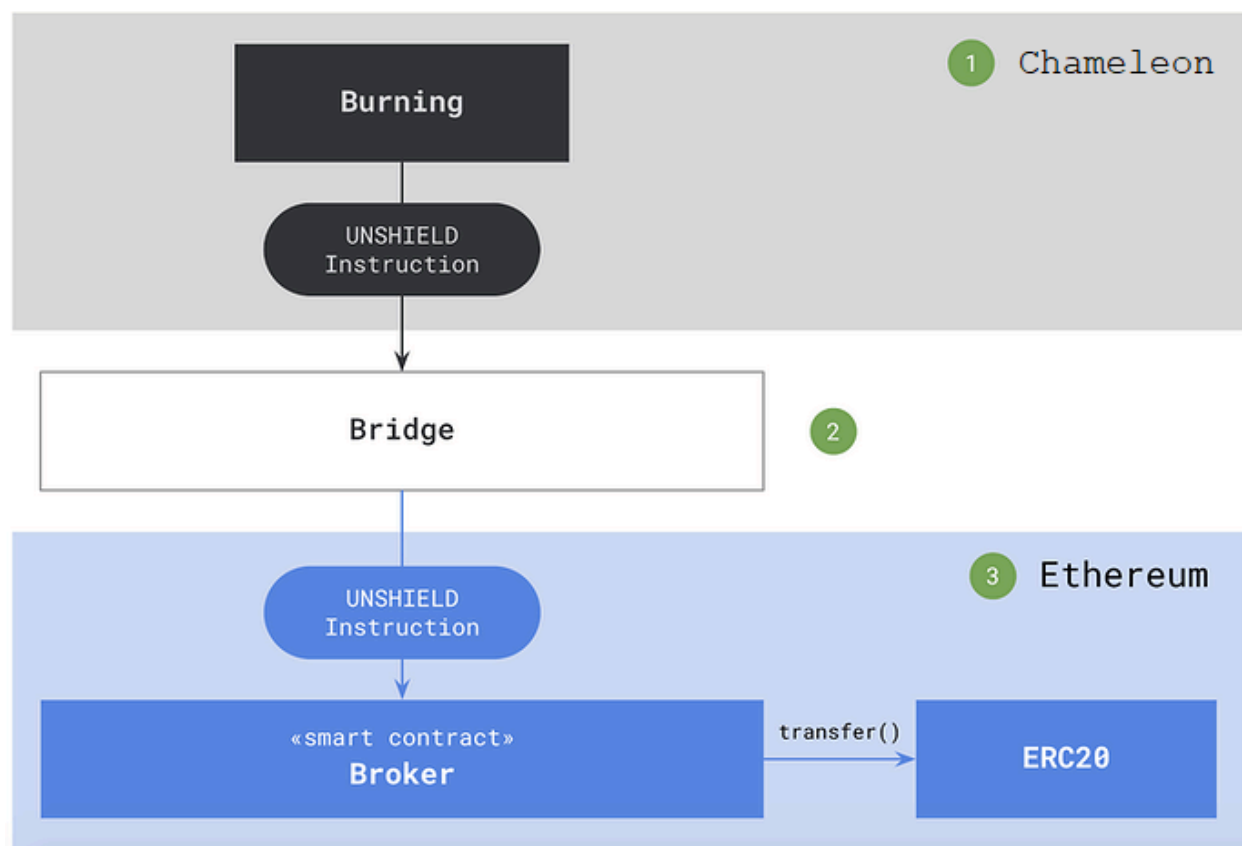


1. A user deposits some amount of an ERC20 token into the Broker smart contract. Once the transaction is confirmed on Ethereum, the user obtains a deposit proof.
2. The user sends a SHIELD instruction to Chameleon, along with the deposit proof, via the bridge.
3. Chameleon validators parse the SHIELD instruction to retrieve the deposit proof, which is verified using Ethereum Simplified Payment Verification (SPV). They also extract the minting parameters, which are then used to mint the privacy coin counterpart of the same value.


## UNSHIELD Instruction

Unshielding is the reverse process of shielding: converting privacy coins back into public ERC20 tokens.

Following is an overview of the UNSHIELD instruction flow:





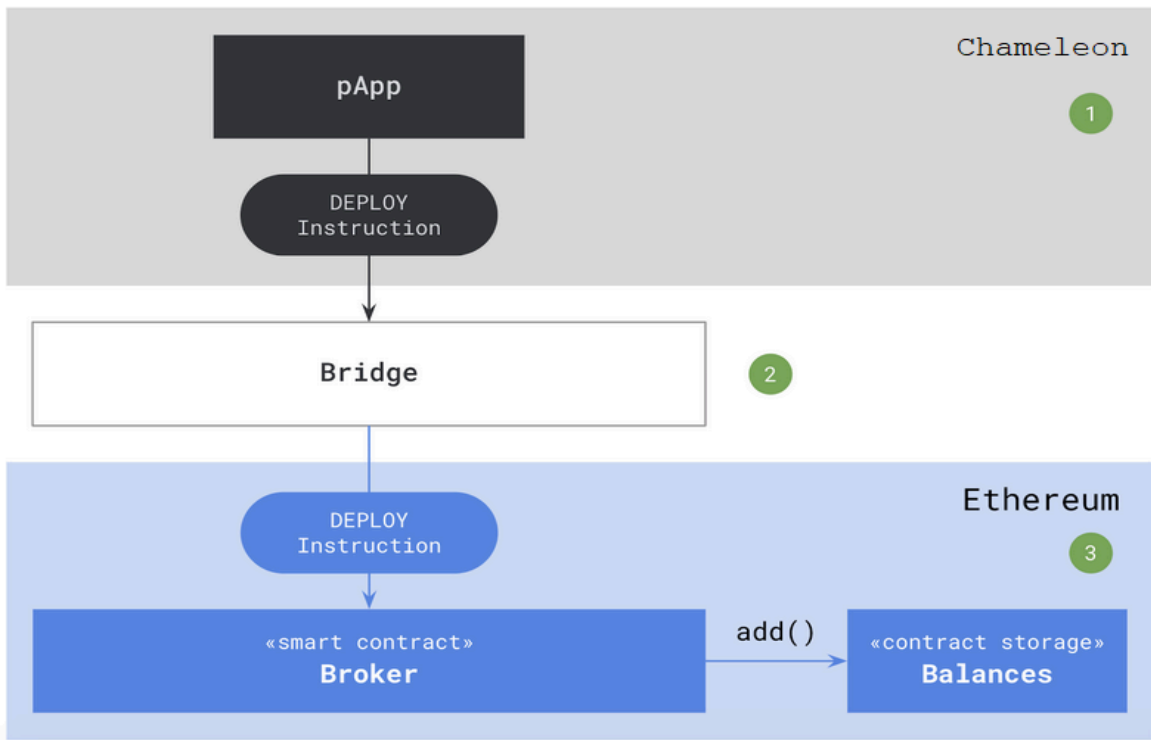
- 
1. A user initiates an unshielding transaction on Chameleon, specifying the privacy coins they want to unshield and the amount. Once the transaction is confirmed on **Chameleon**, the user obtains a burn proof.
  2. The user sends an UNSHIELD instruction to the Broker smart contract, along with the burn proof, via the **bridge**.
  3. The Broker smart contract parses the UNSHIELD instruction to retrieve the burn proof, which is verified by counting the number of signatures from Chameleon validators, as well as the burning parameters, which are used to transfer the public **ERC20** tokens back to the user.

## DEPLOY Instruction

Once shielded, privacy coin transactions are confidential and untraceable. However, they are limited to basic features like sending and receiving. DEPLOY, EXECUTE, and UNDEPLOY are instructions that enable users to use their privacy coins in their favorite Ethereum dApps. For example, users can trade **pETH** for **pDAI** on Uniswap or collateralize **pETH** to borrow **pUSDC** on Compound.

Deploying is the process of moving funds from Chameleon to Ethereum so that users can spend them in Ethereum dApps.

Following is an overview of the DEPLOY instruction flow:

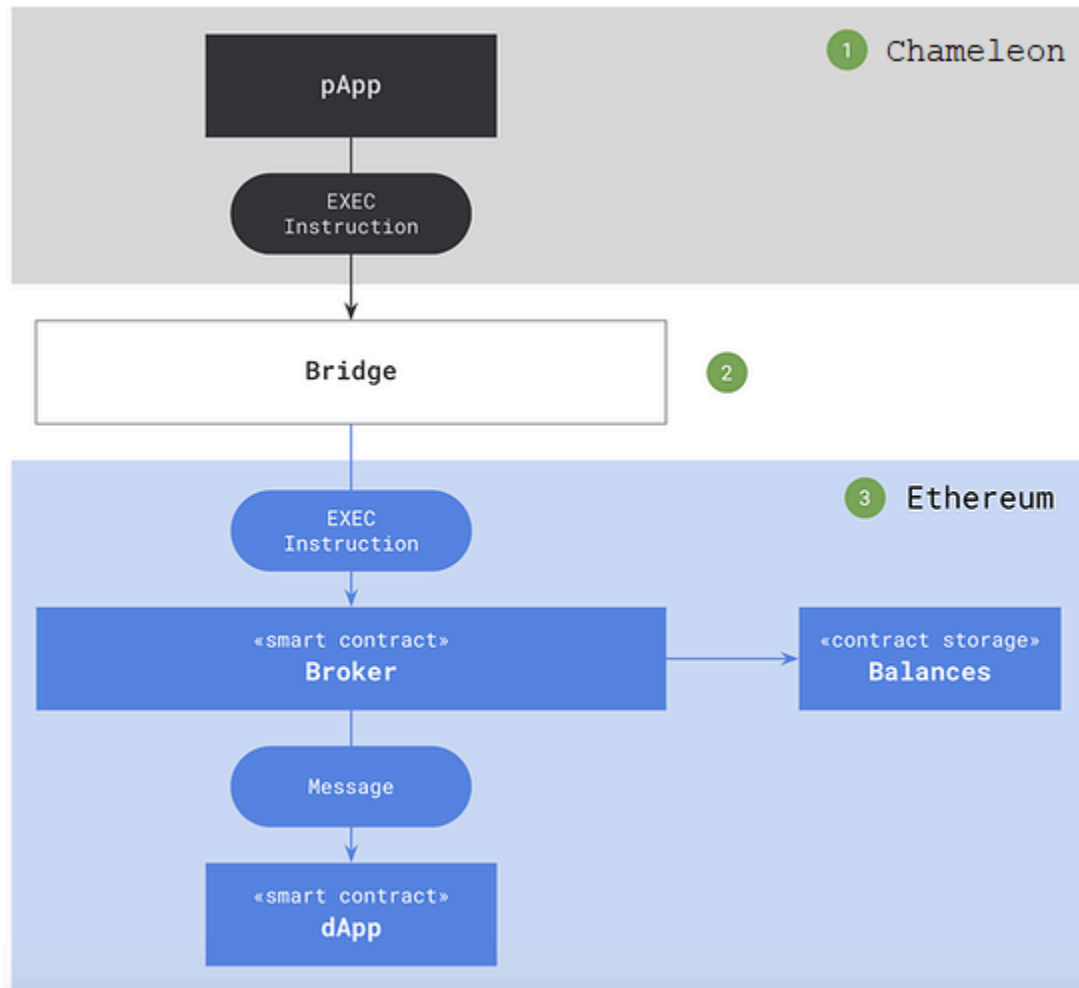


1. A user confidentially initiates a DEPLOY transaction on Chameleon, specifying the privacy coins they want to deploy and the amount. Once the transaction is confirmed on Chameleon, the user obtains a deploy proof, which functions similarly to a burn proof.
2. The user sends a DEPLOY instruction to the Broker smart contract, along with the deploy proof, via the bridge.
3. The Broker smart contract parses the DEPLOY instruction to retrieve the deploy proof, which is verified by counting the number of signatures from Chameleon **validators**, as well as the deploy parameters, which are used to update the user's deployed balances.

## EXECUTE instruction


Executing involves **anonymously** running a function call of an Ethereum smart contract. For example, performing operations like **swap(pETH, pDAI)** on Uniswap or **borrow(pUSDC)** on Compound while maintaining user anonymity.

The following is an overview of the EXECUTE instruction flow:



1. A user confidentially initiates an EXECUTE transaction on **Chameleon**, specifying the desired function call and parameters. Once the transaction is confirmed on Chameleon, the user obtains an execute proof.
2. The user sends an EXECUTE instruction to the Broker smart contract, along with the execute proof, via the bridge.
3. The Broker smart contract parses the EXECUTE instruction to retrieve the execute proof, which is verified by counting the number of signatures from Chameleon validators. The verified proof is then used to call the specified Ethereum smart contract function on behalf of the user.

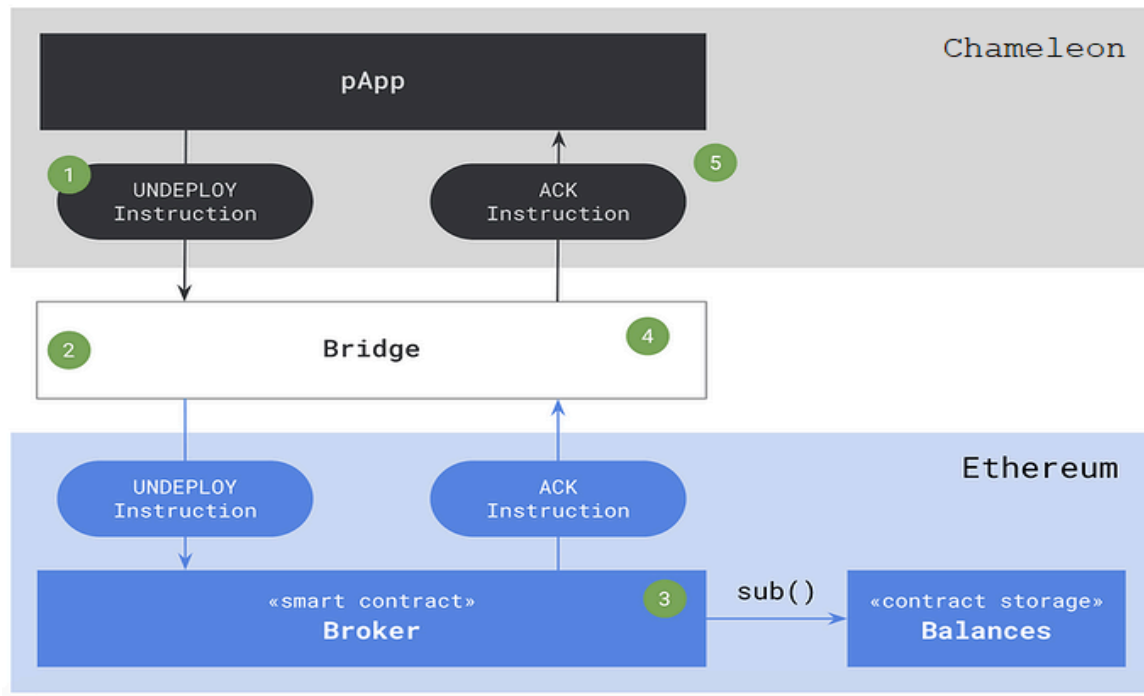
An EXECUTE instruction contains the following parameters:

- 
1. **Input Token:** The token the user intends to spend in this transaction.
  2. **Input Amount:** The amount of the input token to spend in this transaction, which must not exceed the user's balance in the Broker smart contract.
  3. **Output Token:** The token returned from the execution, if applicable.
  4. **dApp Contract Address:** The address of the target decentralized application (dApp) on Ethereum.
  5. **Encoded ABI:** The encoded Application Binary Interface (ABI) of the target function in the dApp contract.
  6. **Timestamp:** The timestamp of the transaction for reference and verification purposes.
  7. **Signature:** A cryptographic signature on the combined data of all the above parameters, ensuring the integrity and authenticity of the instruction.

## UNDEPLOY instruction

Undeploying is the reverse process of deploying: moving funds from Ethereum back to Chameleon.


Following is an overview of the UNDEPLOY instruction flow:



1. A user initiates an undeploy transaction on **Ethereum**, specifying the privacy coins they want to **undeploy** and the amount.
2. The bridge forwards the UNDEPLOY instruction to the **Broker smart contract**.
3. The Broker smart contract parses the UNDEPLOY instruction, verifies the user's signature, and subtracts the user's currently deployed balances. Once the transaction is confirmed on Ethereum, the user receives an **undeploy proof**.
4. The bridge forwards an ACK instruction to Chameleon, along with the undeploy proof.
5. Chameleon validators parse the **ACK** instruction to extract the undeploy proof, which is verified using Ethereum **Simplified Payment Verification (SPV)**, as well as the undeploy parameters. These are used to mint the privacy coin counterpart and send it to the user.

## pEthereum Developer Resources

The pEthereum SDK enables developers to build their own **pApps** on top of their existing dApps.



The pEthereum Developer Guide provides instructions for creating **privacy-protecting decentralized applications**—both from scratch and by integrating privacy features into existing dApps.

## Source Code

All Chameleon development is **public**. The code will be open-source on GitHub. Progress is shared through weekly updates on **chml.network** .

## Conclusion

Crypto's lack of privacy poses a significant threat to the growth of our new economy and slows the adoption of innovative financial products. We believe **privacy** is the missing piece for many everyday users.

We have proposed a method to build privacy-protecting **decentralized** applications on top of Ethereum. Rebuilding an EVM from scratch is unnecessary; Ethereum already boasts a vast developer and user base. By leveraging Ethereum's robust ecosystem, we can focus on addressing the privacy challenge. Developers can continue building dApps on Ethereum using Solidity while utilizing the **pEthereum SDK** to integrate **Chameleon** into their **dApps**.



## Future Work: Smart Contracts, Confidential Assets, Confidential IP, and More

Chameleon is continuously evolving. Ongoing research and development will be driven by the Core Development Team and the **Chameleon community**.

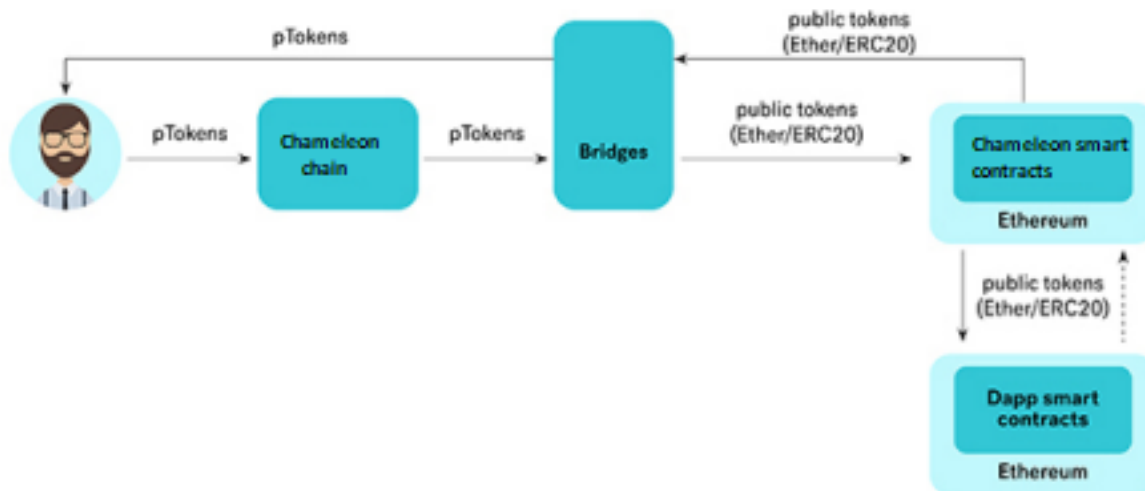
### Chameleon for Smart Contracts

Currently, thousands of developers around the world are writing smart contracts to build **decentralized applications (dApps)**. Privacy concerns, however, can be a barrier to adoption beyond the crypto niche. Traditional investors may be hesitant to reveal how much they trade on a trading dApp like **Uniswap** or how often they invest or borrow on a lending dApp like Compound.

What's needed is Chameleon for **smart contracts**. While there are other teams working on privacy for smart contracts (such as Oasis and RenVM [Cheng et al., 2019; Ren, 2019]), Chameleon believes in a different approach.

We're exploring a way to simply **unshield** the input when making a smart contract function call, then shield the output (or "returned value") from it. The smart contract runs as usual on Ethereum.

This approach avoids the need to build a new **Ethereum Virtual Machine (EVM)** from scratch. Instead, we reuse the existing Ethereum EVM and thousands of existing Ethereum dApps, focusing on solving the privacy issue for dApp users. Developers can continue building dApps on Ethereum as they normally would, while also giving users the option to access these dApps in Chameleon.



**Figure 1. Chameleon for smart contracts**

## Highway: Scaling Chameleon to 16,384 Validators

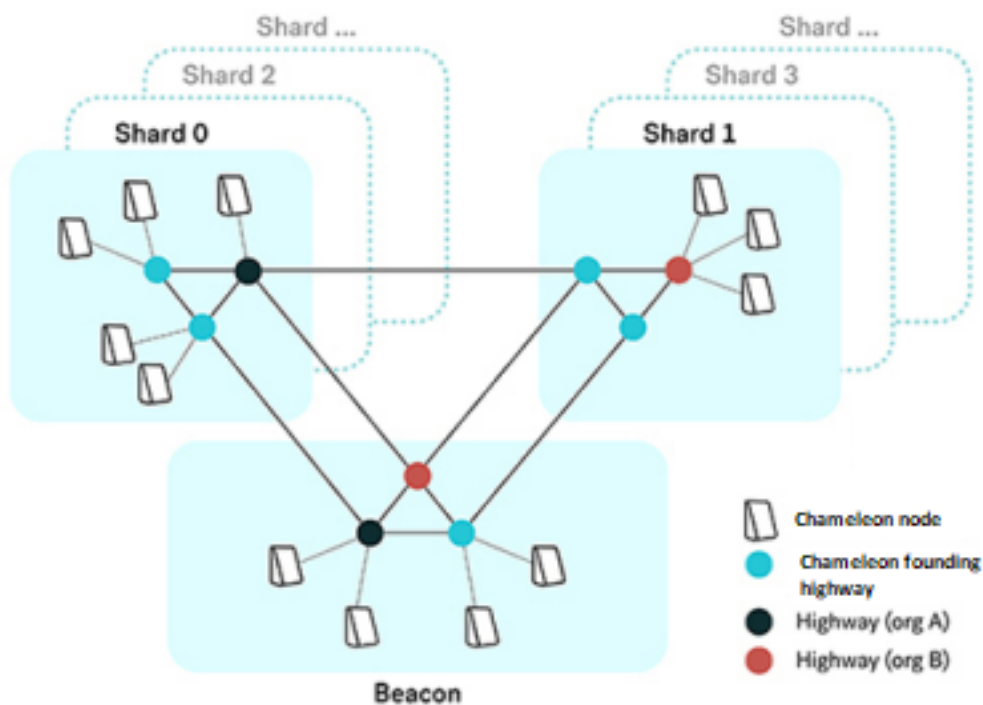
Despite its many advantages, **pBFT** consensus algorithms face high communication costs. As the number of validators grows, Chameleon needs a more efficient way to broadcast messages.

To address this, we're developing a new network topology called **Highway**, aimed at scaling Chameleon to 16,384 validators (64 shards x 256 validators per shard). Highway introduces specialized nodes responsible for receiving and forwarding messages within the Chameleon network. This new approach is designed to solve two key problems:

1. **Efficient Messaging:** Highway nodes serve as a fast and efficient channel for messages to be transmitted with minimal latency.
2. **Bypassing Network Barriers:** For Chameleon validators operating behind NATs and firewalls, Highway nodes are configured with public IP addresses and stable network connections, ensuring messages can still be forwarded effectively even when direct connections are hindered.

It's important to note that Highway nodes cannot alter or forge any content in the messages, as every message on the network is **cryptographically signed**. The sole purpose of Highway nodes is to ensure that messages reach their destination as quickly as possible.

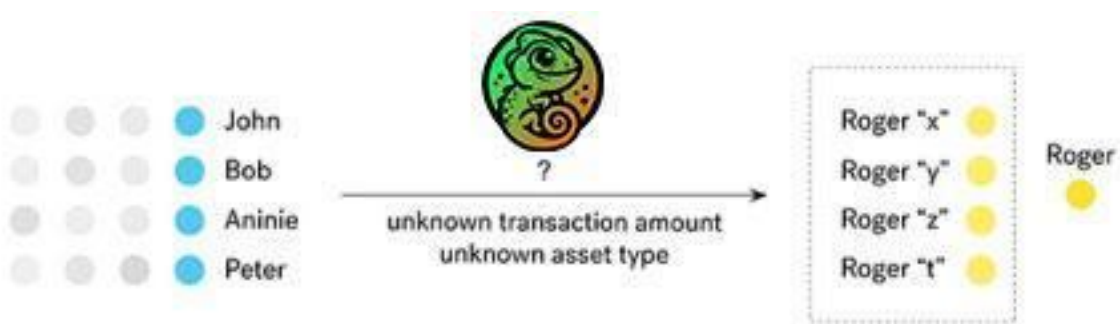




**Figure 2. Highway network topology**

## Confidential Assets: Unknown Asset Types

**Chameleon** will implement ring **signatures**, **stealth** addresses, and confidential transactions to shield senders, receivers, and transacted amounts. Unlike Zcash and Monero, Chameleon faces a unique problem as a platform for **privacy coins**: there are multiple privacy coins on **Chameleon**. We're working on adding Confidential Assets to Chameleon by also shielding asset types.



**Figure 3. Shielded asset types**

## Confidential IP on Chameleon Network

All senders and receivers are shielded, as are the transacted amounts. Soon, asset types will also be shielded. Currently, however, IP addresses are exposed when transactions are initiated. While IP addresses are not stored on Chameleon public ledger, it does leave a user open to network monitoring and analysis. This is a hard attack vector to pull off, but work is in progress to hide IP addresses and further improve the privacy of the **Chameleon network** [Bojja Venkatakrisnan et al., 2017; Kovri, 2018].

### Privacy Goals for Chameleon

#### 1. Improve Bulletproofs Verification Time

Bulletproofs offer an efficient method for confidential transactions. Chameleon aims to improve their implementation to speed up verification and enhance overall transaction throughput.

*Reference:* B. Bünz, J. Bootle, D. Boneh, et al. "Bulletproofs: Short Proofs for Confidential Transactions and More." Blockchain Protocol Analysis and Security Engineering, 2018.

[Link to paper.](#)

#### 2. Confidential Assets Based on Bulletproofs

Chameleon is working on further supporting confidential assets using Bulletproofs, ensuring privacy for different types of digital assets on the network.

*Reference:*


- TariLabs on Confidential Assets. [Link.](#)
- Cathie Yun. Building on Bulletproofs. [Link.](#)

## Availability

**Goal:** Increase robustness and availability of the chain

#### 1. Robustness and Availability

Chameleon aims to ensure that the network remains highly available even under variable validator numbers or high network traffic conditions. We are studying various designs to maintain optimal availability.



**Reference:** Christian Badertscher, et al. "Ouroboros Genesis: Composable proof-of-stake blockchains with dynamic availability." CCS 2018. [Link](#).

## 2. **Dynamic Sharding**

Chameleon plans to implement dynamic sharding, adjusting the number of shards based on the network's needs. This would enable the network to scale up or down depending on the requirements.

**Reference:** Alex Skidanov, Illia Polosukhin, "Nightshade: Near protocol sharding design." [Link](#).

## 3. **Random Number Generation**

The random number must have the following properties:

- **Unpredictable:** No one should be able to predict the random number before it's generated.
- **Unbiased:** The process of generating the random number should not be made biased by the participants.
- **Verifiable:** The validity of the generated random number should be verifiable.
- **Scalable:** The algorithm for randomness generation should scale to a large number of participants.

To achieve randomness, Chameleon currently uses the block hash of the Bitcoin chain. However, we are studying alternative approaches, such as Verifiable Random Functions (VRF) and **Verifiable Delay Functions (VDF)**, to independently generate an unbiased, unpredictable random number for the network.



**Reference:**

- E. Syta, P. Jovanovic, E. Kokoris-Kogias, N. Gailly, L. Gasser, I. Koffi, M. J. Fischer, and B. Ford. "Scalable Bias-Resistant Distributed Randomness." In 38th IEEE Symposium on Security and Privacy, May 2017. [Link](#).
- Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. "Verifiable Delay Functions." In CRYPTO 2018, 2018. [Link](#).

## Security

**Goal:** Strengthen the security of the chain

### 1. Staking & slashing mechanism

Currently, **Chameleon** fixes the required staking amount. We are studying how to make the mechanism dynamic, to create incentives for more validators to join the network. The slashing mechanism will prevent the misbehaving nodes from harming the chain.

**Reference:**

A Kiayias, A Russell, B David, R Oliynykov - 2017, Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol. [Link](#).

### 2. Sharding: State Validity and Data Availability

Sharding, as a method of partitioning a blockchain's data, introduces a potential challenge regarding state validity and data availability. Unlike traditional non-sharded blockchains, where participants validate the entire chain, sharding complicates this by breaking the network into multiple partitions or "**shards**." In a sharded network, participants cannot always be certain that the state they interact with corresponds to a valid sequence of blocks, or that the sequence of blocks is the canonical chain of that shard.

To address this challenge, we are investigating the use of advanced cryptographic techniques like **Merkle Trees** and **Polynomial Commitments**. These solutions allow participants to verify the correctness of data with minimal data retrieval, rather than having to download and verify the full history of a shard.

## References:

- [Using Polynomial Commitments to Replace State Roots](#)

Additionally, the **Fisherman concept**, where any honest validator can provide proof of an invalid block with only a small amount of information, will help maintain the integrity of Chameleon's sharded structure.

We are also exploring Erasure Codes and leveraging ideas from **Polkadot's approach**, where participants store only part of the data. This allows them to cooperate and verify the validity of the entire chain, thus ensuring the correctness and availability of data even in the absence of full node participation.

## References:

- [Nightshade: Near Protocol Sharding Design](#)
- [Polkadot White Paper](#)

## Scalability

**Goal:** Maximize Transaction Throughput

Throughput remains one of the most significant challenges in blockchain technology. Chameleon is addressing this issue by applying **Sharding** techniques at the core layer to enhance scalability. This enables the network to process multiple transactions in parallel, dividing the workload across multiple shards and improving overall throughput.

In addition, we are studying how to incorporate Lightning Network and Payment Channel techniques at the second layer. By utilizing these off-chain scaling methods, **Chameleon** aims to further increase transaction speeds and reduce congestion on the main chain, ensuring fast and efficient transaction processing.


## References:

- [Lightning Network Paper](#)
- [A Survey on the Lightning Network](#)

## On-chain Storage

**Goal:** Minimize Data Stored On-chain

**Chameleon** aims to reduce the amount of data stored on-chain by applying advanced



cryptographic techniques such as zk-SNARKs. This allows us to minimize the ledger size while maintaining the **security** and validity of the data. We are carefully weighing the tradeoff between keeping transaction history for transparency and reducing the amount of data stored, such as by exploring the possibility of removing used **UTXOs** (Unspent Transaction Outputs).

#### References:

- [zk-SNARKs Paper](#)
- [Coda Protocol Whitepaper](#)
- [Grin Whitepaper](#)

#### Research on ZKPs: Recent Advances

**Goal:** Leverage Advances in Zero-Knowledge Proofs to Improve Chameleon

In line with broader advancements in **cryptographic** technologies, Chameleon is actively investigating recent developments in zero-knowledge proofs (ZKPs). We are focused on integrating these innovations to improve the privacy, scalability, and efficiency of our platform. Notable advancements include Halo, a recursive proof composition system that eliminates the need for a trusted setup, and SuperSonic, which offers a practical zk-SNARK with a nearly trustless setup.

By incorporating these advancements into **Chameleon**, we aim to push the boundaries of privacy-preserving technologies and improve our blockchain's performance.

#### References:

- [Halo: Recursive Proof Composition](#)
- [SuperSonic: Practical zk-SNARK](#)
- [Awesome Zero-Knowledge Proofs](#)

---

## Conclusions, Acknowledgments, and References ▶

### Conclusions

We are building a new **privacy-centric cryptonetwork** to enhance privacy for other blockchain ecosystems. To achieve this, we have developed a decentralized network of trustless validators and implemented advanced cryptographic techniques such as linkable ring signatures, homomorphic commitments, and zero-knowledge range proofs. To scale the network's performance, we are using innovative solutions like sharding, **pBFT**, and **proof-of-stake**.

While **Chameleon** has not yet launched its mainnet, we are actively working on its development. We aim to provide users with a privacy-preserving infrastructure for **decentralized** applications, and when launched, Chameleon will feature a highly scalable architecture capable of handling increasing transaction volumes.

**Cryptocurrencies** continue to evolve at a rapid pace. New assets are being introduced, some of which enhance the efficiency of traditional assets like fiat (USDC) or commodities (DGX), while others give rise to entirely new asset classes, such as programmable governance tokens (MKR). If this trend continues, cryptocurrencies will likely become a central part of both individual and institutional portfolios. **Chameleon** aims to ensure that these digital assets, and their owners, will have the privacy and security they deserve in an increasingly transparent world.

### Acknowledgments

We extend our heartfelt thanks to the 600+ founding nodes for their pivotal role in powering the **Chameleon network** since its inception.

Furthermore, we are grateful for the support from our strategic partners, the broader crypto community, and the early adopters whose continuous feedback and engagement help us improve and evolve the network every day.

## References

- [Adam, 2018] Adam, H. (2018). *Uniswap whitepaper*. URL: [https://hackmd.io/C-DvwDSfSxuh-Gd4WKE\\_ig](https://hackmd.io/C-DvwDSfSxuh-Gd4WKE_ig).
- [Baneth, 2019] Baneth, T. (2019). *Waterloo - a decentralized practical bridge between EOS and Ethereum*. URL: <https://blog.kyber.network/waterloo-a-decentralized-practical-bridge-between-eos-and-ethereum-1c230ac65524>.
- [Benet and Dias, 2019] Benet, J., and Dias, D. (2019). *libp2p specification*. Technical report, URL: <https://github.com/libp2p/specs>.
- [Boneh et al., 2018] Boneh, D., Drijvers, M., and Neven, G. (2018). Compact multi-signatures for smaller blockchains. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 435-464. Springer.
- [Boudot, 2000] Boudot, F. (2000). *Efficient proofs that a committed number lies in an interval*. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 431-444. Springer.
- [Bojja Venkatakrisnan et al., 2017] Bojja Venkatakrisnan, S., Fanti, G., and Viswanath, P. (2017). *Dandelion: Redesigning the Bitcoin network for anonymity*. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 1(1):1-34.
- [Bünz et al., 2018] Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., and Maxwell, G. (2018). *Bulletproofs: Short proofs for confidential transactions and more*. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315-334. IEEE.
- [Buterin et al., 2014] Buterin, V. et al. (2014). *A next-generation smart contract and decentralized application platform*. White paper, 3(37).
- [BTC Relay, 2019] BTC Relay (2019). *A bridge between the Bitcoin blockchain & Ethereum smart*





contracts. URL: <http://btcrelay.org>.

[Castro et al., 1999] Castro, M., Liskov, B., et al. (1999). *Practical Byzantine fault tolerance*. In *OSDI*, volume 99, pages 173-186.

[Chaum and Van Heyst, 1991] Chaum, D. and Van Heyst, E. (1991). *Group signatures*. In *Workshop on the Theory and Application of Cryptographic Techniques*, pages 257-265. Springer.

[Cheng et al., 2018] Cheng, R., Zhang, F., Kos, J., He, W., Hynes, N., Johnson, N., Juels, A., Miller, A., and Song, D. (2018). *Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contract execution*. *arXiv preprint arXiv:1804.05141*.

[Croman et al., 2016] Croman, K., Decker, C., Eyal, I., Gencer, A. E., Juels, A., Kosba, A., Miller, A., Saxena, P., Shi, E., Sirer, E. G., et al. (2016). *On scaling decentralized blockchains*. In *International conference on financial cryptography and data security*, pages 106-125. Springer.

[Diffie and Hellman, 1976] Diffie, W. and Hellman, M. (1976). *New directions in cryptography*. *IEEE Transactions on Information Theory*, 22(6):644-654.

[Dwork and Naor, 1992] Dwork, C. and Naor, M. (1992). *Pricing via processing or combatting junk mail*. In *Annual International Cryptology Conference*, pages 139-147. Springer.

[Finney, 1993] Finney, H. (1993). *Detecting double-spending*. URL: <https://nakamotoinstitute.org/detecting-double-spending>.

[Fujisaki and Suzuki, 2007] Fujisaki, E. and Suzuki, K. (2007). *Traceable ring signature*. In *International Workshop on Public Key Cryptography*, pages 181-200. Springer.

[Gentry and Boneh, 2009] Gentry, C. and Boneh, D. (2009). *A fully homomorphic encryption scheme*, volume 20. Stanford University.

[Go, 2009] Go, T. (2009). *The Go programming language specification*. Technical report, [http://golang.org/doc/doc/go\\_spec.html](http://golang.org/doc/doc/go_spec.html), Google Inc.

[Goldreich et al., 1991] Goldreich, O., Micali, S., and Wigderson, A. (1991). *Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems*. *Journal of the ACM (JACM)*, 38(3):690-728.

[Privacy mode for Ethereum. Technical report.](#)

[pDEX: The first privacy-protecting decentralized exchange. Technical report.](#)

[Jedusor, 2016] Jedusor, T. E. (2016). *Mimblewimble*. URL: <https://scalingbitcoin.org/papers/mimblewimble.txt>.

[Juels, 1999] Juels, A. (1999). *Client puzzles: A cryptographic countermeasure against connection depletion attacks*. In *Proc. Networks and Distributed System Security Symposium (NDSS)*, 1999.

[King and Nadal, 2012] King, S. and Nadal, S. (2012). *Ppcoin: Peer-to-peer cryptocurrency with proof-of-stake*. Self-published paper, August, 19.

[Kokoris-Kogias et al., 2018] Kokoris-Kogias, E., Jovanovic, P., Gasser, L., Gailly, N., Syta, E., and Ford, B. (2018). *Omniledger: A secure, scale-out, decentralized ledger via sharding*. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 583-598. IEEE.

[Kovri, 2018] Kovri. *The Kovri Project*, 2018. URL: <https://gitlab.com/kovriproject/kovri>.

[Kwon, 2014] Kwon, J. (2014). *Tendermint: Consensus without mining*. Draft v. 0.6, fall, 1(11).

[Li et al., 2018] Li, C., Li, P., Zhou, D., Xu, W., Long, F., and Yao, A. (2018). *Scaling Nakamoto consensus to thousands of transactions per second*. *arXiv preprint arXiv:1805.03870*.

[Liu et al., 2004] Liu, J. K., Wei, V. K., and Wong, D. S. (2004). *Linkable spontaneous anonymous group signature for ad hoc groups*. In *Australasian Conference on Information Security and Privacy*, pages 325-335. Springer.

[Luu et al., 2016] Luu, L., Narayanan, V., Zheng, C., Baweja, K., Gilbert, S., and Saxena, P. (2016). A

secure sharding protocol for open blockchains. Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security.

[Luu and Yaron, 2017] Luu, L. and Yaron, V. (2017). *Kyber Network: A trustless decentralized exchange and payment service*. URL: <https://coinpaprika.com/storage/cdn/whitepapers/539.pdf>

[Maxwell, 2015] Maxwell, G. (2015). *Confidential transactions*. URL: [https://people.xiph.org/greg/confidential\\_values.txt](https://people.xiph.org/greg/confidential_values.txt) (Accessed 09/05/2016).

[Merkle, 1980] Merkle, R. C. (1980). *Protocols for public key cryptosystems*. In *1980 IEEE Symposium on Security and Privacy*, pages 122-122. IEEE.

[Morais et al., 2019] Morais, E., Koens, T., Van Wijk, C., and Koren, A. (2019). *A survey on zero knowledge range proofs and applications*. *SN Applied Sciences*, 1(8):946.

[Nakamoto, 2008] Nakamoto, S. (2008). *Bitcoin: A peer-to-peer electronic cash system*. URL: <http://www.bitcoin.org/bitcoin.pdf>.


[Noether et al., 2016] Noether, S., Mackenzie, A., et al. (2016). *Ring confidential transactions*. *Ledger*, 1:1-18.

[Pedersen, 1991] Pedersen, T. P. (1991). *Non-interactive and information theoretic secure verifiable secret sharing*. In *Annual International Cryptology Conference*, pages 129-140. Springer.

[Reid and Harrigan, 2013] Reid, F. and Harrigan, M. (2013). *An analysis of anonymity in the bitcoin system*. In *Security and Privacy in Social Networks*, pages 197-223. Springer.

[Ren, 2019] Ren (2019). *A privacy preserving virtual machine powering zero-knowledge financial applications*. URL: <https://renproject.io/litepaper.pdf>.

[Rivest et al., 2001] Rivest, R. L., Shamir, A., and Tauman, Y. (2001). *How to leak a secret*. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 552-565. Springer.



[Van Saberhagen, 2013] Van Saberhagen, N. (2013). *Cryptonote v 2.0*. URL:  
<https://cryptonote.org/whitepaper.pdf>.

[Sasson et al., 2014] Sasson, E. B., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., and Virza, M. (2014). *Zerocash: Decentralized anonymous payments from bitcoin*. In *2014 IEEE Symposium on Security and Privacy*, pages 459-474. IEEE.

[Szabo, 2005] Szabo, N. (2005). *Trusted third parties are security holes*. White Paper. URL:  
<https://nakamotoinstitute.org/trusted-third-parties>.

[TBTC, 2019] TBTC (2019). *tBTC: A decentralized redeemable BTC-backed ERC-20 token*. URL:  
<http://docs.keep.network/tbtc/index.pdf>.

[Visa, 2018] Visa (2018). *Visa acceptance for retailers*. URL:  
<https://usa.visa.com/run-your-business/small-business-tools/retail.html>.

[WBTC, 2019] WBTC (2019). *Wrapped Bitcoin*. URL:  
<https://www.wbtc.network/assets/wrapped-tokens-whitepaper.pdf>.

[Wood, 2014] Wood, G. (2014). *Ethereum: A secure decentralised generalised transaction ledger*. URL: <https://ethereum.github.io/yellowpaper/paper.pdf>.

[Zamani et al., 2018] Zamani, M., Movahedi, M., and Raykova, M. (2018). *Rapidchain: A fast blockchain protocol via full sharding*. *IACR Cryptology ePrint Archive*, 2018:460.

[Zilliqa, 2017] Zilliqa, T. (2017). *The Zilliqa technical whitepaper*. URL:  
<https://docs.zilliqa.com/whitepaper.pdf>.

