ZXELAR:
Connecting Applications with Blockchain Ecosystems

Abstract

Multiple blockchain ecosystems are emerging that provide unique and distinct features attractive to users and application developers. However, communication across the ecosystems is very sparse and fragmented. To enable applications to communicate across blockchain ecosystems frictionlessly, we propose ZXELAR. ZXELARstack provides a decentralized network, protocols, tools, and APIs that allow simple cross-chain communication. ZXELARprotocol suite consists of cross-border routing and transfer protocols. A decentralized open network of validators powers the network; anyone can join, use it, and participate. Byzantine consensus, cryptography, and incentive mechanisms are designed to achieve high safety and liveness requirements unique for cross-chain requests.

1 Introduction

Blockchain systems are quickly gaining popularity and attract new use cases for asset tokenization, decentralized finance, and other distributed applications. Several major platforms such as Ethereum, Monero, EOS, Cardano, Terra, Cosmos, Avalanche, Algorand, Near, Celo, and Polkadot offer distinct features and development environments that make them attractive for different applications, use-cases, and end-users [5, 12, 4, 22, 21, 24, 25, 20, 6, 15, 26]. However, the useful features of each new platform are currently offered to less than 1% of the ecosystem's users, namely the holders of the native token on that platform. Can we allow platform developers to plug in their blockchains to other ecosystems easily? Can we enable application builders to build on the best platform for their needs while still communicating across multiple blockchain ecosystems? Can we allow users to interact with any application on any blockchain directly from their wallets?

To bridge the blockchain ecosystems and enable applications to communicate frictionlessly across them, we propose ZXELARnetwork.
Validators collectively run a byzantine consensus protocol and run the protocols facilitating cross-chain requests. Anyone can join the network, participate, and use it. The underlying network is optimized for high safety and liveness requirements unique for cross-chain requests. ZXELAR network also includes a protocol suite and APIs. The core protocols are:

- Cross-Chain Gateway Protocol (CGP). This protocol is analogous to Border Gateway Protocol on the Internet. This protocol is used to connect multiple autonomous blockchain ecosystems and is responsible for routing across them. Blockchains do not need to "speak any custom language", their platform developers do not need to make any custom changes on their chains, and their chains can be plugged into the global network easily.

- Cross-Chain Transfer Protocol (CTP). This protocol is analogous to application-level protocols File Transfer, Hypertext Transfer Protocols on the Internet. It is an application-level protocol stack that sits on top of routing protocols (such as CGP and other routing technologies). Application developers can connect their dapps on any chain to perform cross-chain requests. Users can use the CTP protocol to interact with applications on any chain using simple API calls analogous to HTTP GET/POST requests. Developers can lock, unlock, and transfer assets between any two addresses across any blockchain platforms, execute cross-chain application triggers (e.g., an dapps on chain A, can update

its state if some other application on chain B satisfies some search criteria (interest rate $> X$)), and perform general cross-chain requests between apps across chains (a smart contract on chain A can call to update a state of a smart contract on chain B). This protocol enables the composability of programs across blockchain ecosystems.

ZXELARnetwork offers the following advantages:

- *For blockchain platform builders*: Ability to easily plug-in their blockchains to all other blockchain ecosystems. Only a threshold account needs to be set up on the chain to plug into the network.
- *For dapps builders*: Application builders can host their dapps anywhere, lock, unlock, transfer assets, and communicate with applications on any other chain via CTP API.
- *For users*: Users can interact with all applications across the ecosystem directly from their wallets.

A Platform for Builders. Finally, ZXELARnetwork is a platform for developers and a global community. Its governance model is open to anyone. Developers can propose new integration points, routing, and application-level protocols, and the users can decide whether to adopt them by voting on the proposals and, if approved, validators will adopt the changes.

1.1 Existing Interoperability Solutions

Previous attempts to solve interoperability across blockchains fall in one of four categories: centralized exchanges, interoperable ecosystems, wrapped assets, and token bridges. We briefly summarize these approaches below.

Centralized Systems. Today centralized systems are the only truly scalable solutions to interoperability needs for the ecosystem. They can list any asset or onboard any platform relatively easily. However, centralized systems are known to have various security issues and are not good enough to power the emerging decentralized financial system that requires robust security, transparency, and open governance. On their own they cannot power the decentralized applications as they grow.

Interoperability Hubs. Projects such as Cosmos, Polkadot, Ava labs address interoperability between *sidechains* native to their ecosystems using custom inter-chain communication protocols [24, 26, 25]. For example, one can spin-up a sidechain (a Cosmos Zone) that can communicate with the Cosmos Hub. The sidechain must be based on the Tendermint consensus and speak the protocol natively understood by the Cosmos Hub. Connections to other blockchains and ecosystems that speak different languages is left to external technologies.

Pairwise Bridges. Wrapped assets (e.g., wrapped Bitcoins) try to fill-in the missing cross-chain interoperability gap in the ecosystem. One example is tBTC [10], which is a custom protocol where a clever combination of smart contracts and collateral is used to secure the transfers. These solutions require substantial engineering efforts to build – for each chain pair, developers must build a new smart contract on destination chain that parses state proofs from origin chain (similar to how each side-chain could, in principle, parse state of other chains). Only a handful of bridges have been deployed using this approach. These approaches do not scale when one of the underlying blockchains wants to upgrade its consensus rules or transaction format. This is because all smart contracts that depend upon the state of these chains would need to be upgraded. One must also set up validators and require them to lock up different assets in order to overcollateralize any asset transfer, which limits the economic efficiency of such transfers.

We have also seen a few other single-purpose bridges by platform developers that rewrite state-transition logic in smart contracts to bridge to other ecosystems [1, 7]. They suffer from multiple scalability issues, do not allow the ecosystem to scale uniformly, and introduce additional dependencies for applications. For instance, if one platform changes, then all smart contracts on all bridges will need to be upgraded. This

will ultimately put the ecosystem in a gridlock where no-one will be able to upgrade. Finally, if one single-purpose bridge connects platforms A and B, and a second single-purpose bridge connects B and C, it does not mean that applications on A will be able to talk to applications on C. One might need to create another single-purpose bridge or rewire application logic.

Other attempts to tackle interoperability include federated oracles (e.g., Ren [8]), and application specific interoperable blockchains [11, 9].

To summarize, existing solutions for interoperability require heavy engineering work from both platform developers and application builders that must understand different communication protocols to communicate across every pair of ecosystems. And so, interoperability is virtually non-existing in today's blockchain space. At the end of the day, platform developers want to focus on building platforms and optimize them for their use-cases and be able to plug in to other blockchains easily. And application developers want to build dapps on the best platforms for their needs while still leveraging users, liquidity and communicate with other dapps on other chains.

2 The Quest for Scalable Cross-Chain Communication

At the core, cross-chain communication requires that heterogeneous networks find the ability to communicate using the same language. To solve this, we explain the ZXELARprotocol suite, describe its high-level properties,and explain how these properties address the core of scalable cross-chain communication.

1. *"Plug-and-play" integration.* Blockchain platform builders should not be required to perform heavy engineering or integration work to speak some "custom language" to support cross-chain. The cross-chain protocol should be able to plug in any existing or new blockchain frictionlessly. New assets should be added with minimal effort.

2. *Cross-chain routing.* Functions such as the discovery of network addresses, routing paths, and networks are at the core of the Internet and facilitated by BGP and other routing protocols. Similarly, to facilitate communication across blockchain ecosystems, we need to support the discovery of addresses across them, applications, and routing.

3. *Upgradability support.* If one of the blockchain ecosystems changes, it should not affect the interoperability of other blockchains. The system needs to recognize updates, and minimal effort should be required to support them (i.e., no "state transition logic" should be rewritten, and applications should not break).

4. *Uniform language for applications.* Applications need a simple protocol to lock, unlock, transfer, and communicate with other applications no matter which chain they reside on. This protocol must be chain-agnostic and support simple calls, analogous to HTTP/HTTPS protocols that allow users and browsers to communicate with any web-server. As more networks and assets join the lower-level routing protocols, applications should be able to use them for communications without rewriting their software stacks.

Next, we summarize the security requirements that these protocols must meet.

1. *Decentralized trust.* The network and protocols must be decentralized, open, and allow everyone to participate fairly.

2. *High safety.* The system must satisfy high safety guarantees. The system needs to preserve the safety of assets and the state as the cross-chain network processes it.

3. *High liveness.* The system must satisfy high liveness guarantees to support applications leveraging its cross-chain features.

Satisfying a subset of these properties is easy. For instance, one can create a federated multisig account with their friends and lock/unlock assets on corresponding chains. Such systems are inherently vulnerable to collusion and censorship attacks and lack proper incentives for the validators to protect them. Creating a decentralized network and protocol suite where anyone can participate while being correctly incentivized can enable frictionless cross-chain communication, but solving it is a hard problem that requires a careful combination of consensus, cryptographic, and mechanism design protocols.

3 ZXELARNetwork

ZXELAR*network provides a uniform solution to cross-chain communication that meets the needs of both plat -form developers – no integration work is required from them, and application builders – one simple protocol and API to access global liquidity and communicate with the entire ecosystem.*

ZXELARnetwork consists of a decentralized network which bridges blockchain ecosystems that speak diff er-ent languages and a protocol suite with APIs on top, making it easy for applications to perform cross-chain requests. The network connects existing stand-alone blockchains such as Bitcoin, Stellar, Terra, Algorand,and interoperability hubs such as solutions like Cosmos, Avalanche, Ethereum, and Polkadot. Our mission is to enable application developers to build such apps easier using a universal protocol and API without rolling out their proprietary cross-chain protocols underneath or rewriting applications as new bridges are developed. Towards this, we designed a protocol suite that includes Cross-Chain Gateway Protocol (see Section 6) and Cross-Chain Transfer Protocol (see Section 7).

A core component of the network are the underlying decentralized protocols. Validators collectively maintain the ZXELARnetwork and run the nodes that secure the ZXELARblockchain. They are elected through a delegation process by the users. Validators receive voting power pro-rata according to the stake delegated to them. The validators reach consensus on the state of multiple blockchains that the platform is connected to. The blockchain is responsible for maintaining and running the cross-chain routing and transfer protocols. Governance rules allow network participants to enact protocol decisions such as which blockchains to bridge and which assets to support.

ZXELARblockchain follows a Delegated Proof-of-Stake (DPoS) model similar to Cosmos Hub. Users elect validators who must bond their stake to participate in the consensus and maintain high-quality service. The DPoS model allows maintenance of large decentralized validator set and robust incentives to guarantee that the validators are responsible for maintaining bridges and shares of cryptographic threshold schemes. As part of consensus, validators run light-client software of other blockchains, allowing them to verify the state of other blockchains. The validators report these states to the ZXELARblockchain, and once enough of them report, the state of Bitcoin, Ethereum, and other chains is recorded on ZXELAR.

Subsequently, the ZXELARbase layer is aware of the state of external blockchains at any point in time,creating the " incoming bridges" from other blockchains. The validators collectively maintain *threshold sig-nature accounts* on other blockchains (e.g., 80% of validators must approve and co-sign any transaction out of it), which allows them to lock and unlock assets and state across chains and to post state on other blockchains, the "outgoing bridges." Altogether, one can view the ZXELARnetwork as a *decentralized cross-chain read/write oracle*.

The remainder of the document describes preliminaries and building blocks behind the network (Section 4), some technical details of the network (Section 5), cross-chain gateway protocol (Section 6), and cross-chain transfer protocol (Section 7).

4 Preliminaries

4.1 Notation and Assumptions

Let V^r denote the set of ZXELARvalidators at round R. Each validator has a *weight*, a number in (0, 1]denoting the voting power of that particular validator. The weights of all validators add up to 1.
A validator is *correct* if she runs a node that is consistent with the rules of the ZXELARprotocol. To fi nalize blocks, or to sign cross-chain requests, ZXELARrequires correct validators of total weight $> F$.
We call the parameter $F \in [0.5, 1]$ the *protocol threshold*.

ZXELARcan be based on an *instant finality Delegated-Proof-of-Stake* blockchain.
The validators run *Byzan-tine Fault Tolerant (BFT) consensus* at each round i to finalize the i_{th} block.
Once the i_{th} block is finalized,new BFT consensus is run to finalize the $i + 1_{th}$ block, and so on.
The validators are elected through stake delegation.
A user with some stake may elect to run a validator node, or delegate their voting power (stake)
to an existing validator, who then votes on their behalf. The validator set can be updated, validators join/ leave the set, and users delegate/undelegate their voting power.

Different blockchains work under different network assumptions. *Synchronous communication* means that there is a fixed upper bound Δ on the time messages take to be delivered, where Δ is known and can be built into the protocol. *Asynchronous communication* means that messages may take arbitrarily long to be delivered, and it is known that BFT protocols cannot be built for asynchronous networks even in the presence of just one malicious validator. A realistic compromise between synchrony and asynchrony is the assumption of *partially synchronous communication*. The network may be completely asynchronous until some unknown global stabilization time (GST), but after GST communication becomes synchronous with a known upper bound Δ [18].

Typical blockchains work under the assumption of $> F$ correct validators. For synchronous networks $F = 1/2$ is typically set, but for the weaker assumption of a partially synchronous network $F = 2/3$. Bitcoin, its forks, and the current Proof-of-Work version of Ethereum only work assuming synchrony. Others like Algorand and Cosmos only require partial synchrony. When connecting chains through ZXELAR, the connection works assuming the strongest network assumptions out of these chains,
which is synchrony in the case of connecting Bitcoin and Cosmos, for instance.
The ZXELARblockchain itself works in a partially synchronous setting and thus requires $F = 2/3$,
but it is possible to improve the threshold requirement by
assuming that other existing blockchains are secure and leveraging their security.

4.2 Cryptographic Preliminaries

Digital Signatures. A *digital signature scheme* is a tuple of algorithms $(Keygen, Sign, Verify)$. $Keygen$ outputs a pair of keys (PK, SK). Only the owner of SK can sign messages, but anyone can verify the signatures given the public key PK. Most blockchain systems today use one of the standard signature schemes such as ECDSA, Ed25519, or a few of their variants [2, 3].

Threshold Signatures. A *threshold signature scheme* enables a group of n parties to split a secret key for a signature scheme in such a way that any subset of $t + 1$ or more parties can collaborate to produce a signature, but no subset of t or fewer parties can produce a signature or even learn any information about the secret key. The signatures produced by the threshold protocols for ECDSA and EdDSA look identical to the signatures produced by the stand-alone algorithms.

A threshold signature scheme replaces the *Keygen* and *Sign* algorithms for an ordinary signature scheme with distributed n-party protocols $T.Keygen$, $T.Sign$. These protocols typically require both a public broadcast channel and private pairwise channels among the parties, and they typically involve several rounds of communication. After successful completion of $T.Keygen$ each user holds a share s_i of a secret key SK and the corresponding public key PK. The $T.Sign$ protocol allows these parties to produce a signature for a

given message that is valid under public key PK. This signature can be verified by anyone using the *Verify* algorithm of the original signature scheme.

4.3 Properties of Threshold Signatures

There are several properties a threshold scheme might have that are especially desirable for decentralized networks:

Security against a dishonest majority. Some threshold schemes have the restriction that they are secure only when a majority of the n parties are honest. Thus, the threshold parameter t must be smaller than $n/2$ [16]. This restriction is typically accompanied by the fact that $2t + 1$ honest parties are needed to sign, even though only $t+1$ corrupted parties can collude to recover the secret key. Schemes that do not suffer from this restriction are said to be *secure against a dishonest majority*.

As discussed later in Section 5.2, cross-chain platforms must maximize the safety of their networks and be able to tolerate as many corrupted parties as possible. Thus, schemes that can tolerate dishonest majority are necessary.

Pre-signatures, non-interactive online signing. In an effort to reduce the burden of communication upon the parties to sign a message, several recent protocols have identified a significant portion of the work for a signature that can be done "offline", before the message to sign is known [19, 14]. The output of this offline phase is called a *pre-signature*. The production of pre-signatures is viewed as a separate protocol $T.Presign$ distinct from $T.Keygen$ and $T.Sign$. The outputs of the pre-signature protocol must be kept private by the parties until they use them at the signing phase. Later, when the message to sign becomes known, only a small amount of additional "online" work remains to be done in $T.Sign$ in order to complete the signature.

The online $T.Sign$ phase does not require any communication among the parties. Each party simply does a local computation on the message and pre-signature and then announces her share s_i of the signature. (Once public, these signature shares $s_1, \ldots, s_{t+1}$ are easily combined by anyone to reveal the actual signature s.) This property is called *non-interactive online signing*.

Robustness. Threshold schemes guarantee only that a subset of malicious parties cannot sign messages or learn the secret key. This guarantee does not, however, preclude the possibility that bad actors can block everyone else from producing keys or signatures. In some schemes, malicious behaviour by even a single party can cause $T.Keygen$ or $T.Sign$ to abort with no useful output. The only recourse is to restart the protocol, possibly with different parties.

Instead, for decentralized networks, we want $T.Keygen$ and $T.Sign$ to succeed if at least $t + 1$ of the parties are honest, even if some malicious parties send malformed messages or drop messages in the protocols. This property is called *robustness*.

Fault attribution. The ability to identify bad actors in $T.Keygen$ or $T.Sign$ is called *fault attribution*. Without fault attribution it is difficult to reliably exclude or punish bad actors, in which case the costs imposed by bad actors must be borne by everyone. This property is also important for decentralized networks where malicious behavior should be identifiable and economically disincentivized via slashing rules.

Security in concurrent settings. The signature scheme needs to be secure in a concurrent setting, where multiple instances of the keygen and signing algorithms can be involved in parallel. (Drijvers et al. [17] for instance, showed an attack against Schnorr multisignature schemes in these settings). There are versions of both ECDSA and Schnorr schemes that satisfy these properties [14, 23].

ECDSA and EdDSA are by far the most widely deployed signature schemes in the blockchain space. As such, threshold versions of both schemes have been the focus of a recent resurgence in research and development. Readers interested in state-of-the-art can refer to [23, 14, 19] and a recent survey paper [13].

5 ZXELARNetwork

5.1 Designing an Open Cross-Chain Network

The bridges that ZXELARnetwork maintains are backed up by threshold accounts such that (almost) all valida-tors must collectively authorize any cross-chain request.
Designing a network where anyone can participate
to secure these bridges requires meeting the following technical requirements:

- *Open membership.* Any user should be able to become a validator (following the rules of the network).
- *Updates to membership.* When a validator leaves the system honestly, their key needs to be revoked appropriately.
- *Incentives and slashing.* Malicious validators should be identifiable and their actions must be identified and addressed by the protocol.
- *Consensus.* Threshold schemes on their own are defined as stand-alone protocols. To propagate messages between nodes we need both broadcast and point-to-point private channels. Moreover, validators need to agree on the latest state of each invocation of threshold schemes since they often have multiple round of interactions.
- *Key-management.* Just as ordinary validators in any PoS system must carefully guard their keys, so too must ZXELARvalidators guard their threshold shares. Keys need to be rotated,
  split between online
  and offline parts, etc.

ZXELARstarts with Delegated Proof-of-Stake model,
where the community elects a set of validators to run the consensus.
Note that standard threshold schemes treat every player identically and have no notion of "weight" in the consensus. Hence, the network must adapt them to take validators' weight into account. A simple approach is to assign multiple threshold shares to larger validators. Outlined below are three basic functions that validators collectively perform.

- *Threshold Key Generation.* Existing threshold key generation algorithms for standard blockchain signature schemes (ECDSA, Ed25519) are interactive protocols between multiple participants (see Section 4).
  A special transaction on the ZXELARnetwork instructs the validators to commence execution
  of this stateful protocol. Each validator runs a threshold daemon process that is responsible for the secure keeping of the secret state. For each phase of the protocol:
  1. A validator keeps the state of the protocol in its local memory.
  2. It calls the secret daemon to generate the messages as per the protocol description for other validators.
  3. It propagates the messages either via the broadcast or via the private channels to other validators.
  4. Each validator executes state transition functions to update its state, proceed to the next phase of the protocol, and repeat the above steps.

  At the end of the protocol, a threshold public key is generated on the ZXELARchain,
  and it can be displayed back to the user (e.g., for deposits)
  or to the application that generated the initial request.

  - *Threshold Signing.* Signing requests on the ZXELARnetwork are processed similarly to the key-generation requests. These are invoked, for instance, when a user wants to withdraw an asset from one of the chains. These are interactive protocols, and state transition between the rounds is triggered as a function of the messages propagated via the ZXELARblockchain view and every validator's local memory
- *Handling Validator Membership Changes.* The validator set needs to be rotated periodically to allow for new stakeholders to join the set. Upon a validator set update, we need to update the threshold key to be shared across the new set. Thus if we allowed anyone to join at any time, we would have to update the threshold key very frequently. To prevent this, we rotate validators every T blocks. Within intervals of T rounds, the set V^R and the threshold key are fixed. At every round that is an integral multiple of the parameter T, we update the validator set as follows:

1. At any round R, the ZXELARstate keeps track of the current validator set V^R. $V^{R+1} = V^R$ unless $R+1$ is a multiple of T.
2. During rounds $((i-1)T, iT]$, users post bonding/unbonding messages.
3. At the end of round iT, these messages are applied to V^{iT-1} to get V^{iT}.

• *Threshold Key Generation and Signing in the Presence of Rotating Validators.*
ZXELARblockchain may
issue a request for a new key or a threshold signature at round R. The signing process takes longer than one round, and we don't want to slow down consensus, so we request that the signature is produced before round $R+10$ starts. In particular, validators start round $R+10$ only after seeing a certificate for round $R+9$ and a signature for each keygen/signature request issued at round R. The outcome of all round R requests must be included in block $R+11$. In other words, a round R block proposal that does not contain the outcomes from a round $R-11$ is considered invalid, and validators don't vote on it. To ensure that all threshold messages are signed before a validator set update,
ZXELARdoes not
issue any threshold requests during a round equal to $-1, -2, \ldots, -9$ modulo T.

5.2 Network Security

The security of blockchain systems relies on various cryptographic and game theoretic protocols, as well as the decentralization of the network. For instance, in proof-of-stake blockchains, without the proper incentives validators may collude and rewrite the history, stealing other users' funds in the process. In proof-of-work networks, without sufficient decentralization, it is quite easy to create long forks and double spend, as the multiple attacks on Bitcoin Gold and Ethereum Classic have proven.

Most of the research on blockchain security has focused on sovereign chains. But once chains interoperate, new attack vectors have to be considered. For instance, assume that Ethereum talks to a small blockchain X through a direct bridge controlled by two smart contracts, one on Ethereum and one on X. Besides the engineering challenges we summarized in Section 1.1, one must decide what happens when the trust assumptions of X are violated. In this case, if ETH has moved to X, the validators of X may collude to forge a history of X where they hold all the ETH, post the forged consensus proofs on Ethereum and steal the ETH. The situation is even worse when X is connected with multiple other chains through direct bridges, where if X forks the effects propagate through every bridge. Setting up recovery governance guidelines for each pairwise bridge is an overwhelming task for any individual project.

ZXELARnetwork addresses the security concerns using the following mechanisms:

• *Maximum Safety.* ZXELARsets the safety threshold to 90%,
meaning that almost all validators will need
to collude to withdraw any funds that are locked by its network or forge state proofs[1]. In practice, it has been observed that PoS validators have very high up-time (close to 100%), assuming they are properly incentivized. Hence,
ZXELARnetwork will produce blocks even despite this high threshold.
However, in the rare case that something goes wrong and the network stalls, the network needs robust fall-back mechanisms to reboot the system described next.

• *Maximum Decentralization.* Since the network uses threshold signature schemes, the number of validators can be as large as possible. The network is not bounded by the number of validators we can support, transaction limits or fees that would arise from using, for instance, multi-signatures on different chains where the complexity (and fees) increase linearly with the number of validators.[2]

• *Robust Fall-back Mechanisms.* The first question that must be addressed in a network with high safety thresholds as above is what happens when the network itself stalls. Suppose ZXELARnetwork itself stalls. Can we have a fall-back mechanism that would allow users to recover their funds? To address any potential stall of the ZXELARnetwork itself, each threshold bridge account on a blockchain X that the ZXELARvalidators collectively control has an "emergency unlock key". This key can be shared

[1] The final parameter that will be chosen for the network deployment may be adjusted.
[2] For some blockchains, multi-signatures offer a reasonable alternative where gas fess are small and supported message formats are appropriate. But they do not scale for two of the most largest platforms like Bitcoin and Ethereum.

across thousands of parties and may even be a custom key for blockchain X that is shared across the community of that chain. Hence, if ZXELARnetwork stalls, this key will act as a fall-back and enable recovery of the assets (see below for more details).

- *Maximum Decentralization of Fall-Back Mechanisms.* This fall-back mechanism includes a secondary *recovery set* of users, in which just anyone can participate without any cost. These users do not need to be online, run nodes, or coordinate with each other. They are only "called on duty" if ZXELARnetwork stalls and cannot recover. The network's security is enhanced by a very high threshold on the primary validator set and a maximally decentralized secondary recovery set.
- *Shared Governance.* A common protocol governs the ZXELARnetwork. Collectively, the users can vote on which chain should be supported through its network. The network will also allocate a pool of funds that can be used to reimburse users in case of unexpected emergencies, controlled via the governance protocols as well.

Various security mechanisms are discussed below.

Fall-Back Mechanisms. When ZXELARstalls due to the high threshold, an "emergency unlock key" takes control of the network. There are multiple ways to instantiate this unlock key, and certain chains/applications may opt to utilize a different variation for the "recovery set" or opt-out completely:[3]

- *Option a.* Share the key across foundations of blockchain projects and reputable people in the community.
- *Option b.* Share the across parties elected through the delegated PoS mechanism.
- *Option c.* For accounts managing assets and information for chain/application X, share a custom key across the stakeholders/validators of X. Assuming X has governance mechanisms in place, the same governance mechanisms can be applied to determine a course of action if ZXELARstalls.

Now, given the recovery users' identities and their public keys, a simple protocol generates shares of the recovery key that no-one knows. Moreover, the users of recovery set do not need to be online until called to recover via the governance mechanisms. Following the standard distributed key-generation protocols, each ZXELARvalidator shares a random value. The recovery secret key is generated by summing up these values. Instead of doing the summations in the clear, all shares are encrypted under the public keys of the recovery users and then added up homomorphically (this assumes additively homomorphic encryption and an additional layer of zero-knowledge, both of which are easily obtainable). The result of this protocol is a recovery public key RPK and potentially thousands of encryptions (under the public keys of the recovery users) of the shares of the corresponding secret key $Enc_i(s_i)$ that are distributed to their owners (e.g., posted on chain).
ZXELARbridge contracts include an option to recover funds using RPK under certain conditions.Finally, it is also possible to update this recovery key and even change the set of users holding its shares without requiring any work from the participating shareholders.

If chain X that is connected to ZXELARbreaks, there are a couple options:

- Impose limits on the USD value of assets that can be moved in/out of X on any single day. Thus a malicious chain X can only steal a small fraction of all assets that are bridged to it before ZXEL AR validators detect this, and the governance mechanisms from the following bullets kick in.
- The ZXELARgovernance module can be used to vote on what happens in those situations. For instance, if there is a benign bug and the community restarts X, ZXELARgovernance can determine to restart the connection from where it left off.
- If ETH had moved to X, a custom Ethereum recovery key can determine what happens to the ETH assets.

[3]The final deployment on the ZXELARnetwork will be finalized closer to the network launch.

6 Cross-Chain Gateway Protocol (CGP)

In this section, we explain the cross-chain gateway protocol and routing mechanisms on two core examples common between many applications' needs:

State synchronization (Section 6.2). Post information about the state of a source blockchain S into the state of a destination blockchain D.

(For example, post a Bitcoin block header to the Ethereum blockchain.)

Asset transfer (Section 6.3). Transfer a digital asset from S to D and back again.

(For example, transfer bitcoins from the Bitcoin blockchain to the Ethereum blockchain, and then back to the Bitcoin blockchain.)

For simplicity we assume that chain D has at least minimal support for smart contracts but S can be any blockchain whatsoever.

6.1 Accounts on other chains

To bridge different chains, threshold accounts are created on each chain that control the flow of value and information across them. For chain $Chain$, denote the account by $Chain_{\mathsf{ZXELAR}}$.

Bitcoin account. For Bitcoin and other non-smart contract chains ZXELARvalidators create a threshold ECDSA key as per section 5.1. This key controls the ECDSA account on Bitcoin, and is the destination address where users send deposits. Personalized threshold keys may be created per user request. The key may be updated periodically,
and the latest key and personalized keys can be found by querying an ZXELAR node.

Threshold bridge account on chains with smart contracts. Denote the chain by SC. the validators create a threshold ECDSA or ED25519 key as per section 5.1, depending on which key type the chain supports. We denote this key by PK_{ZXELAR}, when there is no ambiguity as to which chain we are referring to. This key controls a smart contract account on SC, denoted by SC_{ZXELAR}, and any application on SC can query $SC_{\mathsf{ZXELAR}}$to learn the PK address of that key. This way, any SC application can recognize messages signed by SK_{ZXELAR}. The protocol also needs to account for rotating values of PK_{ZXELAR}. This happens as follows:

1. Initialize SC_{ZXELAR}on SC. It stores PK_{ZXELAR}as part of its state, which is initialized as its genesis value on ZXELAR. SC_{ZXELAR}also includes rules for updating the PK.
2. To update PK_{ZXELAR}, a transaction of the format $(update, PK_{new})$ must be submitted with a signature from the current SK_{ZXELAR}. Then the contract sets $PK_{\mathsf{ZXELAR}} = PK_{new}$.
3. Every time the validators update the threshold key for SC from PK^i to PK^{i+1}, ZXELARrequests that validators use SK^i to sign $(update, PK^{i+1})$. Subsequently this signature is posted to SC_{ZXELAR}which updates PK_{ZXELAR}.

6.2 State synchronization

Let q_S denote an arbitrary question about the state of chain S. Examples of such questions include:

- *"At what block round, if any, did a transaction tx appear?"*
- *"What is the value of a certain data field?"*
- *"What is the Merkle root hash of the entire state of S at block round 314159?"*

Let a_S denote the correct answer to q_S and suppose an end-user or application demands that a_S be posted to chain D. ZXELARnetwork meets this demand as follows:

1. The user posts a request q_S on one of the bridge accounts (which are subsequently picked up by the the validators) or directly to the ZXELARblockchain.

2. As part of ZXELARconsensus, each validator must run node software for chains S, D.

ZXELARvalidators
query the API of their chain S node software for the answer a_S and report the answer to the ZXEL
AR
chain.

3. Once $> F$ weighted validators report the same answer at round R,

ZXELARasks validators to sign a_S.4. Using threshold cryptography the validators sign a_S.

The signature is included in block $R + 11$.

5. Anyone can take the signed value a_S from block $R + 11$ and post it to D.

6. The request has been serviced. Any application on D may now take the signed value a_S, query D_{ZXELAR} for the latest PK_{ZXELAR}, and verify that the signature of a_S corresponds to PK_{ZXELAR}. The validators also post a_S to the bridge account on chain D, which applications can retrieve.

6.3 Cross-Chain Asset Transfer

The network enables cross-chain transfers of digital assets by extending the state synchronization workflow of Section 6.2.

A sufficient supply of pegged-S tokens is printed and controlled by D_{ZXELAR}upon its initialization. Suppose a user demands to exchange x amount of tokens on source chain S for x amount of pegged-S tokens on destination chain D, to be deposited at a D-address w_D of the user's choice. We present the fully general workflow, which supports arbitrary source chains S—even chains such as Bitcoin that do not support smart contracts:

1. The user (or an application acting on the user's behalf) posts a transfer request (x, w_D) to the threshold bridge account which is subsequently routed to the ZXELARnetwork.

2. ZXELARvalidators use threshold cryptography to collectively create a fresh deposit address d_S for S. They post d_S to the ZXELARblockchain.

3. The user (or an application acting on the user's behalf) learns d_S by monitoring the ZXELARblockchain. The user sends x amount of S-tokens to address d_S via an ordinary S-transaction tx_S using her favourite software for chain S.

   (Due to the threshold property of d_S, tokens cannot be spent from d_S unless a threshold number of the validators coordinate to do so.)

4. tx_S is posted on ZXELAR. The validators query the API of their chain S node software for existence of tx_S and, if the response is "true", report the answer to the ZXELARchain.

5. Once $> F$ weighted validators report "true" for tx_S at round R, ZXELARasks validators to sign a transaction a_D that sends x amount of pegged-S tokens from $D_{\mathsf{ZXELAR}}$to w_D.

6. Using threshold cryptography the validators sign a_D. The signature is included in block $R + 11$.

7. Anyone can take the signed value a_D from block $R + 11$ and post it to D.

8. The request has been serviced, once a_D is posted on D the transfer is processed.

Now suppose a user demands to redeem x' amount of wrapped-S tokens from chain D back to chain S, to be deposited at a S-address w_S of the user's choice. The workflow is as follows:

1. The user initiates a transfer request (x', w_S) by depositing x' amount of wrapped-S tokens into c_D via an ordinary D-transaction using her favourite software for chain D.

2. (x', w_S) is posted on ZXELAR. The validators query the API of their chain D node software for existence of (x', w_S) and, if the response is "true", report the answer to the ZXELARchain.

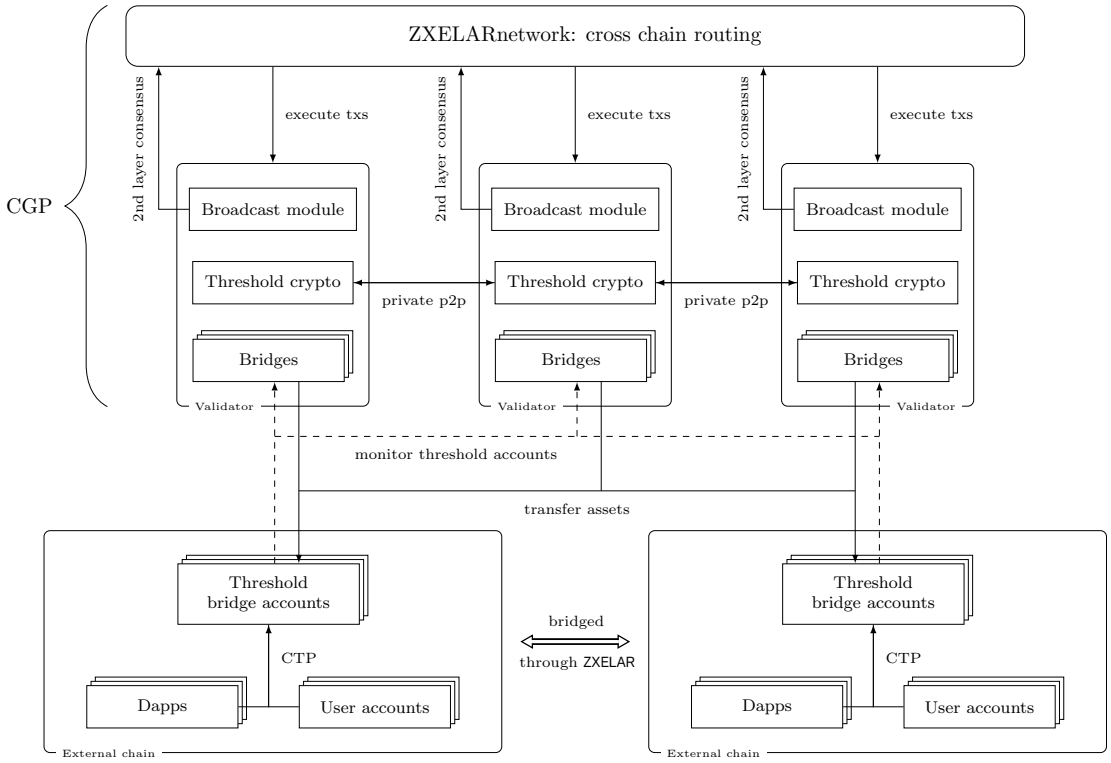

Figure 1: Component diagram

3. Once $> F$ weighted validators report "true" for (x', w_S) at round R, ZXELARasks validators to sign a transaction a_S that sends x' amount of S tokens from $S_{\mathsf{ZXELAR}}$to w_S.
4. Using threshold cryptography the validators sign a_S. The signature is included in block $R + 11$.
5. Anyone can take the signed value a_S from block $R + 11$ and post it to S.
6. The request has been serviced, once a_S is posted on S the transfer is processed.

Additional requests supported by the CGP routing layer include locking, unlocking or transferring assets across chains.

Achieving Atomic Cross-Chain Transaction Flow. Depending on the cross-chain request type, ZXELAR tries to ensure that the corresponding transactions are executed on multiple chains or none. Towards this, every request can be in one of the following states in ZXELARblockchain: *(initialized, pending, completed,timed out)*. If a *timeout* at the pending stage is triggered, the request returns an error code. Some timeout events also begin a *refund* event: for instance, if an asset from one chain needs to be transferred into an asset on another chain, if the receiving chain did not process the transaction, the asset is refunded back to the original user.

7 Cross-Chain Transfer Protocol (CTP)

CTP is an application-level protocol that makes it easy for applications to leverage cross-chain features. We explain the integration by focusing on asset transfer features (e.g., used in DeFi). These applications typically consist of three main components: front-end GUI, smart contracts on one chain, and an intermediary node that posts transactions between the front-end and the smart contracts. The front-ends interact with the user's wallets to accept deposits, process withdrawals, etc. Applications can leverage cross-chain features

by calling CTP queries analogous to HTTP/HTTPS GET/POST methods. These queries are subsequently picked up by CGP layer for execution and results are returned back to the users.

- *CTP Queries.* Application developers can host their applications on any chain and integrate their smart contracts with threshold bridge accounts to execute CTP queries.
- *Threshold bridge accounts.* Suppose an application developer builds their contracts on chain A. Then, they would reference threshold bridge contracts to obtain cross-chain support. This contract allows applications to:
  - Register a blockchain it would like to communicate with.
  - Register assets on that blockchain that it would like to leverage.
  - Perform operations over the assets such as accept deposits, process withdrawals, and other functions (similar to, say, ERC-20 contract calls).

Suppose a prominent DeFi application, MapleSwap, that natively resides on chain A registers with a threshold bridge account.
The ZXELARvalidators collectively manage the contract itself on the corresponding chain.
Suppose a user wants to submit a deposit into a trading pair between assets X and Y that reside across the two chains, respectively. Then, when a user submits such a request, it is routed via the threshold bridge account to the ZXELARnetwork for processing. Form there, the following steps are performed:

1. ZXELARnetwork understands that this application registered for the cross-chain support across the assets. It generates the deposits key leveraging threshold cryptography and consensus for the user on the corresponding chains A and B.
2. The associated public keys are returned to the application and displayed to the user who can use their favorite wallets to submit deposits. The corresponding secret key is shared across all ZXELARvalidators.
3. When the deposits are confirmed, ZXELARupdates its cross-chain directory to record that the user on the corresponding chains has deposited these assets.
4. The ZXELARvalidators execute multi-party protocols to generate a threshold signature that allows updating the threshold bridge account on chain A where the application resides.
5. The CTP query is then returned to the DeFi application smart contracts, which can update its state, update its yield formulas, exchange rates, or execute other application state-related conditions.

Throughout this process, the ZXELARnetwork, on a high-level, acts as a decentralized cross-chain read/write oracle, CGP is the routing layer in between chains, and CTP is the application protocol.

Additional Cross-Chain Requests. CTP supports more general cross-chain between applications across blockchains such as:

- Perform Public Key Name Services (PKNS). This is a universal directory for mapping public keys to phone numbers/twitter handles (a few projects, such as Celo, provide these features within their platforms).
- Cross-chain application triggers. An application on chain A can update its state if some another application on chain B satisfies a search criteria (interest rate $< X$).
- Smart contract composability. Smart contract on chain A can update its state based on state of contracts on chain B, or trigger an action to update a smart contract on chain B.

On a high-level, these requests can be processed since collectively, the protocols CTP, CGP, and ZXELAR network can pass and write arbitrary verifiable state information across blockchains.

8 Summary

Over the next years, significant applications and assets will be built on top of multiple blockchain ecosystems. ZXELARnetwork can be used to plug-in these blockchains into a uniform cross-chain communication layer. This layer provides routing and application-level protocols that meet both platform builders and ap-plication developers' demands. Application developers can build on the best platforms for their needs and leverage a simple protocol and API to access global cross-chain liquidity, users, and communicate with other chains.

References

[1] Althea peggy. https://github.com/cosmos/peggy. [Cited on page 2.]

[2] Deterministic usage of the digital signature algorithm (dsa) and elliptic curve digital signature algorithm (ecdsa). https://tools.ietf.org/html/rfc6979. [Cited on page 5.]

[3] Edwards-curve digital signature algorithm (eddsa). https://tools.ietf.org/html/rfc8032. [Cited on page 5.]

[4] Eos.io technical white paper v2. https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md. [Cited on page 1.]

[5] Ethereum: A secure decentralised generalised transaction ledger. https://ethereum.github.io/yellowpaper/paper.pdf. [Cited on page 1.]

[6] The near white paper. https://near.org/papers/the-official-near-white-paper/. [Cited on page 1.]

[7] Rainbow bridge. https://github.com/near/rainbow-bridge. [Cited on page 2.]

[8] Ren: A privacy preserving virtual machine powering zero-knowledge financial applications. https://whitepaper.io/document/419/ren-litepaper. [Cited on page 3.]

[9] Serum. https://projectserum.com/serum_white_paper.pdf. [Cited on page 3.]

[10] tbtc: A decentralized redeemable btc-backed erc-20 token. https://docs.keep.network/tbtc/index.pdf. [Cited on page 2.]

[11] Thorchain: A decentralized liquidity network. https://thorchain.org/. [Cited on page 3.]

[12] Kurt M. Alonso. Zero to monero. https://www.getmonero.org/library/Zero-to-Monero-1-0-0.pdf. [Cited on page 1.]

[13] Jean-Philippe Aumasson, Adrian Hamelink, and Omer Shlomovits. A survey of ecdsa threshold signing. Cryptology ePrint Archive, Report 2020/1390, 2020. https://eprint.iacr.org/2020/1390. [Cited on page 6.]

[14] Ran Canetti, Nikolaos Makriyannis, and Udi Peled. Uc non-interactive, proactive, threshold ecdsa. Cryptology ePrint Archive, Report 2020/492, 2020. https://eprint.iacr.org/2020/492. [Cited on page 6.]

[15] cLabs Whitepapers. https://celo.org/papers. [Cited on page 1.]

[16] Ivan Damgård, Thomas Pelle Jakobsen, Jesper Buus Nielsen, Jakob Illeborg Pagter, and Michael Bæksvang Østergård. Fast threshold ECDSA with honest majority. In *SCN*, volume 12238 of *Lecture Notes in Computer Science*, pages 382–400. Springer, 2020. [Cited on page 6.]

[17] Manu Drijvers, Kasra Edalatnejad, Bryan Ford, Eike Kiltz, Julian Loss, Gregory Neven, and Igors Stepanovs. On the security of two-round multi-signatures. In *IEEE Symposium on Security and Privacy*, pages 1084–1101. IEEE, 2019. [Cited on page 6.]

[18] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. https://groups.csail.mit.edu/tds/papers/Lynch/jacm88.pdf. [Cited on page 5.]

[19] Rosario Gennaro and Steven Goldfeder. One round threshold ecdsa with identifiable abort. Cryptology ePrint Archive, Report 2020/540, 2020. https://eprint.iacr.org/2020/540. [Cited on page 6.]

[20] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. Proceedings of the 26th Symposium on Operating Systems Principles, 2017. https://dl.acm.org/doi/pdf/10.1145/3132747.3132757. [Cited on page 1.]

[21] Evan Kereiakes, Do Kwon, Marco Di Maggio, and Nicholas Platias. Terra money: Stability and adoption.
https://terra.money/Terra_White_paper.pdf. [Cited on page 1.]

[22] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. https://eprint.iacr.org/2016/889.pdf. [Cited on page 1.]

[23] Chelsea Komlo and Ian Goldberg. Frost: Flexible round-optimized schnorr threshold signatures. Cryptology ePrint Archive, Report 2020/852, 2020. https://eprint.iacr.org/2020/852. [Cited on page 6.]

[24] Jae Kwon and Ethan Buchman. Cosmos: A network of distributed ledgers.
https://cosmos.network/resources/whitepaper. [Cited on pages 1 and 2.]

[25] Avalanche Team. Avalanche platform.
https://www.avalabs.org/whitepapers. [Cited on pages 1 and 2.]

[26] Gavin Wood. Polkadot: Vision for a heterogeneous multi-chain framework.
https://polkadot.network/PolkaDotPaper.pdf. [Cited on pages 1 and 2.]