XM : A Blockchain with Native Cross-chain Smart Contracts

XM

Abstract. This white paper proposes XM , a blockchain with generic universal smart contract support that connects both smart contract blockchain such as Ethereum, Ethereum L2 rollups and EVM compatible chains (Polygon, Avalanche, BSC), Solana, SUI, and TON network, and even non smart contract blockchains such as Bitcoin and Dogecoin. XM consists of a Proof-of-Stake blockchain and observers and signers for external blockchains. The observers scan external chains for relevant events, transactions, and states at a point in time, and reach consensus on observation on XM 's blockchain. The signers collectively possess a single Threshold Signature Scheme (TSS) key that is able to send authenticated messages to external chains and hold assets like normal accounts/addresses on external chains. Smart contracts on XM support arbitrary logic that executes conditionally on external chain events, and can directly update external chain states via its TSS signed transactions. XM thereby enables universal dApps that interact with different blockchains natively and directly without wrapping or bridging any assets.

1. Introduction

It's hard to imagine a single blockchain would suffice for all our society's use cases. A multi-chain future seems inevitable. However, a multi-chain future without interoperability between the blockchains could be paralleled to the Internet before TCP/IP. Today's blockchains are too fragmented and are by-nature not interoperable, hindering mass adoption of the technologies. For example, a decentralized application (dApp) must be married to a specific blockchain. If a user onboards into the crypto ecosystem through a given dApp, this fragmentation makes immense barriers for the user to fluidly adopt or try a dApp on another chain. To address the issues of interoperability, there have been a few proposals and projects that specifically emphasize the ability to interoperate. However, the majority of interoperability systems only apply to specific blockchains, standardize their protocols within their own systems requiring other blockchains to adopt or through complicated, restricted, and/or less secure bridges to join (see Figure 1). In this whitepaper we propose a novel, public L1 blockchain that actively and agnostically connects blockchains and facilitates interoperability. Furthermore, we propose a generic smart contract on blockchain that can hold and manipulate assets on external blockchains directly, thereby enabling generic

smart contracts that can custody assets on external chains. This opens the door to boundless cross-chain dApps.

Blockchains are naturally closed systems. The goal of this whitepaper is to design and specify a practical system that is generic in its inter-blockchain capability, without forcing existing blockchains to adopt new standards or a new blockchain that every asset needs to move to, and do so in a decentralized, byzantine fault tolerant way. In other words, we aim to create a public blockchain that supports real cross blockchain transactions, message delivery, and general cross-chain smart contracts. According to our extensive survey, to satisfy this goal, the best pragmatic approach is the decentralized notary scheme on top of an incentivized Proof-of-Stake replicated state machine (aka blockchain) which we call XM .

XM is first of all a public blockchain with Proof-of-Stake validators. It's trusted that a super majority (>66% nodes) of the validator nodes are honest and act according to protocol, and collectively serve as notaries. In addition to being a blockchain, interoperability requires observing other blockchains. Thus each XM validator node is attached with an observer that scans other blockchains for relevant events (event log, transaction, or state at a certain time). The observers report the relevant events to XM and reach consensus. XM uses custom logic to update its state in response to the reported events. On the other hand, in order to change state on other blockchains, each validator is also attached with a signer holding a key share. Collectively all the validators hold a single public/private key pair which can initiate transactions on other blockchains to change state directly. The signature scheme can be some kind of threshold signature scheme such as GG18/GG20 ECDSA/EdDSA, or BLS threshold/aggregate signatures, depending on the cryptography on different chains and their smart contract capability/cost. The presence of a single public key and address in the XM system allows XM to custody assets on external blockchains which might not have adequate smart contract capability such as Bitcoin. Such ability allows powerful cross-chain (or ‘universal’) dApps to be built on top of native XM cross-chain smart contracts. This capability looks much like on Ethereum where a smart contract can be trusted to manage assets according to predetermined rules, except on XM , a smart contract can leverage and manage assets on any connected blockchain.

In summary, XM is designed to be a decentralized cross-blockchain smart contract platform. The vision of XM is to be a public computer on all important blockchains, on top of which cross-blockchain decentralized applications can be easily built as public, trustless, and persistent smart contracts.

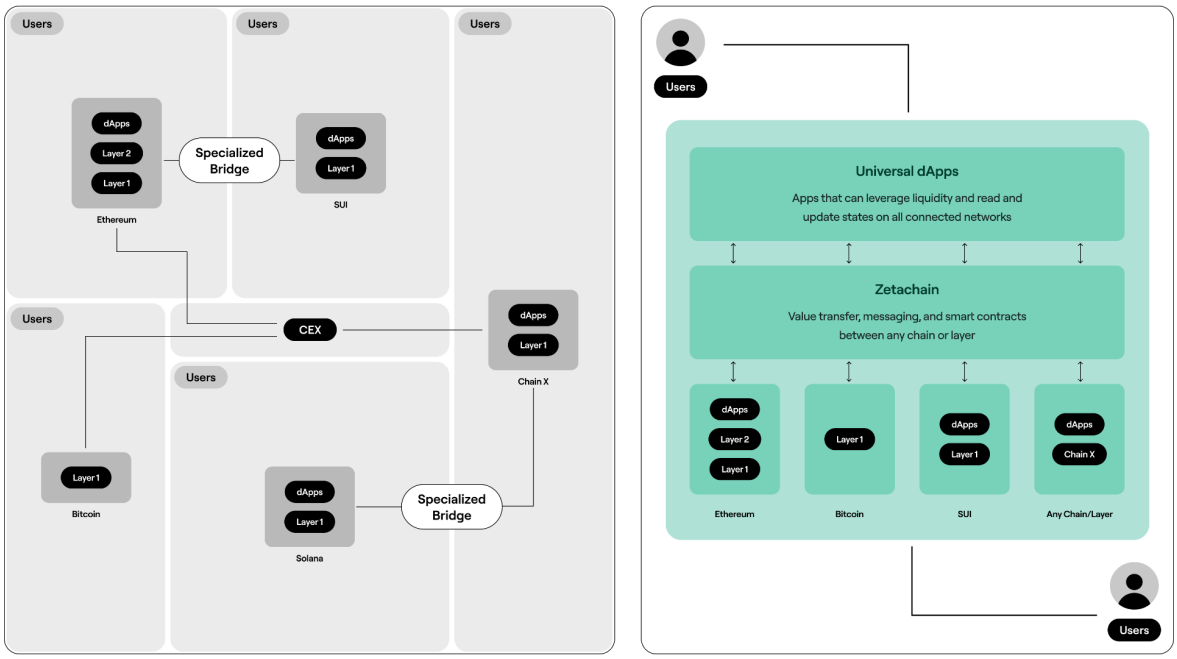

Figure 1. Before and After XM . Sub figure (a) on the left: Current ecosystem. Users and developers are siloed into respective chains, and current cross-chain solutions are disparate, resulting in major, growing fragmentation. Sub figure (b) on the right: Ecosystem with XM . Users, developers, and apps can operate across chains in a seamless manner. New paradigm of Universal dApps enabled.

2. Background: Evolution of Blockchains

2.1. Bitcoin: the original decentralized cryptocurrency

Blockchain, pioneered by Bitcoin, is a decentralized and permissionless public ledger built on cryptography. The core mechanism is byzantine fault tolerant distributed consensus, which Bitcoin solved by a combination of techniques from cryptography, economic incentives, and computer science. Key innovations in Bitcoin include the use of elliptic curve digital signatures algorithm (ECDSA) for self-custody of funds, and the use of Proof-of-Work to reach distributed consensus (ordering of the ever growing log of transactions) and maintain resistance against sybil attacks. Bitcoin also introduces the first major application of blockchain technology—a p2p cryptocurrency. The Bitcoin network has been highly successful, even though it has not fulfilled its promise of being electronic cash. Rather, Bitcoin has become the most secure, decentralized, and stable store of value due to its technical simplicity and robustness, high degree of decentralization and low barrier of participation, and predictable and conservative monetary policy.

The Bitcoin network consists of nodes connected by a p2p network. Participants include users and miners. The Bitcoin network collectively maintains a growing ledger that is a sequence of user transactions. A user transaction is a signed message that "spends" a certain amount of coins controlled by the user. The Bitcoin network does not explicitly maintain the balance state of each account; the only state of the network is the set of current UTXOs—unspent transaction outputs. A user's balance of BTC is the sum of all UTXOs that can be spent by the user. A user transaction includes one or more UTXOs as inputs, and creates one or more UTXOs as outputs, thus changing the state (UTXO set). Bitcoin supports a limited form of scripting: a transaction can send coins to a script, and whoever can satisfy the script (i. e. supply data to make it evaluate to 1) may spend the coins. The scripting language is deliberately simple and Turing-incomplete —namely without branch and looping structures — but supports quite a few simple but fundamentally useful applications such as multi-sig, atomic swaps, etc.

2.2. Ethereum: the programmable blockchain with smart contracts

While Bitcoin is conceptually a simple ledger (ordered sequence of transactions) with basic scripting features that has served as the canonical example of a blockchain, it is not the limit of what a blockchain can do. For example, due to the limited scope of the verification function of the Bitcoin protocol, it's not possible to issue new coins on the Bitcoin network. The Bitcoin network is not programmable in the sense that an arbitrary state transition function can be implemented. The only state transition function that Bitcoin supports is the hard-coded UTXO set change. In summary, no applications other than BTC currency can piggy-back on the Bitcoin network, inheriting its consensus, decentralization, and security. To extend the scope of blockchain to support Turing-complete programmability, Ethereum was born. Ethereum borrows the Proof-of-Work from Bitcoin for its consensus, and has made several important innovations that make it a public programmable blockchain. First, Ethereum defines a virtual machine (EVM) that provides a Turing-complete sandbox environment to specify arbitrary state transition functions (smart contracts). Second, Ethereum moves away from the UTXO model in Bitcoin to an account-based system where account store state. There are two kinds of accounts: External Owned Accounts (EOAs) which are controlled by a private key, and smart contract accounts which work autonomously according to their own logic. The availability of smart contracts on Ethereum makes it one of the most widely used dApp blockchains with thousands of applications deployed, such as financial derivatives, exchanges, NFTs, gambling, and games. Smart contracts on Ethereum are like objects in an object-oriented programming language where state can be stored and functions can be called to change its state. Users can interact with smart contracts by sending messages to it, and smart contracts can also send messages to other smart contracts (invoking) to change their state. The smart contracts can enable very complex applications, and can enable some very powerful operations such as flash loans or flash swaps that have no analogy in non-blockchain applications. This is made possible by the powerful atomicity of transaction that invokes smart contract functions: it either completes or completely reverts. Over the years more and more blockchains such as Polkadot, Solana, Avalanche, and Cosmos have arisen and support nearly Turing-complete smart contracts.

2.3. Emergence and challenges of multi-chain

While some people tend to favor one chain to rule them all, the reality is that blockchain technology and markets are evolving at an astonishing pace and it is becoming more and more apparent that the future of the ecosystem will be comprised of multiple blockchains serving their own purpose with their own

tradeoffs in terms of security, decentralization, scalability, speed, cost, compliance, and so on. In this multi-chain future, a key limitation is that blockchains are designed to be a closed system. Transactions that happen on a blockchain can only rely on the state of their respective blockchain, and can only modify the state of their respective blockchain. External information cannot be reliably brought to the blockchain without a trusted third party (oracle). Transactions that involve multiple blockchains must go through a trusted party, such as a centralized exchange. As a result, there is currently no decentralized, permissionless, and public service that facilitates generic atomic transactions (not only atomic swapping, but also arbitrary logic) that involves multiple blockchains.

Popular cross-chain or cross-blockchain strategies include side-chains, relays, notary schemes, hash time-lock contracts, and blockchains of blockchains.

- First, side-chains/relays are popular solutions to implement bridges that primarily enable portable assets. In these, some assets have a home ledger that is authoritative on its ownership, but through bridges one can move the asset to other blockchains while being confident that the asset is able to move back to the home blockchain. Relay is one of the direct mechanisms to facilitate interoperability, where instead of relying on a trusted intermediary to provide information on chain A to chain B, chain B implements a thin client of chain A using smart contracts, and is able verify whether a particular event, transaction, or state at certain point in time has occurred on chain A. This is often called trustless because there is no additional trust assumption beyond trusting the two involved chains. In other words, no trust is required on the validity of the delivery mechanism of messages from chain A to chain B, other than that the message is delivered and delivered in time. Examples of relays include the BTCRelay on Ethereum (a SPV client of Bitcoin) and the Rainbow bridge of Ethereum on the NEAR blockchain. Relays are also popular mechanisms for side-chains.
- Second, notary schemes are mechanisms where a trusted entity (or a set of) is tasked with notarizing claims such as “event X has happened on blockchain A”. The most obvious notary schemes are centralized exchanges, which are trusted entities to facilitate cross-blockchain exchanges of coins. Notary schemes do not have to be centralized; for example the Interledger project, in its “atomic mode” can be categorized as a decentralized, byzantine fault tolerant notary scheme to facilitate cross-ledger transfers. Note that notary schemes are the most flexible in terms of interoperability use cases, because they are able to act with arbitrary logic in response to events on discrete blockchains. Another notable decentralized notary scheme is THORChain which implements a DEX for native coins across several different chains, using a set of incentivized validators as notaries.
- Third, hash time-lock contracts (HTLC) are constructs of smart contracts that can facilitate atomic swaps across blockchain chains trustlessly without additional trust beyond the participating two blockchains. The key words are “atomic” and “trustless”. Atomic means that the transactions (involving two parties) are either complete or revert (as if nothing has happened). Trustless means no third-party needs to be trusted for the atomic swap. It works roughly by two parties interactively deploying and interacting with smart contracts on both sides. The core idea is with a hash of secret that is conceived by party A and used by both parties, and party A is forced to reveal the secret when claiming party B's coin, which can then be used by party B to claim party A's coin. Examples of HTLC include XClaim BTC/Ethereum or BTC/Polkadot bridge, and the Lightning Network on Bitcoin.

- Fourth, blockchains of blockchains (BoB) are frameworks that provide data, network, consensus, incentive, and contract layers for constructing application-specific blockchains that interoperate between each other. Note that BoB does not solve current interoperability problems directly. Rather it enables the creation of new interoperable blockchains. To connect to "legacy" chains, some sort of bridge or other mechanism shown above must be employed. Important examples of BoB are Polkadot and Cosmos, built on Substrate and Tendermint as consensus engines, and XCMP and IBC as cross-chain communication protocols.

Strategy	**Use scenarios**	**Trust Assumption**
Relay/Side-chain	Portable Assets	Trustless
Notary Scheme	Arbitrary	Trustful
HTLC	Atomic Swaps	Trustless
BoB	New blockchains	Trustless/Trustful

Table 1. Comparison of existing cross-chain strategies

Each of these broad strategies has its strengths and weaknesses in technical complexity, trust assumptions, level of interoperability, and use cases. Our discussion here is brief and incomplete, but still we can very roughly categorize the characteristics of these strategies; see Table 1 for a comparison of these strategies.

3. Interoperability Related Work

In this section, we pick some of the recent and most relevant projects, ideas, and trends to provide context for this paper and XM. For more academic cross-blockchain research please refer to a comprehensive survey [1].

3.1. Cross-chain Communication

A basic building block of any cross-blockchain interoperability is the ability to communicate and prove to chain B that a certain transaction happened on chain A.

BTCRelay [4], Rainbow Bridge [5]: Consider the task of building a one-way bridge on Ethereum from Bitcoin. When a user on Bitcoin sends 1 BTC to a given custody address, one wrapped BTC is issued on

Ethereum. To do this in a trustless way, a smart contract on Ethereum can verify the transaction on Bitcoin, and issue a corresponding wrapped BTC coin on Ethereum. BTCRelay is such an example. For an Ethereum smart contract to verify the transaction on Bitcoin, someone (off-chain service) can submit the transaction, together with the transaction Merkle proof. The Ethereum smart contract verifies the proof based on the chain of block headers stored in the smart contract. This smart contract is essentially a light client of Bitcoin. Even though the strength of the proof is a bit lower than a full node (would be vulnerable to certain 51% attacks), this kind of bridge is strong and trustless, albeit rather expensive to operate because the chain of the block headers have to be constantly updated in the smart contract. The Rainbow Bridge is also a good example of a trustless bridge, between Ethereum and NEAR.

Wormhole [17]: Wormhole is also a cross-chain message delivery service, but it's not trustless. Rather, it depends on a set of validator nodes to attest the validity of the message delivered. Consider the same task of building a one-way bridge on Ethereum from Solana. When a user sends 1 SOL to a certain custody address, one wrapped SOL is issued on Ethereum. The Ethereum smart contract does not verify the transaction on Solana in order to issue the wrapped coin; it trusts that the super majority of the set of Wormhole validators are honest and correct. The security of Wormhole relies on the super-majority of the validators being honest. It appears that Wormhole relies on reputations of validators to build trust.

LayerZero [13]: LayerZero is a communication layer for facilitating cross-chain message delivery. It's essentially a weaker form of Relay (see introduction about Relay). The idea is to enable Chain B to verify that a given transaction or event has happened on Chain A. If Chain B supports general smart contracts, a light client of chain A can be implemented in a smart contract so as to verify information about Chain A in a trustless manner. However, even a light client can be expensive to run in a smart contract, both in terms of computation and storage; for example the BTCRelay on Ethereum appears to be discontinued. LayerZero reduces these costs with an ultra-light client on smart contract which does not report and store the whole chain of block headers (or a significant part of it). Rather, LayerZero relies on trusting a block header without a chain of block headers that can trace back to some known trusted block. The key assumption of the security of LayerZero is that the two parties—Relayer, who provides proof of transaction, and Oracle, who provides the block header—are non-colluding. In our terminology and categorization, LayerZero is not “trustless” due to the trust needed for the independence of the two parties. We use a stricter definition of trustless as where the validity (not necessarily liveness, censorship resistance) of messages does not depend on trust in anything other than the two participating blockchains. If the relayer and oracle collude, they can defraud LayerZero by making up an invalid block header (costs about 2 Ether to compute PoW nonce which is the coinbase reward of each block), and make chain B believe that a non-existent transaction has happened on chain A. LayerZero essentially outsources its security to third-party relayer and oracle.

Axelar [19]: Similar to LayerZero, Axelar provides a message passing protocol for building cross-chain smart contract apps. Axelar connects to 81 chains including heterogeneous smart contract platforms including Solana (SVM), Osmosis (Cosmos IBC), although it cannot connect to non smart contract chains directly such as Bitcoin and Dogecoin. The cross-chain message passing appears to be decentralized; there are a set of validators or observers that observe certain smart contracts on connected chains and forward its message to another smart contract on possibly another chain, by initiating a smart contract call there with the message included. Axelar also has its own L1 blockchain but it seems the programmability of the blockchain (its WASM smart contract platform) is not needed for cross-chain message passing.

IBC [10]: Inter-Blockchain Communication (IBC) protocol is a TCP/IP-like protocol for communication between sovereign blockchains. IBC is an end-to-end, connection oriented, stateful protocol between blockchains. Practically, IBC usually requires fast finality chains such as CometBFT, and the blockchain must support IBC protocol such as Cosmos SDK-built chains. For the blockchains that support IBC, they can establish connections, and through these connections one blockchain can verify proofs against the consensus states of another blockchain. Each blockchain that supports IBC must run a light client that is capable of verifying proofs on the other blockchain in order for them to be connected. The IBC module must also handle production of proofs, and a separate process (relayer) must relay the packet and proof to the counterparty chain. Among the blockchains that support IBC, very strong interoperability can be established, such as coin transfer, atomic swaps, cross-chain decentralized exchanges, and even cross-chain smart contracts. The major drawback of IBC is that it requires adoption–which is a lot to ask of other blockchains, and also might not be possible for legacy blockchains.

Intent [20] based cross-chain solutions: e.g., Across Protocol. Intents are a relatively new mechanism for conducting cross-chain transactions with hybrid on-chain and off-chain components. Users interact with contracts that conform to the Intent EIP [20] and "fulfillers", i.e., third parties compete to fulfill the order for a fee. The cross-chain orders can be bridging some token to another blockchain, or they could be a swap order that results in a different token on a different chain. Note that the Intents themselves do not specify a particular way to complete the cross-chain transaction (bridging or swap), rather it specifies a unified and unambiguous way for contracts and users to signal the "intent" to trade and leave the fulfillment of the trade up to mostly private market makers. Intents are descriptive protocols for a subset of cross-chain transactions.

3.2. Cross-blockchain Asset Transfer

Hop [9]: Hop is a protocol to send coins across rollups and their underlying L1 in a trustless manner. Rollups are by default siloed systems and the asset transfer between rollups and L1 can be slow and expensive. For example, optimistic rollups usually take a week to exit into L1; on the other hand, zk-rollups can instantly validate exit but it involves high computation which is expensive on L1. Hop solves the problem of moving coins across rollups by creating bridges and bridge coins, and uses AMM markets to exchange coins rather than sending coins directly. Specifically, Hop creates bridge coins for each rollup, and the bridge coins can be moved around in batches so as to decrease the cost. The bridge coin acts as an intermediary asset in transferring a coin on rollup A to rollup B. Hop uses the existing rollup bridges to do cross-rollup transactions so it does not need a separate off-chain service.

Connext [3]: Connext is a trust-minimized solution for cross-chain asset swaps. The idea is somewhat like generalized atomic-swaps, using Hash Time Locked Contracts (HTLC) to ensure transaction atomicity. It uses a network of off-chain routers to create a market and AMM style pricing mechanism. The safety of user funds do not depend on third-parties, only the liveness of the system does. Compared to Hop, Connext uses off-chain services and therefore can connect beyond rollups on a single L1; compared to externally verified solutions, Connext is application specific and not general purpose. For example, it cannot be adapted to send arbitrary messages or cross chain contract calls.

THORChain [15], Chainflip [16]: THORChain (along with similarly built competitors like Chainflip) is a decentralized liquidity network that facilitates AMM style native L1 coins on different blockchains,

including Bitcoin, Litecoin, Bitcoin Cash, Ethereum. Notably, THORChain is not, strictly speaking, a bridge, as it does not lock & wrap coins and transact on wrapped coins. Rather, THORChain is an application-specific blockchain that maintains the pool, logic, and management of vaults on different chains for swapping. THORChain distributes the signing key using the GG20 TSS scheme and has its own implementation based on Binance's TSS library. XM is in-part inspired by the design of THORChain, and can be thought of as a simpler and more generalized platform which enables not only swapping, but a generic smart contract platform that allows arbitrary cross-chain applications to be built easily. For example, developers can implement similar functionality to THORChain as a smart contract on XM .

Synapse [14]: According to public information, Synapse is supposed to be an externally verified validator set based system for cross-chain swaps. It issues AMM smart contracts on external chains, and some composite stablecoin as an intermediary asset to cross-chain. To move the intermediary stablecoins across chains it appears to use a burn and mint strategy. Detailed public information about their validator mechanism is not available at the time of writing this paper.

CCTP by Circle : The Cross-Chain Transfer Protocol (CCTP) v1/v2 by circle (the issuer of USDC stablecoin) is a cross-chain asset transfer protocol, with message passing authenticated and facilitated centrally by Circle service provider. It's similar to LayerZero, Wormwhole, or Axelar with USDC attached to every cross-chain messaging. The difference is that CCTP is not decentralized; only that its interfaces on multi-blockchains are contract calls.

3.3. Cross-blockchain Smart Contract

Quant Network [18]: Functionality-wise, the Quant network and its Overledger [18] is the closest to XM . The Quant network is a centralized service that provides a standardized web-service-based access to the connected public or private blockchains, or regional legacy database ledgers. It supports general programmability triggered by events on those blockchains (transaction to/from a given address, smart contract interaction, events, state changes, etc.), via popular languages and frameworks such as Javascript, Java, Python, etc. XM aims to achieve similar general programmability, but with an incentivized public blockchain, with far reduced trust assumptions, more transparency, complete verifiability and auditability.

ICP/Chain-Key [2]: The Internet Computer Protocol (ICP) has proposals to enable interoperability to the Bitcoin network via its Chain-Key technology, which is similar to the distributed threshold signature scheme. With Chain Key, ICP in principle can custody funds on the Bitcoin network. It's unclear how ICP observes the Bitcoin network, and how their smart contract platform interacts with external blockchains.

HyperService [11]: HyperService proposes a cross-chain smart contract platform that is chain agnostic. It consists of two components: a high level language HSL to describe a cross-chain dApp, and an execution layer that ensures financially atomic transactions.

3.4. Blockchain of Blockchains (BoB)

The most prominent BoBs are Cosmos and Polkadot. BoBs are usually frameworks that aim at tight interoperable application-specific blockchains. Polkadot, for example, provides a relay chain which handles all consensus, and Parachains which can be different blockchains with different state-transition functions. The Parachains are tightly integrated and can interoperate seamlessly via the relay-chain.

The Cosmos ecosystem, on the other hand, does not share consensus, so the interoperability between Cosmos chains is less tight. Every Cosmos chain is sovereign with their own choice of consensus (typically CometBFT-based fast finality). The Cosmos ecosystem relies on the IBC protocol (see section 3.1), and special blockchains called Hubs to facilitate cross-chain asset transfers, and even cross-chain smart contracts.

To enjoy interoperability in Cosmos or Polkadot, the blockchains typically need to be built on some common ground. Legacy blockchains, or new blockchains with their own consensus, cannot be part of BoBs.

4. XM Blockchain Architecture

4.1. Overview

At a high level, XM is a Proof of Stake (PoS) blockchain built on the Cosmos SDK and CometBFT Tendermint PBFT consensus engine. As a result, XM enjoys fast block time (~4s) and instant finality (no block confirmation needed, no re-organization allowed). The CometBFT Tendermint consensus engine has been demonstrated to scale to ~300 nodes in production, and with future upgrades with BLS threshold signatures the number can potentially increase to 1000+. The throughput of transactions of the CometBFT Tendermint consensus engine that XM uses can reach 4000+ transactions per second (TPS) under ideal network conditions [10]. Note that the cross-chain TPS cannot reach nearly as high because cross-chain transactions latency/throughput may be limited by external chain latency/throughput, TSS key-sign throughput, and various other factors such as external node RPC speed, etc.

The XM architecture consists of a distributed network of nodes, often referred to as validators. Validators act as decentralized observers that can reach consensus on relevant external state and events, and can also update external chain state via distributed key signing. XM accomplishes these functions in a decentralized (without a single point of failure, trustless, permissionless), transparent, and efficient way. Contained within each validator is the ZetaCore and ZetaClient. ZetaCore is responsible for producing the blockchain and maintaining the replicated state machine. ZetaClient is responsible for observing events on external chains and signing outbound transactions. ZetaCore and ZetaClient are bundled together and run by node operators. Anyone can become a node operator to participate in validation provided that enough bonds are staked. See Figure 2 for a high level illustration.

Validators: XM uses the CometBFT Tendermint consensus protocol which is a partially synchronous Byzantine Fault Tolerant (BFT) consensus algorithm. Each validator node can vote on block proposals with voting power proportional to the staking coins (ZETA) bonded. Each validator is identified by its consensus public key. Validators need to be online all the time, ready to participate in the constantly

growing block production. In exchange for their service, validators will receive block rewards, and potentially other rewards such as gas fees or processing fees, proportional to their bonded staking coins.

Observers: Another set of important participants of XM consensus are the observers who reach consensus on external chain events and states. The observers watch externally connected chains for certain relevant transactions/events/states at particular addresses via their full nodes of external chains. The observers can be further divided into two roles: sequencer and verifier. The sequencer discovers relevant external transactions/events/states and reports to verifiers; the verifiers verify and vote on XM to reach consensus. The system requires at least one sequencer and multiple verifiers. The sequencer does not need to be trusted, but at least one honest sequencer is needed for liveness.

Signers: The XM collectively holds standard ECDSA/EdDSA keys for authenticated interaction with external chains. The keys are distributed among multiple signers in such a way that only a super majority of them can sign on behalf of the XM . It's important to ensure that at no time is any single entity or small fraction of nodes able to sign messages on behalf of XM on external chains. The XM system uses bonded stakes and positive/negative incentives to ensure economic safety.

In practice, all above roles (except sequencer) are collocated in the same computer node, sharing software and credentials such as validator keys and bonded stakes and the associated rewards/slashing. XM is planned to transition from Proof-of-Authority at first to a fully delegated Proof-of-Stake (DPoS) model over time, and gradually delegate the governance of the blockchain to ZETA coin holders via on-chain voting.

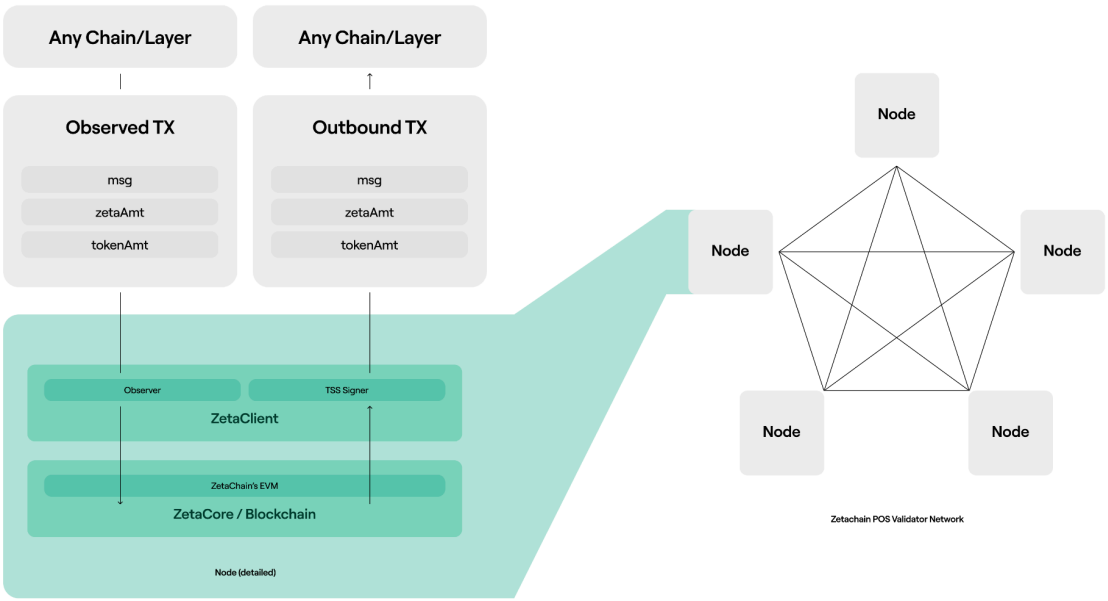

Figure 2. XM High Level Architecture.

4.2. Observers

Observers are tasked with monitoring external chains for relevant transactions. Observers are continually scanning for external chain events responsible for both burning and minting the native coin (ZETA), messages & smart contract calls, as well as other events that dApps register on XM . Each observer independently observes using its own full node of external chains, and all the observations must reach consensus on the XM before being considered finalized. Once events are finalized, it automatically triggers an execution of XM logic, which can be defined as a custom Cosmos SDK module, or XM native smart contract.

There are two modes of observation: Active and Passive mode. Active observation constantly scans the external blockchains for relevant transactions/events/states. Passive mode relies on a sequencer (or a small set thereof) to scan and report transactions/events, together with Merkle proof. The observers verify the proof and reach consensus on the verification on-chain. The active mode has the advantage of being always live and censorship-resistant due to decentralization, but the cost of each node is high because it needs full nodes (of external chains) for the scanning. Passive mode is much less costly, as verification can be done with a light client. Only one or a few sequencers need access to a full node, which is much cheaper and makes scaling to multiple external chains and more validator nodes much easier. The disadvantage of passive mode is that the liveness of external chain inbound observation is dependent on the sequencer, and also subject to censorship by the sequencer. This is the same situation as the optimistic rollup where the liveness of the rollup is dependent on a sequencer. To mitigate this, everyone is able to be a sequencer if they so choose, and a sequencer can be incentivized by the creation of a competitive market. In particular, dApps have a vested interest in running a sequencer. Another advantage of running passive observation mode with a sequencer is that the dApps are in control of the observation ordering. For efficiency reasons, the active mode does not enforce observation ordering, but if the observation ordering is important to a dApp, it can opt to run its own sequencer in synchronous observation mode (i. e. wait for each observation to be finalized by XM before moving on to the next).

4.3. Multi-party Threshold Signature Scheme

XM needs to hold an account on external chains in order to custody funds on that chain (manage a pool, vault, etc.), and to perform privileged actions (burn, mint, move funds out of the vault, etc.). This is required for general-purpose cross-chain smart contracts, as a core feature of smart contracts is to manage assets autonomously. On Ethereum for example, a smart contract has an address and can hold any asset like an External Owned Address (EOA, normal user account). This ability enables many powerful applications such as AMM pools, lending/borrowing pools, etc., where users pool their assets and let smart contracts manage them according to a smart contract's predetermined rules. In order to hold an account, XM needs to have a private key. To avoid a single point of failure (single location of the private key, single dealer in generating the key), XM needs a distributed threshold signature scheme.

This is also needed to support non-smart-contract chains such as Bitcoin, Dogecoin, or smart-contract platforms that are expensive to verify multi-sig. To avoid any single point of failure, XM uses

state-of-the-art multi-party threshold signature scheme (TSS) [7, 8] based on implementations from THORChain TSS [15] and Binance tss-lib [12]. To the outside world, the XM validators collectively possess a single ECDSA/EdDSA private key, public key, and address, and the signature signed by XM can be verified efficiently and natively by standard ECDSA/EdDSA verification procedure by the connected blockchains. Internally, the private key is generated without a dealer, and the private key is distributed in all the validators. At no time is a single entity or a minority of validators able to piece together the private key and sign messages on behalf of the whole network. The key generation and signing procedures are done by Multi-Party Computation (MPC) which reveal no secret of any participating node. Because XM can hold a TSS key and address, XM can support smart contracts that can manage native vaults/pools on connected chains including Bitcoin. This effectively adds smart contract capabilities to the Bitcoin network, and potentially other non-smart contract blockchains. The TSS employed by XM gives the performance and convenience of hot wallet with cold wallet level security. See Figure 3 for an illustration.

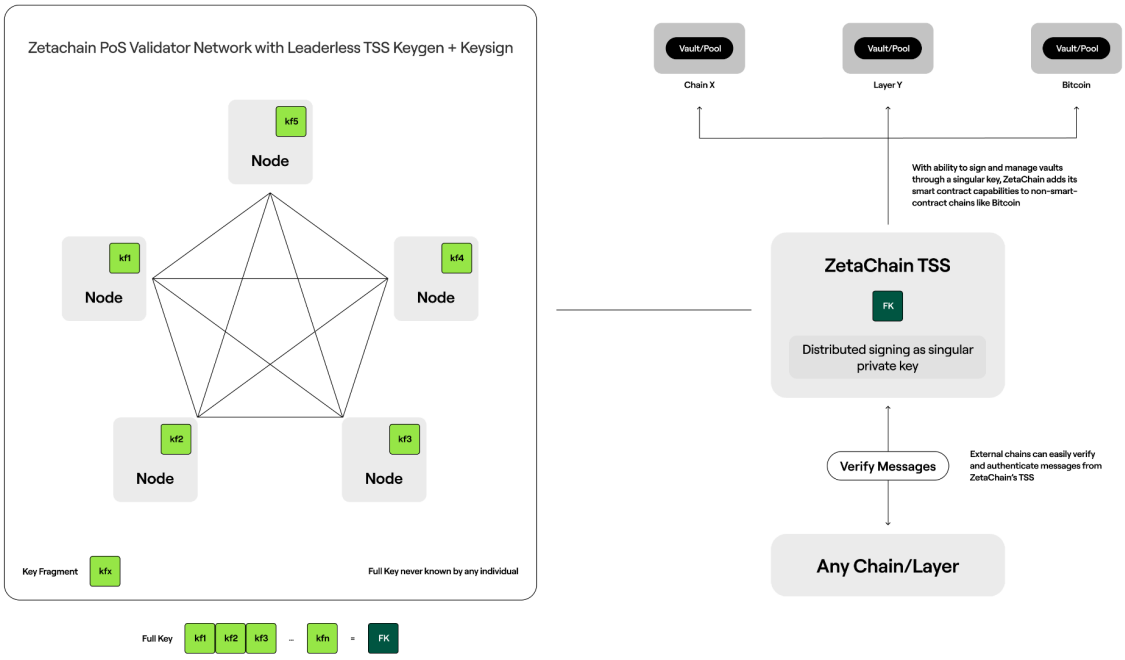

Figure 3. Leaderless TSS Keygen and Keysign Overview

To sign in a decentralized manner, XM employs a multi-party -threshold ECDSA scheme based on[7, 8]. This leaderless Threshold Signature Scheme (TSS) performs key generation and signing in a distributed fashion. That is, no single validator or outside actor has access to the *complete* private key at any point in time, and no private information is leaked in key generation or signing. For efficiency, XM employs batched signing and parallel signing to improve signers throughput.

4.4. Cross-Chain Smart Contracts and Zeta Virtual Machine

The XM hosts an Ethereum Virtual Machine (EVM) compatible execution layer called XM EVM. Aside from supporting all features of EVM and normal interactions with EVM (contract creation, contract interaction, composition of contracts, etc), the distinguishing feature of XM EVM is that

- contracts on XM EVM can be called from external chains
- contracts on XM EVM can generate outbound transaction on external chains

These two additional features make the XM EVM a general purpose programmable platform that supports *cross-chain transactions* that alter states in different chains *atomically* and in *a single step*.

4.4.1. Challenges in General Purpose Cross-Chain Transaction

There are two key challenges in designing a general-purpose cross-chain transaction platform: asynchrony and atomicity.

The first challenge is that communication between chains is necessarily via message passing and inherently asynchronous between heterogeneous chains. This means unlike smart contracts on a single chain (such as EVM), querying or changing the state of another chain is asynchronous. This precludes the common convenient synchronous function calls from cross-chain smart contracts. The cross-chain smart contract programming model thus is best considered as a finite state machine, where state change is triggered by the messages (observations) from external chains. The app contract thus will be structured as a distributed event-driven state machine triggered by messages. This is quite a more complicated programming model from the synchronous model of single chain smart contract.

The second challenge is the atomicity of cross-chain transactions. As cross-chain transactions involve altering states on multiple chains, if one part of the state change fails, all previous state changes need to be reverted. Blockchains reverts are powerful mechanisms to maintain atomicity, but no blockchain is built with consideration such that revert is conditional on what happened on another blockchain. To maintain cross-chain transaction atomicity, any cross-chain solution must adequately handle reverts, otherwise cross-chain applications will be too onerous to reason about and build.

In this paper we explore a viewpoint of hybrid UTXO and account-based approach, playing to the strengths of each. Essentially, we use UTXO to represent and track external blockchain transactions, and use account-based smart contracts for logic and managing shared global states. We treat observed external events as a “synthetic” UTXO. A UTXO includes the amount of ZETA coin (burned), amount of another coin (optional, for example, BTC on the Bitcoin network where it's impossible to issue ZETA coin), and a script msg (roughly equivalent to a message or function call on Ethereum). The smart contract on XM runs the msg and generates an Event that tries to “spend” the UTXO on XM . The Event is then picked up by ZetaClient signers and they will sign a transaction to an external chain. The XM Virtual Machine and ZetaClient will validate certain invariants, one of which is that the output ZETA must be equal to the input ZETA in the UTXO. Once the outbound transaction is confirmed and observed, the UTXO is marked as “spent” and deleted from the state machine. If the outbound transaction fails (insufficient gas, etc.), the UTXO is marked as “revert” and refunds of ZETA and/or associated coins are refunded on the source chain. When the refund is confirmed then the UTXO is deleted from the state machine. See Figure 4 for an illustration.

We use the synthetic UTXO model for its accountability, simplicity, and scalability while avoiding the key limitation of UTXO which is the expressiveness of its scripting, and awkwardness in certain important applications (one TX per block in AMM).

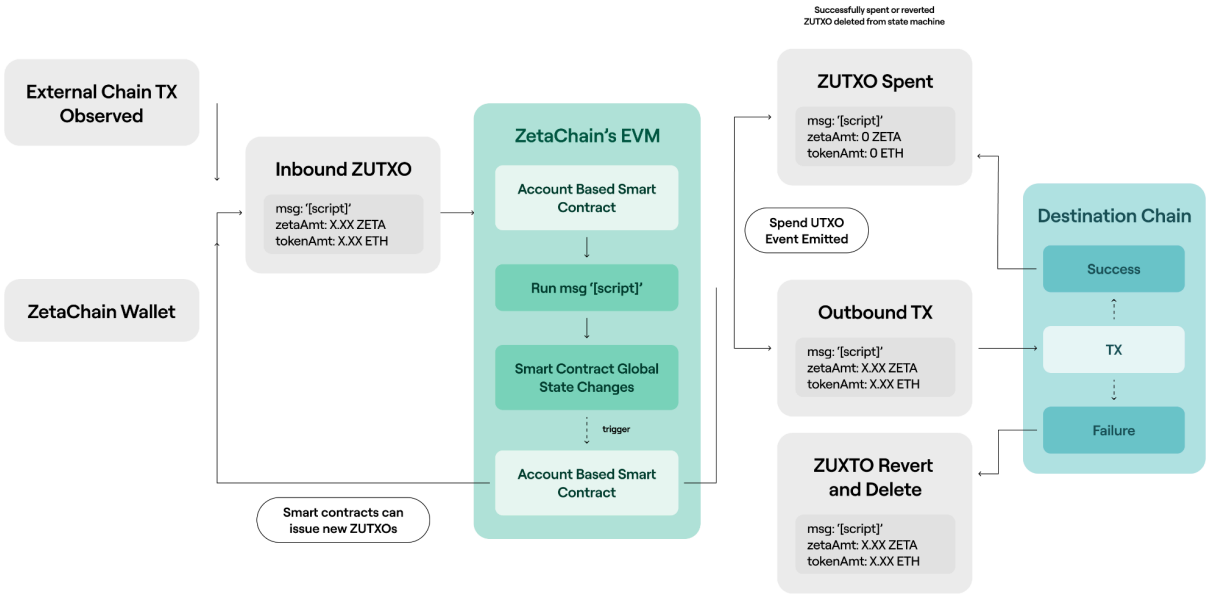

Figure 4. Hybrid UTXO-account flow.

4.4.2. Universal Smart Contract

XM introduces Universal Smart Contracts, a more efficient alternative to typical cross-chain messaging (CCMP) protocols. This model simplifies development by centralizing application logic and state into a single contract on XM 's EVM. This avoids the expense, latency, and complexity of synchronizing state across distributed contracts on multiple chains, a common issue with CCMP that also complicates handling reverts and precludes chains without smart contracts (like Bitcoin).

A universal smart contract is a XM EVM contract that can:

- be arbitrarily programmed.
- directly control assets on external chains.
- be called from external chains.
- call contracts and transfer assets to external chains.

Foreign assets like ETH, BTC, or USDC are controlled by XM 's TSS address on external chains

and are represented on XM EVM as ZRC-20 tokens. Any XM EVM contract can achieve universal functionality by implementing the UniversalContract interface and interacting with ZRC-20s.

```
interface UniversalContract {
    function onCall(
        MessageContext calldata context,
        address zrc20,
        uint256 amount,
        bytes calldata message
    ) external;
}
```

1. Calling a Universal Smart Contract from External Chains

An inbound call is initiated when a user sends assets to a Gateway Contract (or TSS address on a non smart contract chain like Bitcoin) on an external chain with a memo specifying the destination XM EVM contract and a message. XM 's network observes this transaction and invokes the onCall function on the specified contract, populating it with the context of the call (origin chain, sender), the ZRC-20 address of the asset, the amount, and the message. The contract then executes its logic. If the XM EVM execution reverts, the protocol automatically creates a transaction to refund the user.

2. Calling an External Chain Smart Contract from a XM EVM Contract

XM EVM contracts can initiate outbound transactions—both asset transfers and arbitrary function calls—through the Gateway XM EVM contract. This enables two-way communication and orchestration of multi-chain logic.

```
// A simplified interface for XM 's Gateway on zEVM
interface IGatewayZEVM {
    function call(...) external;
    function withdraw(...) external;
    function withdrawAndCall(...) external;
}
```

To make an outbound call, the XM EVM contract must pay the destination chain's gas fee using the corresponding gas ZRC-20 token. The process involves calculating the fee, transferring it from the user, approving the Gateway, and invoking the appropriate Gateway function (call, withdraw, or withdrawAndCall).

```
// Sample zEVM function to call an external contract
function callExternalContract(
    bytes memory receiver, // External contract address
    address gasZRC20,      // Destination chain's gas ZRC-20
    bytes calldata message // Encoded external function call
) external {
    // 1. Calculate gas fee for the outbound transaction.
```

```
    uint256 gasLimit = 100000;
    (, uint256 gasFee) = IZRC20(gasZRC20).withdrawGasFeeWithGasLimit(gasLimit);

    // 2. Collect gas fee from the user and approve the Gateway.
    require(IZRC20(gasZRC20).transferFrom(msg.sender, address(this), gasFee));
    IZRC20(gasZRC20).approve(address(gateway), gasFee);

    // 3. Execute the cross-chain call via the Gateway.
    gateway.call(
        receiver,
        gasZRC20,
        message,
        CallOptions({gasLimit: gasLimit, isArbitraryCall: true}),
        RevertOptions({revertAddress: address(this), ...})
    );
}
```

XM provides robust revert handling. If an outbound call fails, the protocol automatically triggers an onRevert function on a developer-specified contract, allowing for graceful error handling and fund recovery.

3. Example: Cross-Chain Swap Application

A cross-chain swap application demonstrates this model's power, enabling a user to exchange Token A on Chain X for Token B on Chain Y in a single transaction, with all logic managed by one XM EVM contract.

Handling Inbound Swaps (onCall)

The onCall function serves as the entry point, triggered by a user's deposit on a connected chain. It decodes the user's intent, executes the swap logic, and initiates the outbound transfer.

```
// Entry point for incoming cross-chain transactions
function onCall(
    MessageContext calldata context,
    address inputZRC20,
    uint256 amount,
    bytes calldata message // Contains target token and recipient
) external override onlyGateway {
    // 1. Decode message to get the target token and recipient.
      (address targetZRC20, bytes memory recipient, ...) = abi.decode(message,
...);

    // 2. Swap input token for the target token and for gas on zEVM.
          (uint256 outputAmount, address gasZRC20, uint256 gasFee) =
handleGasAndSwap(...);

    // 3. Initiate the outbound transfer of swapped tokens.
```

```
    withdraw(recipient, outputAmount, targetZRC20, gasZRC20, gasFee);
}
```

Executing the Outbound Transfer (withdraw)

After the swap on XM EVM, the withdraw function sends the resulting tokens to the recipient on the destination chain by calling the Gateway.

```
// Sends swapped tokens to the destination chain
function withdraw(
    bytes memory recipient,
    uint256 outputAmount,
    address targetZRC20,
    address gasZRC20,
    uint256 gasFee
) internal {
    // 1. Approve the Gateway to spend the gas and target tokens.
    IZRC20(gasZRC20).approve(address(gateway), gasFee);
    IZRC20(targetZRC20).approve(address(gateway), outputAmount);

    // 2. Call the Gateway to withdraw tokens to the recipient.
    gateway.withdraw(
        recipient,
        outputAmount,
        targetZRC20,
        RevertOptions({revertAddress: address(this), ...})
    );
}
```

This illustrates how the universal model centralizes logic, freeing the developer from managing multi-chain deployments and low-level messaging, while the XM protocol handles the underlying complexity.

Note 1: Contracts that don't need to be called from external chains need not implement the UniversalContract interface.

Note 2: The capabilities of universal contracts are dependent on the available primitives, like the ZRC-20 standard for fungible tokens.

Note 3: Deploying logic and state to a single contract on XM EVM, with protocol-handled reverts, makes building universal dApps significantly easier than message-passing alternatives.

Note 4: This architecture supports non-smart contract chains like Bitcoin, as no contracts need to be deployed on the external chains.

4.4.3. Universal Smart Contracts vs. Messaging

While both mechanisms can support many types of applications, they offer fairly significant differences in the architecture those applications would adopt.

More complicated dApps may prefer Universal Smart Contracts because the logic & state is in a single place, whereas with messaging, you must broadcast messages and state sync across many contracts on different chains, which can lead to more attack surface and more gas fees (each message requires additional gas to be paid, and the number of messages you must send to maintain a full state sync scales). In other words, Universal Smart Contracts behave, for developers, as if all assets were on one chain (see Figure 5).

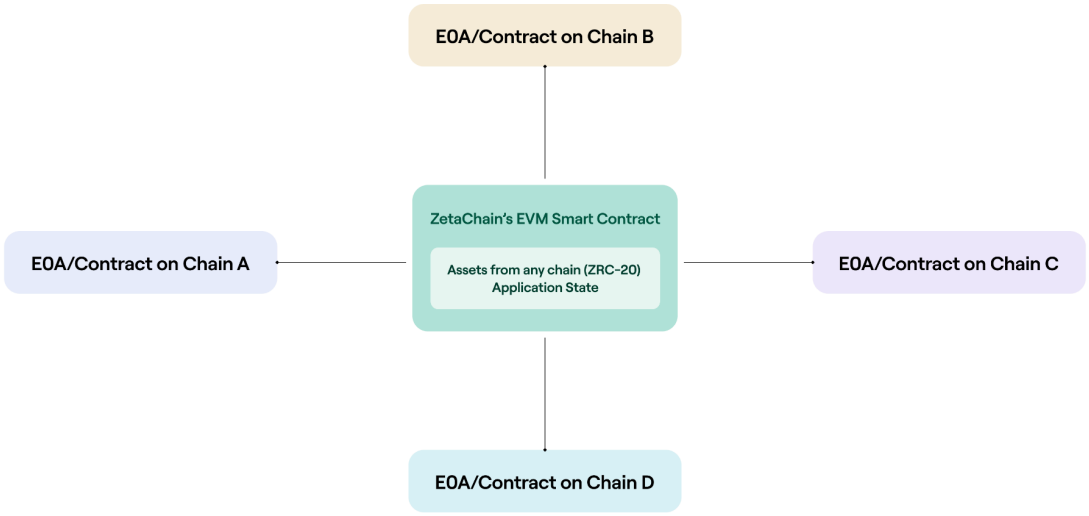

Figure 5. Universal smart contract-based application. Note that there is a single contract that receives input, writes output, maintains state, and orchestrates external assets for the application. The number of external transactions required for a Universal dApp increases only based on the required outbound transactions, like withdrawing assets to an external chain's address

Common applications like Uniswap V2/V3, Curve, Aave, Compound, and so on that have been audited and battle-tested on Ethereum/EVM can easily be deployed and built on top of in XM 's Universal Smart Contracts. One can extend these applications by adding in compatibility with ZRC-20, but those changes are minimal and the majority of logic may remain the same, and users may interact with these applications in single-step transactions just as they would on Ethereum (or by calling them from external chains). On the other hand, with messaging, in many situations (especially those that are more complex),

a developer must recreate the logic in a completely different, asynchronous messaging and state-sync system; messaging cannot leverage existing work in the same way.

ZRC-20 can easily support Bitcoin/Cardano/XRP which do not have capability or efficiency to support general purpose smart contracts for applications like swapping, lending, etc. Messaging cannot work for these chains, because messaging requires smart contracts on any connected chain.

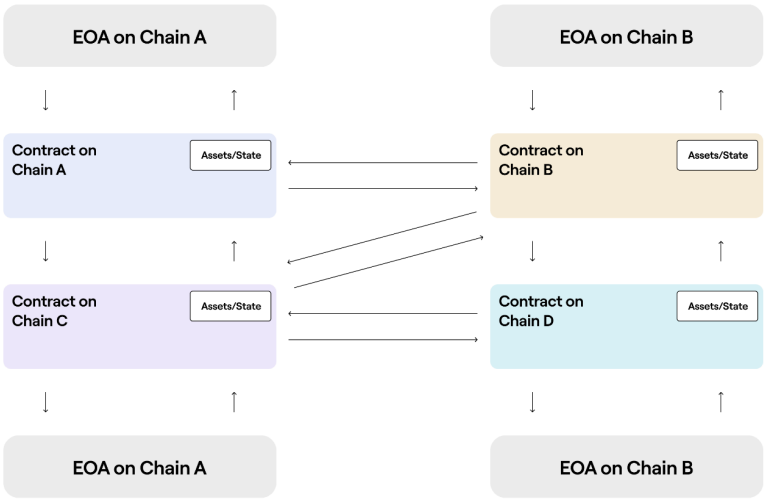

Figure 6. Messaging-based application. Note that for contracts to stay in sync across connected chains, the number of messages required increases exponentially with the number of chains involved.

Messaging generally makes sense in simpler use cases between 2 or just a few chains, or where state should heavily be based in one chain, and sent or interacted with from other chains. Application-specific bridges, for example, where the goal is simply to get data/value into one chain, could make sense to build with messaging. Applications that must utilize contracts on external chains may also need a messaging-based component. For more complex applications, the number of messages (and thus gas/transactions) required to synchronize state across multiple chains can increase exponentially with the number of chains involved (see Figure 6). For example, managing a vault or lending protocol with assets across many chains could be difficult to manage with just messaging.

Message passing style logic and state are distributed on asynchronous chains which adds significant complexity to maintaining cross-chain transaction atomicity, and forces dApps to program in an event (message) driven way that is generally harder to do than synchronously in a single chain. Universal smart

contracts on the other hand offer the novel ability to develop multichain applications in a more synchronous, atomic environment as if they were on one chain.

4.4.4. Fees & Gas

To prevent spam and ensure fair and efficient use of the blockchain resources (compute and storage), the user must pay proper fees for processing the cross-chain transaction.

Unlike transactions on a single chain, a cross-chain transaction naturally might involve several different gas assets and need to pay more than one type of tokens for gas fees. This is rather inconvenient, and may add undue operational cost or risk to operate the cross-chain solution. For example, if one invokes a contract on Ethereum from BSC chain, the user needs to pay both BNB and Ether as gas fees; but how can the user pay Ether on BNB? Do they need to acquire "wrapped" Ether on BSC? Which version of the wrapped Ether? Who wraps and unwraps the Ether?

Alternatively, one might just ask the user to pay in a single asset (for example, only BNB), and then some off-chain service converts the BNB into Ether to reimburse the protocol which needs to pay Ether for the outbound tx processing. This is quite an operational burden, and runs counter to the autonomous nature of sovereign blockchain that does not need centralized operator.

XM completely automates the gas handling and conversion on-chain, and with market force to maintain proper conversion rate. Also, the conversion of different gas assets are synchronous with the CCTX itself so the settlement is as fast as possible. The way XM does it is to rely on ZRC20 and their AMM pools on XM EVM (currently Uniswap v2 pools). All gas assets have a corresponding ZRC20 which pairs with ZETA (native gas token on XM EVM) on XM EVM.

Let us consider the two cases when the user needs to pay gas fees in coins they might not have:

- In message passing, the user pays a single asset (ZETA token) for all gas fees. The XM protocol converts proper amounts of ZETA into outbound chain gas asset ZRC20 synchronously and use the balance to pay outbound transaction gas fees.
- In universal smart contract ZRC20, when a user (or a smart contract) wishes to withdraw the foreign asset, the user will need to pay the outbound gas fee. The withdrawing smart contract can acquire the outbound chain gas asset ZRC20 from the internal AMM pools on XM EVM to pay gas synchronously.

In either case, the multi-gas handling of XM is sound (which means that the protocol always has enough gas asset to pay outbound tx gas fees), and the conversion rate is determined by market force. As XM EVM ZRC20 assets are easily withdrawn to external chain with on-chain contracts, the markets on XM EVM are connected with other markets therefore we can expect market forces to maintain price parities.

5. Use Cases & Applications

In this section we discuss some sample applications of XM . These examples are not anywhere near comprehensive, since the general smart contract and interoperability capabilities of XM provide a platform for virtually unlimited creativity in terms of universal application-building.

5.1. Smart Contract Managed External Assets

A powerful feature of smart contracts is that smart contracts can hold any assets that a normal account can hold, and are able to receive and spend that asset according to programmed logic. However, important blockchains like Bitcoin, Dogecoin, Monero, etc., do not have general enough smart contract capability to support useful applications such as AMM exchanges, collateralized borrowing/lending markets with pools, and the like. There is currently no way to involve native Bitcoin (without wrapping) in arbitrary logic in a decentralized and permissionless manner. The cross-chain smart contract capability of XM can hold and use assets on external chains directly, therefore enabling smart contract managed native Bitcoin on XM , among other native assets such as ETH, ERC20, Algorand ASAs, etc. Furthermore, through XM smart contracts and with message passing, cross-chain dApps can be easily composed with smart contracts on the participating chains, with XM smart contracts managing native Bitcoin vaults.

Let us look at an example in some detail. The mechanism for XM smart contracts to manage BTC on Bitcoin is as follows. The initialization of smart contract requests `KeyGen` to generate a TSS key which acts as the address of a Bitcoin vault. The ZetaClient will monitor the TSS address and upon identifying incoming transactions to the TSS vault, it parses the data from the Bitcoin transaction in OP_RETURN and invokes the `zetaProcess` function with the parsed data on the smart contract. The smart contract takes actions accordingly (such as credit to certain accounts, sending out another asset according to AMM pricing, etc.). To send out Bitcoin from the smart contract, the smart contract emits a specific `Event` that the ZetaClient will pick up and sign & broadcast to the Bitcoin network. The smart contract must also implement a function `zetaExternalTxConfirm` which will be invoked when the outbound external chain transaction is mined.

5.2. Universal Stablecoins

Stablecoins (blockchain-issued tokens pegged to real currencies) are gaining significant traction. However, multiple issuers often issue stablecoins for the same currency (e.g., USD), and a single issuer like Tether or Circle may issue the same stablecoin across various blockchains. Users face challenges in selecting a blockchain for storage and then moving stablecoins to the needed chain quickly, cheaply, and reliably.

XM enables a simple abstraction for a universal USD, backed by multiple USD stablecoins from different blockchains, while maintaining fluidity across chains (on-demand redeemability to different chains). This solution on XM will be fully decentralized and programmable, enhancing transparency and trust.

One method for building a universal USD Stablecoin, backed by, for example, 16 USDC on chains X, Y, Z, etc. (connected to XM), involves the following: First, USDC.X, USDC.Y, and USDC.Z have their respective ZRC20 representations on XM (an ERC20-compatible token contract with

specialized input/output functionality to/from connected blockchains X, Y, Z…). A portion of USDC.X, USDC.Y, and USDC.Z liquidity is pooled in an AMM contract like CurveStableSwapNG, optimized for capital-efficient swaps between stable asset groups. The liquidity position (LP token) of such a pool can serve as the unified USDC, exchangeable for any USDC.XYZ.

Furthermore, since ZRC20 contracts are natively invokable from connected blockchains, the user experience can be improved by providing a deposit function from chains XYZ: USDC.X -> Unified USDC in one cross-chain transaction. Similarly, ZRC20 USDC.X can be redeemed back to USDC on chain X natively via a cross-chain transaction. Therefore, the unified USDC can represent a unified USD coin fluid across various USDC issues on different chains (X, Y, Z…), available to users on-demand. All infrastructure is decentralized, requiring no centralized party. See Figure 7 for an illustration.

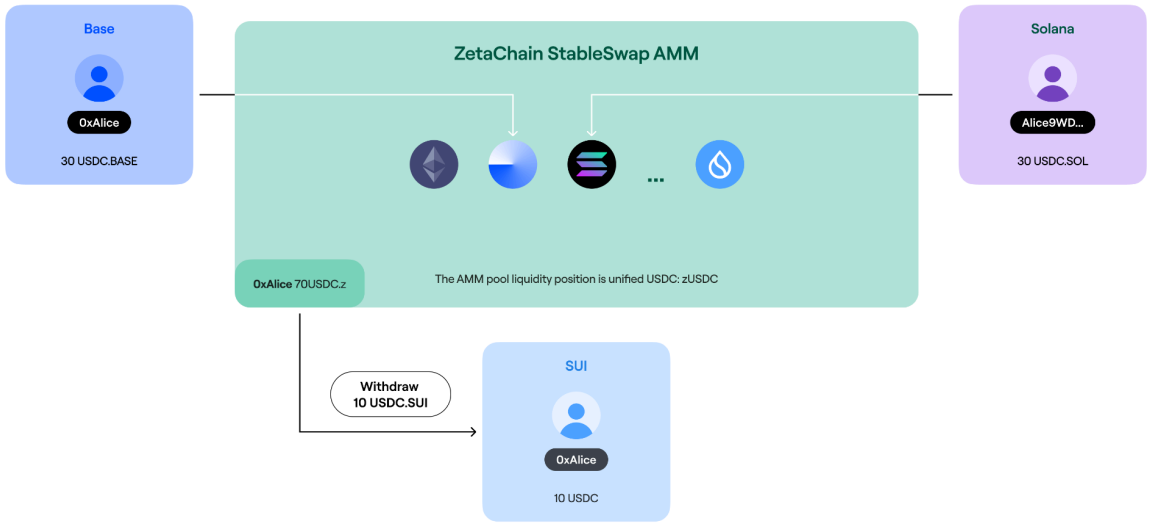

Figure 7. Universal stablecoin on XM . Liquidity from multiple USD-pegged assets on different blockchains (e.g., USDC.X, USDC.Y, USDC.Z) is represented as ZRC-20 tokens and pooled in an on-chain AMM to form a unified stablecoin on XM . This universal USD can be freely swapped or redeemed natively to any connected chain in a single decentralized transaction.

5.3. Cross-chain AMM Exchanges

XM can enable true cross-chain AMM decentralized exchanges, built on top of smart contracts. There are two ways of constructing an AMM DEX on XM : message passing and native XM smart contracts. The key difference is whether the pool is managed by an external smart contract or native XM smart contract. With message passing, the asset pool is managed by smart contracts on external

chains; with the native XM smart contract approach, the pool is managed by XM smart contracts through a TSS account.

Specifically, in message passing, the assets are managed by smart contracts on external chains, paired with a ZETA coin. A swap of asset X on chain A for asset Y on chain B can be accomplished by: 1) swap X for ZETA on chain A using smart contract managed pool and AMM; 2) pass message, together with the ZETA coin from chain A to chain B; 3) chain B smart contract managed pool (Y/ZETA) swaps ZETA coin for Y.

With native XM smart contracts, the XM TSS account holds all the native assets on external chains, which can be managed by XM contracts directly. The XM smart contract implements AMM logic that determines pricing, swap, liquidity providers, and fees.

In the message passing approach, the dApp states and logic are spread across all the external chains; XM only acts as a message verifier and relayer. The advantages in this approach is that existing infrastructure can be reused (for example, on EVM chains Uniswap contracts can be reused to manage pool X/ZETA), and the dApp needs only to handle the cross-chain messaging to implement conditional execution. On the other hand, in the native XM smart contract approach, the logic and state of the dApp lives on XM , a single platform with a unified interface to interact with external chains. The advantages in this approach are the ease of dApp development (minimal development efforts in accommodating new chains), and flexibility (no longer constrained to chain idiosyncrasies and message-passing cross-chain interaction). Additional benefits are that it relies on smart contracts on external chains minimally, so complex logic can work on not only smart-contract chains but also UTXO chains like Bitcoin.

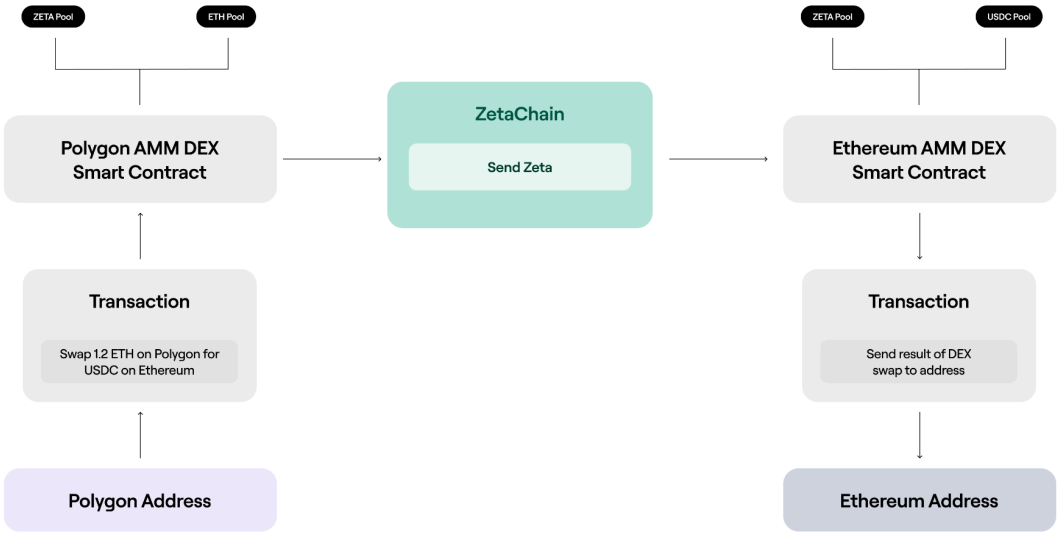

Figure 8. DEX built with XM message passing. Leveraging external chain smart contract DEXs, one can build a cross-chain swap by sending messages with ZETA.

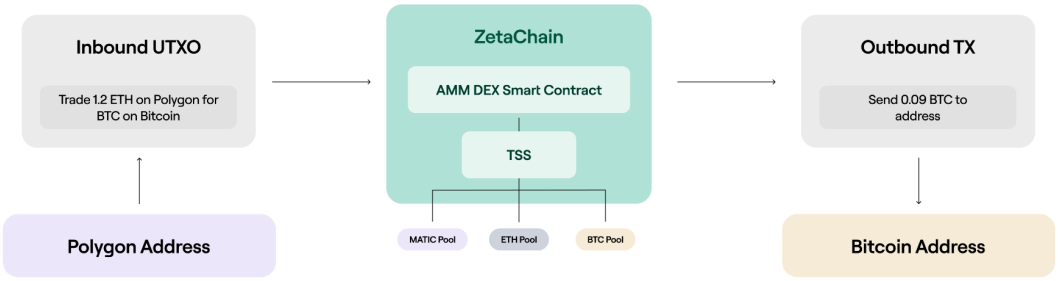

Figure 9. DEX built with XM Smart Contracts. Since XM TSS can manage external chain pools with its smart contracts, DEX can even support non-smart-contract chains and assets where transactions are simple and single-step.

5.4. Cross-chain message passing with value/data

The ability to reliably and securely pass messages from one chain to another can enable powerful cross-chain applications, even without native XM smart contracts. The message passing functionality consists of communication endpoints on all external chains. The XM validators serve as a Byzantine Fault Tolerant notary that attests the validity of events/transactions on chain A to chain B, and as a relayer of messages. Chain B's smart contract only needs to whitelist the TSS address of XM in order to trust that XM has verified the events on chain A. This allows conditional execution on chain B's contract depending on transactions/messages from chain A, which opens a wide range of cross-chain dApps, such as AMM DEXs, NFT, etc. (see more below). An important and convenient feature of the XM system is that the messages can be attached with value in the form of the ZETA coin (natively cross-chain), which considerably simplifies dApps which require moving the value cross- chain.

The messaging service of XM consists primarily of interface contracts on the connected chains. To access the message passing service, a dApp needs to deploy a smart contract on both the source chain and destination chain. On the source chain, the sending smart contract can invoke a `zeta.MessageSend` function with the following information: sending address, destination chain id, destination contract address, ZETA coin to transfer, gas limit on destination chain, contract message for destination transaction (binary or JSON encoded payload), and transaction index. The sending contract must implement a `zetaMessageRevert` function, which will be called by XM when the destination message delivery and processing of a transaction fails (for example, due to out of gas, out of funds, invalid message, etc.). Upon failure, the XM system will refund the ZETA coin to the sending address (less gas fees), and invoke the dApp contract `zetaMessageRevert` function which is supposed to revert application actions (unlocking a locked NFT, for example). On the destination chain, the dApp contract must implement a function `zetaMessageReceive` which takes the same parameters as the sending`zeta.MessageSend`, and can perform application logic (such as minting an NFT that has been locked on the source chain). The destination contract will also receive a ZETA coin (less gas fee), which can be used as a value transfer cross-chain.

Message passing can enable a variety of important applications such cross-chain DEX, borrowing/lending, multi-chain NFT, etc.

5.5. Multi-chain NFT

Non-fungible Token (NFT) is an emerging concept that has found use in art collection, gaming, event tickets, and many other applications. In contrast to fungible tokens such as ETH, BTC, or ERC-20 tokens, each NFT is unique and not interchangeable with another NFT in the same collection. This non-fungibility can be essential in applications such as art, real-estate, etc. On Ethereum, for example, the most common NFT standards are ERC-721 and ERC-1155. In ERC-721, an NFT is basically a tuple `(contractAddress, tokenId)`. The smart contract that issues the NFTs keeps track of the owners of each NFT in a map `owner=>tokenId`. The NFT can be transferred from one owner to another, and each NFT owner can be queried.

In a multi-chain NFT world, where the same collection of NFTs are issued on multiple chains (such as Ethereum, Flow, Solana), and one NFT can transfer to another chain, a challenge in the bridge model is the knowing the provenance of a given NFT – who is the owner of a given NFT now that the NFT could be on one of multiple chains and where are the records of the transactions of the transfers? This problem can be solved by XM smart contracts which facilitate cross-chain ownership transfers of NFTs. It can work as follows. Each chain will have an escrow smart contract controlled by the XM key. To transfer an NFT to another chain, one transfers the NFT to the escrow, pays transaction fee in ZETA coin, and XM will mint the NFT on the destination chain. The smart contract on XM keeps track of the owner and blockchain where the NFT is at any given time. While there have been experimental cross-chain NFT bridges, having a decentralized issuing authority allows an NFT to be natively cross-chain, making it simpler and feasible to create, verify, and exchange NFTs cross-chain.

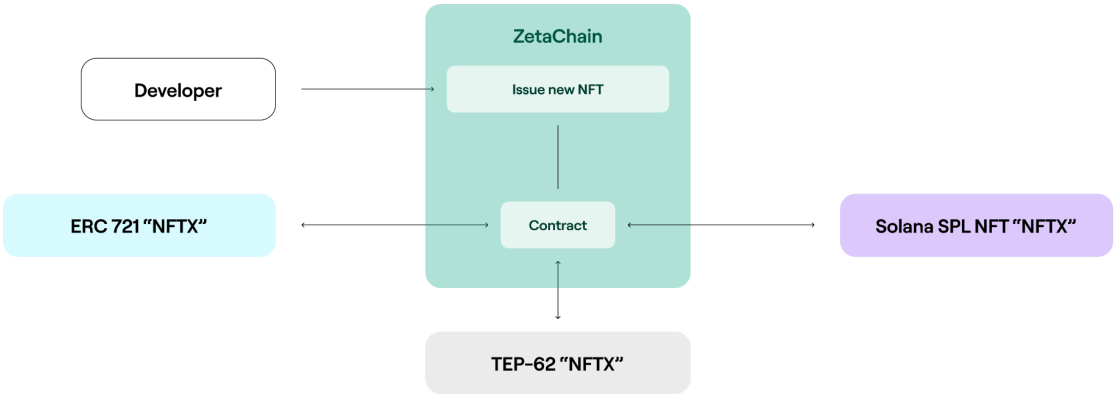

Figure 10. Multi-chain NFT. With a decentralized issuing authority (XM TSS), one can have an NFT that is easily sent between chains, where ownership and current location are easily verifiable.

5.6. Other Use Cases

These are just a few other potential use cases of XM . Given XM is a general smart contractplatform, you can also imagine that any application you deploy on a single blockchain/smart contract platform can be expanded to operate across all connected chains.

5.6.1. Universal Payments

A system that lets users/EOAs send payments from/to any asset on any chain. This can help vendors and customers have a decentralized, universal, and accessible payments route that doesn't require users to have a hyper-specific set of assets on a specific chain.

5.6.2. Universal Identity and Assets

Identity system, name service, or Soul Bound Tokens that can serve as identities across all chains. With universal capabilities, identities can interact with other chains agnostically and in a future-proof manner as XM adds support for more chains. Users need not have individual identities/domains per chain, and can utilize their assets (gaming, collectibles, fungible tokens, etc.) from all chains from a single place.

5.6.3. Multi-chain, Multi-signature vaults