OpenST Protocol White Paper

Benjamin Bollen, Nishith Shah, Lionello Lunesu, Sunil Khedar, Antoine Cote, Jason Banks, Jason Goldberg, Matt Chwierut, Brian Lio

Draft published for peer review* v0.8.3, last updated 5 November 2017

1 Introduction

Ethereum introduced to the blockchain toolset stateful accounts. The storage space associated with these accounts is write-protected by code, and we refer to these accounts as smart contracts[1] This general purpose capability of Ethereum has spurred a vast wave of innovations and a leading use-case of Ethereum has been the ability to create a token on top of Ethereum. For tokens on Ethereum, ERC20[2] has recently been adopted as the standard interface for a smart contract defining such a token. However in order to engineer utility into a token to incent greater uptake, several more challenges become apparent. Some of these challenges include latency, transaction fees, scale and privacy; for these challenges we attempt to contribute towards solutions in this technical whitepaper. Other challenges come from legal requirements and economic modeling to support a token economy; confronting these forms an integral part of the Simple Token project[3].

With Simple Token we set out to be pragmatic about the capabilities of existing decentralization technologies and look to find answers that solve for these problems today with a clear roadmap towards internet-scale performance. All the while we strive that all technical mechanisms are open and independently cryptographically verifiable.

OpenST operates as a non-profit and governs the development of the OpenST Protocol. In addition it performs high-level guardian tasks on the instance of the protocol that is associated with the Simple Token EIP20 token on public Ethereum. These guardian tasks are limited but necessary when a technical answer only is insufficient. A primary example can be the review of a new member company which desires to launch its own branded token within the OpenST platform.

First we establish a lexicon to help present the new patterns OpenST Protocol introduces in the token space. For a young token economy, speculative value of a token can easily drown out the intended utility. We describe how a utility token can

*for review contact review@simpletoken.org

[1]We use smart contract interchangeably for the code associated with an account, or the stateful instantiation of that code; the meaning should be clear from the context.

[2]See EIP20 (formerly ERC20)

[3]Find our thinking on governance, economic modeling and incentives in OST sidepapers.

be built on top of value assets that back it.

We introduce Simple Token as a freely tradable EIP20 token on Ethereum mainnet. Simple Token can be staked as a valuable crypto-asset to mint utility tokens. Furthermore Simple Token functions as the base token on the utility chains accounting for gas consumption.

We continue to outline how desired behavior can be enforced on the utility tokens so that the token serves the user transactions within an existing consumer application. We call such utility tokens branded tokens. We describe how user financial sovereignty is preserved on the blockchain as a user can choose to hard-exit the value of the branded tokens on Ethereum mainnet.

In the next section of the protocol we describe how rich interactions in consumer applications can be mapped to fundamental transactions on the blockchain by providing an API for developers to integrate OpenST into their application - making the development of a consumer tokenized economy simple.

While these last mechanisms are off-chain mechanisms, we consider them an integral part of the protocol to enable any third-party developer to build on top of the OpenST Platform. Additionally including them in the protocol enables open innovation and audits to realize best practices for minimizing correlatable data on chain of pseudo-anonymous accounts that could empower inferring personally identifiable data or application metrics. Future work in this part of the protocol layer as such includes technology to increase the noise-to-signal ratio on-chain, or reduce the signal by moving it off-chain with payment channels.

To conclude we describe how no trust needs to be placed in the validator pool of the utility chain, as in the case of chain halting, all ownership state can be carried back to the value chain.

We recapitulate the protocol at a high level using an explicit example and build on this example to illustrate how an end-user can use Simple Token on Ethereum to obtain and interact with a branded token.

We describe how OpenST Platform can be an open network of utility chains, serving different consumer applications and provided by third-parties.

While the OpenST Protocol and Platform concerns only application logic - and OpenST will focus on implementing it as Ethereum smart contracts - and off-chain technology, it is still worthwhile to detail our considerations with respect to the available chain technology upon which the OpenST Platform can execute. We discuss these architectural requirements lastly and how OpenST can contribute to existing projects in this area. This concludes the OpenST Platform as the second part of this paper.

Lastly we put forward the roadmap for OpenST.

2 Related Work

OpenST has been born out of the chasm between two worlds. One side holds the promise of open payment networks and financial sovereignty of users in a digital world. On the other side sit millions of potential users for whom the technical adoption curve is steep, and businesses who are looking to tokenize, but who need to

work within existing regulations and tax law. Challenges currently facing cryptocurrencies and applications hoping to leverage tokenization include user-experience, economic and legal constraints and the opportunities that can be unlocked with the right technical solution. While the OpenST protocol white paper may read at times like a scalability proposal - and obviously a scalable architecture is a requirement - it is not, nor do we intend it to become that. We set out to build on and contribute to the work done by great teams to present a protocol that helps bridge cutting-edge blockchain technology with mainstream consumer applications.

Interledger Protocol - Interledger uses a two-phased escrow on two ledgers where a connector has funds on both ledgers and functions as a market maker between the two chains; additionally allowing for a graph of connectors to transfer across multiple hops. We drew inspiration from the Interledger protocol, but solved for a different problem: we look to mint a new token representation on a utility chain of the value staked on a value chain.

Tendermint - Tendermint is a leading protocol for cryptographically sealed Byzantine fault-tolerant, Proof-of-Stake consensus. It is used / implemented by Ethermint, Parity and Hyperledger Burrow.

Cosmos - Cosmos by Tendermint is a framework for interoperability between blockchains. It has pioneered the concept of InterBlockchain Communication (IBC), which we apply here in a specific context for transferring proofs of utility between Ethereum and a utility chain. Cosmos as a network also provides structure between different chains. We explore for utility chains to be natively compatible as Cosmos zones (when running Tendermint consensus).

Lightning / Raiden - work on payment channels has both inspired the OpenST protocol, and also forms a natural complement to OpenST. Raiden can transfer Simple Token between different utility chains (and Ethereum mainnet). Payment channels specifically can be hosted by member companies to scale the volume of transactions for their branded token off the utility chain. In this sense payment channels also aid in protecting user privacy and help shield a member companys inner metrics of its application to the outside world, while being open and accountable with their branded tokens.

Plasma - while we didnt learn about Plasma until it was published, we are enthused that OpenST can be seen as a very specific application of some of the ideas that are also abstractly proposed in the Plasma white paper. We welcome the alignment and look forward to mutual contributions.

Polkadot - Polkadot is a protocol for heterogeneous general state transfer between multiple chains. With OpenST we aim to solve a particular problem, but see benefits to strengthening the guarantees against Byzantine validators across utility chains.

Casper - as OpenST spans out over many utility chains with all value staked on Ethereum, all work to strengthen and scale Ethereum greatly benefits the ecosystem, and as a result OpenST.

3 A Protocol for Simple Tokens

3.1 Staking Value for Utility

The OpenST Protocol establishes a bridge between two differently purposed blockchains. A *value blockchain*, which is required in order to hold cryptographically secured valuable assets; and a *utility blockchain*, which has utility tokens in favor of which the assets are held on the value blockchain. The utility chain needs to support lower transaction fees and lower transaction confirmation times than the value chain. We put forward this condition seeing as one desired outcome of the OpenST Protocol is to enable micro-transactions within mainstream consumer applications using the utility tokens.

We present the discussion for two Ethereum-based chains, but note that this is not a requirement for the value chain.

Through the lens of the Simple Token project and the Simple Token EIP20 token on public Ethereum, the value chain in our discussion is planned to be Ethereum mainnet. For the utility chain we consider an Ethereum-based chain with a Byzantine fault-tolerant consensus engine which seals blocks cryptographically, as examples Proof-of-Authority[4] or Proof-of-Stake consensus engines.

[4]Proof-of-Authority is not Byzantine fault-tolerant, but it can still be considered useful in the context of utility chains.

3.2 Establishing a Bridge

To establish a channel between two chains, we require both chains to have a light client smart contract on each chain tracking the blocks on the other chain. In several configurations prior or ongoing work already exists.

When we take into consideration the specifics of these chains then we can consider specific light client contracts which relieve the responsibility of a central party on mutually committing the latest state hashes on the other chain.

As Ethereum mainnet operates a Nakamoto consensus engine (as the value chain) there is a soft requirement to set a threshold for the number of block confirmations to wait for until a state transition on Ethereum is considered finalised. If the utility chain is cryptographically validated by a known set of validators then those validators can each report the block hashes they have seen on public Ethereum and a block hash is considered final when consensus among the validators is reached.

In the opposite direction, to report the latest block hash of the utility chain to the value chain, the logic involves no subjective threshold parameter when the utility chain is cryptographically sealed by a known set of validators as a complete light client can be implemented as a smart contract on Ethereum.

In general a value chain is a Proof-of-Work generated chain (for the near future at least), while a utility chain for efficiency reasons should be assumed to be cryptographically sealed. It is therefore worthwhile to note that this asymmetry is by design: any halting or Byzantine failure of

the utility chain can be proven by any user on the value chain to forcefully recover the staked assets on the value chain after a sufficiently long grace period should the utility chain fail to recover. This way users are assured to always be able to recover their original assets on the value chain. On the other hand a value chain can hard-fork at which point the utility chain can evaluate out of band whether to track both or one of the forked value chains going forward.

As a net result of having complementary light client tracking contracts on both chains, the smart contracts have at their disposition knowledge of the state root of the other chain[5]. Such transactions committing the blocks allow consequentially for state from the other chain to be asserted as true only if a Merkle-proof for those state variables can be proven against the committed root hash.

Committing the root hash of the alternate chain allows for any party to present statements of what is true on that chain, removing the need for trusted oracles or trusted parties. A second benefit of this approach is that it strengthens immutability of either chain as the latest blocks are anchored into an independent chain. In particular if the utility chain is cryptographically sealed, anchoring the latest block on regular intervals into a (Proof-of-Work) value chain prevents any of the validators from rewriting the block history.

[5]What we outlined here is in effect an application of the concept of InterBlockchain Communication (IBC) as proposed by Tendermint in the Cosmos Whitepaper.

3.3 Two Sides of the Same Token

Tokens form a natural basis upon which to build functional sharding. Tokens, like smart contracts, are contained within the blockchain they are defined on. Unlike smart contracts, tokens have a universal metric for coarse graining, namely their total supply, and the extrinsic property of their market valuation.

If we consider the total supply of a token then all forms of monetary transactions leave the total supply of tokens unchanged. Whether it concerns a send transaction, an escrow, or even an Interledger protocol exchange across different ledgers, within the chain defining the tokens, the total supply is unaffected by these transactions.

By grouping value on a value chain we can denominate that value as a new token, a utility token, and transactions of the utility token do not change the total amount of grouped value. If the utility token would be defined on the same blockchain, then we would not have gained additional transaction throughput capacity. However, by defining this utility token on a new chain, the utility chain, transactions of the utility token need not concern the value chain, and we have a logical model for functional sharding of transactions.

When a user adds crypto-assets on the value chain to the grouped value, she should expect to receive an equivalent amount of utility tokens on the utility chain. While we call this process minting utility tokens, no value is created, only a new representation of that original value is created while the value assets themselves are locked.

In the reverse process, if she can prove

ownership of utility tokens and intent to return them in favor of an equivalent stake of the grouped value, then our user can do so at any point. As the utility tokens are backed for the full value at any time, all users can run on the bank and they would all recover their full value on the value chain.

3.4 Minting Utility Tokens

To mint utility tokens on a utility chain out of value staked on a value chain, or to redeem value on the value chain by relinquishing ownership of utility tokens on the utility chain, the protocol needs to atomically act on two blockchains. OpenST Protocol requires a two-phased commit for either action. To declare a utility token the user needs to deploy a staking contract with hashed timelock escrow (HTLC) functionality on the value chain; on the utility chain the user deploys a corresponding minting contract with a similar HTLC escrow, where both contracts need to be linked to the respective light client contract that tracks the state root of the opposite chain.

When a user wants to stake value she can transfer her crypto-assets into the escrow of the staking contract on the value chain. The escrow function of the staking contract must hash the *intent data*[6] of the user together with a staking sequence number, chain identifier and the escrow time-out block height. By storing this hash under a key in the Merkle-tree a Merkle-proof can be constructed that provides the hash path of this key and its stored value to the state root of the block in which this transaction is accepted (or a later block for as long as the hash is stored in the escrow). We require a unique sequence number to be included in the pre-image data to avoid a replay attack, similarly as a sequence number is included in a transaction.

The pre-image data together with the hash and its Merkle-proof can now be considered as a mint-precommit. The user has declared the intent to stake a certain amount of crypto-assets on the value chain in favor of obtaining utility tokens on the utility chain into the same address - as she controls the private key for that address on the value chain, she also controls the same address on the utility chain.

Before the mint-precommit can be accepted on the utility chain, the light client tracking contract on the utility chain needs to acknowledge a block of the value chain in which the staking escrow contract holds the crypto-assets of the user. Once such a block is accepted, the user can submit a transaction with the mint-precommit to the minting contract on the utility chain. The minting contract can verify the Merkle-proof included in the mint-precommit against the relevant state root it can query from the light client tracking contract; it can verify that the pre-image data in the mint-precommit hashes to the one proven by the Merkle-proof.

Note that the security for the information transfer between the chains is not derived from the transaction signature. Anyone can submit the mint-precommit on behalf of the user. Whether or not the mint-precommit is valid is determined by whether it is consistent and can be proven against the light

[6]Intent data of moving crypto-assets into the staking contract escrow must include the asset identifier, amount staked and the user account.

client tracking contract.

If the minting contract on the utility chain determines the mint-precommit is valid and with sufficient time left on the staking escrow, it can mint the corresponding amount of utility tokens into timed-release escrow[7] of the minting contract to benefit the user who staked the original crypto-assets on the value chain.

To release the minted utility tokens from the escrow the user needs to present a signed receipt to the utility chain. This receipt is the *mint-commit*. The same mint-commit needs to be presented to the value chain to move the crypto-assets from the escrow into the staking contract fully.

While only the user can sign a receipt to complete the second phase of the minting process, we want to ensure that the receipt is first presented to the value chain before it is presented to the utility chain. Once presented on either chain, any observer can carry it to the other chain. The user however can stand to benefit to only present the receipt on the utility chain, and not on the value chain, as it would (after long time-out) release the crypto-assets from the escrow.

We therefore require the user to have put forward a bounty before she can start the two-phase minting process. Anyone who presents the receipt to the value chain, who is not the user, will receive a fraction of the bounty and the remainder is donated to a non-OpenST foundation. This stake gives a strong incentive for the user to present the receipt to the value chain first (as the signature of the receipt is unknowable by others until first presented) and then secondly she can present the receipt to the utility chain for risk of never obtaining the utility tokens she now staked for.

Should she not present the receipt in this order - first on the value chain, secondly on the utility chain - she might hope to obtain both the utility tokens and eventually have her crypto-assets reverted back to her. However by presenting the receipt on the utility chain, her signature becomes known, and anyone can race to present the receipt on the value chain before her, claiming her bounty and completing the two-phased minting process successfully. We note that while the two-phased commit can reasonably complete in three blocks on the value chain, the escrow on the staking contract should have a large time-out in the order of weeks (or longer) to block any attempts on gaming the synchronicity of carrying back receipts between the chains.

The net effect of the two-phased commit process has been that the users crypto-assets are controlled by the staking contract on the value chain and the minting contract on the utility chain has minted an equivalent amount of utility tokens on the utility chain and transferred them to the user. To unstake crypto-assets on the value chain ownership of utility tokens needs to be proven on the utility chain. This process runs analogous to the staking process but the value chain and utility chain are in-

[7] This escrow needs to revert before the escrow on the value chain allows reverting. Safe margins can be made because approximations to the relative blockspeed of the two chains is known and considered a constant, but the light client tracking contract allows for precise triggers to ensure the correct order of closing on both chains. Note that the time-out on staking escrow should be as long as acceptable as a mechanism design choice. The time-out on the minting escrow contract can be acceptably short.

terchanged.

The user has to move her utility tokens into escrow on the minting contract. This escrow contract now carries the long time-out and bounty requirement. The user can construct the *claim-precommit* from the hash stored by the escrow resulting from the relevant intent data and similar identifiers as we listed for the mint-precommit. She can present the claim-precommit to the staking contract on the value chain once the light client tracking contract on the value chain acknowledges the block against which she can prove her claim-precommit. On validation of the claim-precommit, the staking contract can move the equivalent portion of the staked value into escrow benefiting the user. Similarly as before the user can only move the escrows forward by presenting a signed receipt, the *claim-commit*, and similarly we want to ensure that she has an incentive to first present it to the utility chain which is accomplished by the long time-out on this escrow and the bounty[8] to benefit any other user presenting her receipt first.

As there is a strong symmetry between the two processes we will present the specification of these processes in an abstracted form in the specification documents. We call this part of the OpenST Protocol *Proof-of-Utility*, as it provides cryptographic proof that a minted token on a more performant substrate has been backed with cryptographically valuable assets on a different chain, anchoring the value of the token, and freeing up its utility.

3.5 Simple Token EIP20

A utility chain, in this case an Ethereum-based chain, cannot function without a base token that can pay for the gas cost of transactions. On Ethereum mainnet Ether is this token, and gas is - of sorts - a hidden utility token bought and sold at the beginning and end of every transaction execution from and to the miner. While on a cryptographically sealed chain the costs incurred by validators securing the chain are significantly reduced[9], for it to be an open chain[10] a gas cost is still a requirement to protect the chain from DDOS attacks. This base token needs to be a market valued asset for a real cost to be associated with the executions of transactions on the utility chain.

Simple Token (ST) will be issued at a fixed supply of eight hundred million Simple Token in an EIP20 contract on Ethereum mainnet.

The first use of Simple Token on Ethereum is for a user to stake Simple Token in a staking contract on Ethereum as the value chain in order to mint a newly designed utility token on a utility chain, which

[8]This bounty needs to be put up on the utility chain for the unstaking process. The bounty cannot be put forward in a utility token as that would defeat the purpose of the bounty (it needs to be independently valued). The resolution lies in that we need the utility chain to have a base token - like Ethereum has Ether - that pays for gas consumption. The user will be required to put up a bounty in the base token of the utility chain. We will describe how the base token is obtained on the utility chain in the next section.

[9]The electricity cost required by Proof-of-Work is removed, however, the costs of Hardware Security Modules (HSM) for production use is not negligible.

[10]We consider a chain open if transactions sent to it are valid or invalid on their own right as determined by the smart contract execution, and anyone can connect a fully verifying node to the peer-to-peer network and submit transaction through this node to the network.

she can name and define specific behaviours for at the time of creating the staking and minting contracts. Importantly she can also at that point define what the conversion rate is between the value of staked assets on Ethereum and the utility token; effectively determining the denomination of the utility token.

A special case of this minting process is where a staking contract and minting contract is created where Simple Token is staked at a one-to-one ratio for a utility token on the utility chain. If there are no restrictions on who can stake additional Simple Token into the staking contract, or unstake it, and there is no special behaviour enforced on the utility token, then the utility token is freely tradable on the utility chain in the same way as its unstaked Simple Token counterpart on Ethereum.

If in addition the genesis block of the utility chain specifies that at genesis eight hundred million base tokens for the chain are awarded to this special minting contract, then this minting contract can award to users the base token, rather than a smart contract defined EIP20 token. This construction allows the creators of a utility chain to define Simple Token as the base token on that utility chain. It is important here that the minting contract requires in this case knowledge of an upper bound on the number value tokens that can be staked as the contract cannot produce new base tokens on the utility chain. Therefore the total supply of the tokens that can be staked in favor of them needs to be have an upper bound to.

The first reason why Simple Token is issued at a constant supply is to give any utility token derived from staking Simple Token EIP20 freedom to define its own monetary policy for this utility token. Having a constant Simple Token supply shields the monetary policies of different utility tokens.

Secondly, as a technical benefit, a constant supply of Simple Token makes it a suitable currency for a base token on a utility chain, as a utility chain can be created with a constant supply of base tokens in the designated minting contract.

We refer to Simple Token staked as a valuable asset to obtain a freely tradable utility token that has a one-to-one mapping to Simple Token on Ethereum as Simple Token prime [ST "] if it functions as the base token on that utility chain. Simple Token prime can then be sent between account balances on the utility chain and be charged for gas prices of transactions without acting on Ethereum mainnet. In this manner Simple Token when transferred (through the minting process) to a utility chain effectively takes the position Ether has on Ethereum mainnet as the base token that gas costs get paid in. Clearly there is no block reward for Simple Token prime beyond the transaction fees; no new Simple Token prime is awarded to validators for the act of sealing blocks.

3.6 Branded Utility Tokens

In the preceding sections we built up mechanisms for us to obtain the capability to issue a token on a utility chain that has a known value locked in a staking contract on Ethereum mainnet. We set out to have such a utility token to allow existing mainstream consumer applications to tokenize the interactions of their user base.

By tokenizing an existing application, a

companys users can earn and spend tokens that have redeemable external value, increasing the appeal of the application to new and existing users. It is worth emphasizing that this differs from existing reward programs. As an example airline miles can be redeemed within the network of services, but can only at the discretion of the airline and at a very low rate that is arbitrarily set by the airline and doesnt reflect the actual market value. With utility tokens cryptographically backed by crypto-assets on Ethereum users can redeem the tokens they own either through the member company, at a price determined solely within the discretion of the member company, or forcibly through the protocol on Ethereum into Simple Token.

Thus far we only indicated performance gains by having a more performant utility chain that has its capacity dedicated to a set of applications. We have not yet elaborated on how utility can be constructed. To define utility we require a context in which services are rendered; for Simple Token these contexts are the consumer applications.

A company can set out a monetary policy for the utility token in the staking smart contract when it designs the token for the purposes of its user base and puts up the initial stake. As a consequence the staking contract needs to be whitelisted for minting new utility tokens. Otherwise any user can stake crypto-assets, increasing the total supply of the utility tokens in circulation. A utility token where the staking contract is whitelisted is called a *branded token*, as it carries the brand of the company that put up the stake to create its branded token.

When a branded token is redeemable for its known value that has been staked on a value chain, a secondary market for trading these branded tokens is strongly suppressed. However, for existing consumer companies to tokenize the interactions of their users it could be important for legal reasons that no secondary market exists.

Additionally putting consumer interactions on a blockchain has implications for user experience concerning key management and user privacy exposing correlatable data even through pseudo-anonymous addresses.

For these reasons we look to onboard such users with an embedded wallet within these applications. The embedded wallet implies that the keys owning the branded tokens are managed by the company on behalf of the user.

Simple Token strongly believes in *your key, your coin*. To balance a good user experience with ownership of private keys a user can present to the company a *recovery-address*. The company must then provide signed receipts to the users standalone wallet asserting which managed addresses belong to the user. This allows the user to assert off-chain that her managed balances are all recoverable into the recovery-address(es) of which only her standalone wallet knows the private keys.

While this is an emergency mechanism, should the user wish to abruptly exit the managed keys, she can present the receipts signed by the company to the utility chain.

. The branded tokens will then be transferred to her recovery-address, but they will be frozen and non-transferrable. At this point the user can only return the branded tokens to the minting contract to recover the equivalent portion of the stake on the

value chain.

For a company to be a Member Company of OpenST we will require[11] that the company complies with signing recovery receipts for the users if requested.

Under normal circumstances the users of a branded token enter the economy through the Member Company itself.

3.7 Making Tokens Simple

While we acknowledge the user experience for managing keys is a hard problem, so is blockchain technology for most consumer companies not a core competency.

To this end we include as part of the Simple Token project open-source code that helps translate mapping complex user interactions within the consumer application to fundamental transactions on the chain.

We set out to provide a REST API that logically maps to the EIP20 token interface, while in the background the server handles signing with a hardware security module, transaction formulations and contract invocation on the relevant chain(s). While these form the basic requirements, we additionally provide pessimistic concurrency control to minimize the response time on the API call and provide settlement finality at the API level: the API will assume the most pessimistic (lowest) balance when returning a success or failure code to the caller, while waiting for settlement finality. This way the API call can respond in the millisecond range, even when transactions take seconds to finalize on the utility chain. Lastly the API must provide a clean mapping to smart contract events and translate them back to the user space.

In the appendix, we provide the resource paths for interacting with branded token as an application developer for a consumer application together with first specifications for developers to integrate with the APIs.

By extending Simple Token beyond the smart contracts, we not only achieve a better experience for companies developing on top of OpenST; we open a part of the stack crucial to future work on scalability and user privacy.

All companies can benefit from contributing best practices on minimizing exposure of user correlatable data when submitting transactions to the chain. If a company wants to integrate payment channels to radically increase transaction throughput and shield both its own and its users transaction details, that company can contribute such code back to OpenST for all companies to reuse, without changing the integration at the API layer to its own application. We will in a later publication detail how important technical hurdles for payment channels are mitigated within the context of Simple Token.

3.8 Nothing Lost

Lastly we briefly discuss OpenST Protocol in case of Byzantine behavior on the utility chain, or if in general it halts.

OpenST Protocol describes how to atomically act on both the utility and the value chain. However, if the utility chain halts, then all users who hold utility tokens on the halted chain can no longer initiate the claim process, as no new transactions can get processed. This makes it impossible to move

[11] This will be a legal requirement for member companies.

utility tokens into the escrow of the minting contract, which would be the minimal step required to unstake the crypto-assets on the value chain. As a result the crypto-assets would remain locked in the staking contracts on the value chain, even if the utility has disappeared.

OpenST will enable a multitude of utility chains and thus it is critical that while the validators on any cryptographically sealed chain are whitelisted, there should be nothing special about the instance of the utility chain. This is achieved when the utility chain, while stateful, can be abruptly exited and all ownerships proofs can be effected on the value chain.

In particular when the utility chain halts the light client contract on the value chain that tracks the utility chain will no longer be able to progress its acknowledged block height.

Any user can put forward a deposit to claim that the utility chain has halted at a given block height which initiates a significantly long waiting period. Chain halting should be exceptional behavior, and providing a long waiting period allows the validators of that utility chain to recover and continue the chain. Note that they have to be able to resolve the halting building onward from the latest acknowledged block height, as the light client contract on the value chain does not allow for rolling back the block height.

If the validators succeed within the waiting period to continue the chain, they will be able to report to the light client contract a higher block height resolving the claim that the chain had halted. The bounty will be forwarded to a non-OpenST foundation[12].

If the waiting period expires without progressing the light client contract to a higher block height then the contract can unlock and the both the staking and utility chains are deprecated.

When the staking contract is unlocked a reduced claim process can take place. At the height at which the utility chain has been deprecated all Merkle-proofs for ownership of branded token are considered valid - there is no need to return the branded token to the minting contract as there is only the value chain left.

Note that in order to allow for recovery-addresses to be presented a time-window is inserted to present the recovery-receipts users may have to the value chain, before the stake can be effectively claimed on the value chain.

Further note that the validators will have had to stake Simple Token Prime on the utility chain in order to be a validator. At the point where the utility chain is deprecated all validators lose this stake, as it is excluded from the recovery process and these OST remain locked in the staking contract on Ethereum mainnet.

3.9 Playing in Simple Token

To recapitulate OpenST protocol we run over an explicit example at the highest level detailing the two-phased commit and use it

[12]In case the chain successfully restarts the claimant loses his stake, but neither should the validators of the chain be rewarded for restarting the chain. Therefore Ethereum foundation can be a good candidate for rewarding such bounty too, as they provided Ethereum mainnet as impartial judge to resolve the chain halting problem.

to illustrate how a Partner Company could go about accepting Simple Token from users for services directly or to receive branded tokens.

As an explicit example we use Pepo[13] as a Founding Partner Company. For Milestone 1 (see Roadmap) we want Pepo to stake 10.000 Simple Token on Ethereum mainnet As an explicit example we use Pepo as a Founding Partner Company. For Milestone 1 (see Roadmap) we want Pepo to stake 10.000 Simple Token on Ethereum mainnet to mint the first PepoCoin [PC]. PepoCoin is set here at a 100 PC for every ST.

Step 1: Pepo starts out with 10.000 OST, a staking contract for PepoCoin on Ethereum that has no OST, and a minting contract on the utility chain; and no PC exist on the utility chain.

	Ethereum	Utility
Pepo	10.000 OST	0 PC
Esc.Stake	0 OST	
PC Stake	0 OST	
PC Esc.Mint		0 PC

Step 2: To declare its intent Pepo moves 10.000 OST into the escrow of the PepoCoin staking contract on Ethereum mainnet.

	Ethereum	Utility
Pepo	0 OST	0 PC
Esc.Stake	10.000 OST	
PC Stake	0 OST	
PC Esc.Mint		0 PC

Step 3: A proof of the intent by Pepo to mint one million PepoCoin out of ten thousand OST is proven on the utility chain.

	Ethereum	Utility
Pepo	0 OST	0 PC
Esc.Stake	10.000 OST	
PC Stake	0 OST	
PC Esc.Mint		1.000.000 PC

Step 4: A signed receipt by Pepo is first presented on Ethereum to move the stake into a locked position in the staking contract

	Ethereum	Utility
Pepo	0 OST	0 PC
Esc.Stake	0 OST	
PC Stake	10.000 OST	
PC Esc.Mint		1.000.000 PC

Step 5: The same signed receipt can now be used by Pepo to release the million PC into its account.

	Ethereum	Utility
Pepo	0 OST	1.000.000 PC
Esc.Stake	0 OST	
PC Stake	10.000 OST	
PC Esc.Mint		0 PC

Imagine Alice has 100 Simple Token on Ethereum mainnet and wants to obtain 10.000 PepoCoin. We will use this example to additionally illustrate the use of managed accounts and keys owned by Alice herself.

Step 6: We add Alice to the story. She has 100 OST on Ethereum and wants to participate in Pepo.

[13] Pepo.com is a local expertise mainstream consumer application that is engaged with Simple Token to integrate OpenST early on into Pepo.

	Ethereum	Utility
Pepo	0 OST	1.000.000 PC
Alice	100 OST	0 PC
Esc.Stake	0 OST	
PC Stake	10.000 OST	
PC Esc.Mint		0 PC

Step 7: To accept payments inOST ST without minting new PepoCoin Pepo can have a payment-escrow contract on Ethereum. Alice moves her 100OST ST into the Pepo payment-escrow contract. The escrow is locked until Alice signs for having a valid receipt from Pepo.

	Ethereum	Utility
Pepo	0 OST	1.000.000 PC
Esc.Pepo	100 OST	
Alice	0 OST	0 PC
Esc.Stake	0 OST	
PC Stake	10.000 OST	
PC Esc.Mint		0 PC

Pepo conducts know-your-customer processes (KYC) on the funds received into the payment-escrow on Ethereum from Alice. When the KYC clears, Pepo creates a managed account for Alice on the utility chain and moves the equivalent amount of PepoCoin into this account. Note that for the managed account, Alice does not have the private key.

Step 8: With the KYC for Alice cleared Pepo moves 10.000 PC into a managed account for Alice.

	Ethereum	Utility
Pepo	0 OST	990.000 PC
Esc.Pepo	100 OST	
Man.Alice		10.000 PC
Alice	0 OST	0 PC
Esc.Stake	0 OST	
PC Stake	10.000 OST	
PC Esc.Mint		0 PC

Pepo sends Alice off-chain a signed receipt that asserts that the funds in the managed account for Alice are in fact Alices (identified by her account on Ethereum). Alice can verify that on the utility chain the funds have moved into a managed account on her behalf. She can sign on Ethereum that she accepts the receipt and this releases the OST to Pepo.

Step 9: Alice receives the receipt off-chain from Pepo and verifies that the appropriate funds are managed on her behalf within Pepo. She signs the receipt on Ethereum and this releases the funds to Pepo.

	Ethereum	Utility
Pepo	100 OST	990.000 PC
Esc.Pepo	0 OST	
Man.Alice		10.000 PC
Alice	0 OST	0 PC
Esc.Stake	0 OST	
PC Stake	10.000 OST	
PC Esc.Mint		0 PC

Within the Pepo application Pepo manages the private keys of the users. Alice however possesses the signed receipt she received from Pepo that asserts which accounts are managed on her behalf. She presents this on Ethereum to move the OST to Pepo. Pepo will continue to present her

with receipts for all managed accounts on her behalf and Alice can continue to verify that they are correct. Should Alice wish to hard-exit from PepoCoin, she can use the same receipts, but present them on the utility chain as we described before.

Step 10: Alice preserves her sovereignty because she can at any point present the receipts from Pepo on the utility chain and move her managed funds on the utility chain into an account controlled by her.

	Ethereum	Utility
Pepo	100 OST	990.000 PC
Esc.Pepo	0 OST	
Man.Alice		10.000 PC
Alice	0 OST	0 PC
Esc.Stake	0 OST	
PC Stake	10.000 OST	
PC Esc.Mint		0 PC

While this hard-exit scenario should practically never occur, as it is cheaper and more efficient to sell out of her share of PepoCoin, it is crucial that the OpenST provides an exit path where Alice can act independently, without anyones permission.

Step 11: When Alice has hard-exited her PepoCoins, she can no longer use them within the Pepo application, but she can use the unstaking process to recover her share of Simple Token on Ethereum. The end-result of which is given below.

	Ethereum	Utility
Pepo	100 OST	990.000 PC
Esc.Pepo	0 OST	
Man.Alice		0 PC
Alice	100 OST	0 PC
Esc.Stake	0 OST	
PC Stake	9.900 OST	
PC Esc.Mint		0 PC

We note that this example ignored loss due to transaction fees on Ethereum, the utility chain, and commissions, fees and taxes to Pepo or applicable authorities. The example is illustrative only.

4 OpenST Platform

The OpenST Protocol described in the first chapter allows for value on Ethereum mainnet (the value chain) to be staked in favor of a branded token on a utility chain. An important balance to strike for Simple Token is to bridge a good end-user experience with actual cryptographic value stored on Ethereum. We require this bridge to be cryptographically secured. Even if we want to be realistic and acknowledge that the majority of end-users may never want to manage their own private keys, it is important that any user at any point can decide to take control over her private keys.

Likewise, while we present utility chains as a more performant and dedicated substrate on which applications can settle accounts, OpenST ensures at a protocol level that no trust needs to be placed in the utility chains or their validators, as all value can, if needed, be recovered on Ethereum mainnet, while at the same time achieving transaction throughput and user privacy us-

ing functional sharding, account scrambling and payment channels.

4.1 Network of Networks

An end-user can obtain or return branded tokens through the Partner Company that hosts the consumer application for Simple Token. As OST prime Simple Token prime is present on the utility chain as its base token, it is also equivalent to Simple Token EIP20 on Ethereum mainnet. The transfer of branded token for Simple Token transaction can happen therefore on the utility chain (in OST prime Simple Token prime), making it instant, and removing the need to stake or unstake OST in the staking contract of the branded tokens.

If we consider multiple utility chains, they each have a base token OST prime Simple Token prime, Simple Token double prime, etc. each equivalent to Simple Token on Ethereum mainnet, and by transition equivalent to each other.

Existing inter-chain exchange or transfer protocols can be applied to allow the efficient value transfer of Simple Token (prime and double prime) between different utility chains; again without touching on Ethereum mainnet, which keeps the staking contracts for branded tokens on different chains invariant.

A user who holds Simple Token (with Simple Token Wallet) can then seamlessly earn and spend across different branded tokens - without needing to be aware she is moving value across chains in the background.

As an open protocol OpenST describes how different utility chains can interface with each other and with a value chain. In the case of the OpenST Platform Simple Token EIP20 is defined on Ethereum mainnet from where it allows utility chains to shard for specific applications with branded tokens. Value can flow between these utility chains further catalysing the adoption of existing protocols that solve for various needs.

It is crucial to point out that there is no preferred entity hosting these utility chains. While a given utility chain will have dedicated validators, the OpenST Platform itself spans across chains and any utility chain, hosted by any set of validators, can interact with the other utility chains on open and equal footing. Every utility chain is always guaranteed to be fully redeemable on Ethereum mainnet and as such always replaceable.

4.2 Simple Token Architecture

As OpenST is an open network of utility chains, each hosting one or more branded tokens with value staked on Ethereum mainnet, it is paramount that each utility chain (and Ethereum) have a unique chain identifier associated with it (derived from the genesis block that started that utility chain for fork-accountable chains). All transactions need to enforce that they are only valid on the intended utility chain.

Further it is desirable that a utility chain can be restricted to not accept the creation of new smart contract code when a transaction deploys code to a new address. Ethereum mainnet functions as a general computation substrate that allows anyone to deploy any code contract. We note that

this is not a requirement, but a preference to be decided by the validators of the utility chain.

Simple Token prime is present on the utility chain as the base token to account for all gas usage. In this sense there is no requirement to make the utility chain *application-specific*. However, while a cost will be accounted for running foreign contracts on the utility chain, it eats into the capacity of the chain that could otherwise be used for the branded tokens within the consumer applications.

We explained that, in the case of branded tokens, by default users start out with accounts managed by the respective branding Partner Company[14]. As such OST Partner Companies are responsible for funding the balances of the managed accounts (just-in-time) with Simple Token prime[15]. This way there are no transaction costs charged to the user when acting within the consumer application - but any user transferring Simple Token prime (or exiting her branded tokens to her own keys) pays for this in Simple Token prime.

OpenST is agnostic to the blockchain implementation used to deploy an instance of the OpenST contracts. As the OpenST Platform originates from OSTSimple Token EIP20 on Ethereum mainnet, Ethereum mainnet forms the pivot point for all utility chains within the OpenST platform. To this end we expect to work with Ethereum-based implementations, where the only condition on the consensus engine is that a light client tracking contract can be implemented on Ethereum mainnet for the utility chains specific node implementation and consensus mechanism.

OpenST will therefore aim to not depend on a single node implementation and will want to work with and contribute back to existing cryptographically sealed Ethereum blockchains. These can have a consensus engine that runs on Proof-of-Authority, PBFT, or Proof-of-Stake Tendermint - or other Byzantine fault-tolerant (BFT) cryptographically sealed consensus engines[16].

4.3 User Privacy Considerations

As OpenST looks to connect mainstream consumers with blockchain records user privacy is paramount. To start of utility chains only store pseudo-anonymous addresses and their balances. Unlike Ethereum mainnet, because utility chains themselves only serve as an application substrate they can have a finite lifetime and user balances can over longer timespans migrate over differ-

[14]While users start by default with managed keys, OpenST requires they be provided - upon request - with signed receipts that can be submitted to the utility chain to claim branded token balances into recovery-addresses for which they control the private keys.

[15]The Simple Token prime balances of managed accounts do not get transferred to the recovery-address, as this is Simple Token prime owned by the Partner Company.

[16]We note that Proof-of-Authority (PoA) is not Byzantine fault-tolerant; however a light client tracking contract for PoA can be implemented on Ethereum mainnet and this strengthens the security guarantees a PoA-utility chain would have, compared to running independently. This security down-grade can be balanced against the increased robustness and throughput of a PoA consensus engine. Note that a light client tracking contract on Ethereum for PoA utility chain should not naively be implemented, but improvements can be envisioned.

ent chains, allowing for possible and eventual erasure of these pseudo-anonymous addresses.

To strengthen privacy considerations for both the member companies and their users the use of payment channels helps keep the details of microtransactions off the utility chain, only regularly settling the netting on the utility chain.

Finally there may still be correlatable data from pseudo-anonymous balances extractable from the utility chain. As users for their balances of branded tokens within a consumer application are managed by the member company, user balances can be deterministically - but derived from off-chain information shared through receipts with the user - be broken up; forcefully breaking correlations between transfers on the utility chain.