Chain Interoperability

Vitalik Buterin

September 9, 2016

Chain Interoperability

Whereas in the first few years of the blockchain industry one may be forgiven for thinking that there would be only "one blockchain to rule them all", in recent times such a possibility has been receding further and further from reality. Within the public blockchain space, different projects have been staking out different regions of the tradeoff space between security, privacy, efficiency, flexibility, platform complexity, developer ease of use and even what could only be described as political values. In the private and consortium chain space, the notion that there exist different chains for different industries - and even different chains within the same industry - is even less controversial and arguably universally understood as obvious. In such a world, one natural question that emerges is: how do these chains interoperate? One of the advantages of using platforms where cryptographic authentication is naturally baked into every single operation is that we can actually provide much tighter and more secure coupling between platforms than is possible with previously existing systems. We can go far beyond the approach most common in centralized systems of simply having an API from one chain to the other, and in some cases even go so far as to have smart contract code on one chain verify the consensus finality of events on other chains directly, requiring no trust in intermediaries at all.

Interoperable chains open up a world where moving assets from one platform to another, or payment-versus-payment and payment-versus-delivery schemes, or accessing information from one chain inside another (eg. "identity chains" and payment systems may be a plausible link) becomes easy and even implementable by third parties without any additional effort required from the operators of the base blockchain protocols. So far the notion of chain interoperability has seen much theory and little practice, primarily because a live example of successful chain interoperability requires not one, but two, already existing, stable and sufficiently powerful blockchains to build off of, but this is slowly starting to change.

Research in chain interoperability, up until this point, has largely been undertaken in a public blockchain context, and as this document is to some extent a "literature review" of work done so far it does reflect this. Although many of these techniques are just as applicable to private and consortium chain settings, such settings will also have their own unique challenges that are outside the scope of this document. For example, there has not yet been a single permissioned chain that has seen substantial use or adoption, and so it is not fully clear to what extent the consensus algorithms and other features of current experiments will translate. Additionally, a large part of "interoperability" will necessarily entail interoperating with traditional systems and protocols (SWIFT, SEPA, FIX, etc); this research is unfortunately not dual-purposeable for public blockchains because it seems very unlikely that public chains will be connected to such systems directly in the near or medium-term future.

Types of Interoperability

From a technical perspective, there are three primary categories of **strategies** for chain interoperation:

- **Centralized or multisig notary schemes**, where a party or a group of parties agree to carry out an action on chain B when some event on chain A takes place.
- **Sidechains/relays** (systems inside of one blockchain that can validate and read events and/or state in other blockchains)
- **Hash-locking** (setting up operations on chain A and chain B that have the same trigger, usually the revelation of the preimage of a particular hash)

There are also several potential *use cases* that interoperability can achieve:

- **Portable assets** - for example, being able to take one unit of Fedcoin (a hypothetical government-issued digital asset)[1] from its "home ledger" that is ultimately authoritative on its ownership, securely move it to another chain, trade it, use it as collateral or otherwise take advantage of it on that chain, and be confident that the option to move the Fedcoin back to its home ledger is always available if desired (ie. trust-minimized 1-for-1 backing).
- **Payment-versus-payment or payment-versus-delivery** - in technical circles, this concept is also often called "atomic swap", where "atomic" is used in the word's original Greek meaning of "uncuttable", ie. there is a guarantee that either both transfers happen or neither transfer does. Essentially, the goal is to allow user X to transfer digital asset bundle A to user Y in exchange for Y transferring digital asset bundle B to user X (where A and B are on different chains, and X and Y have accounts on each chain), in a way that is guaranteed to be atomic and secure.
- **Cross-chain oracles** - for example, one might imagine a smart contract on one chain that performs some action with some address only when it receives proof that an identity oracle on another chain specifies that the address is a particular unique identity. Note that the chain that is being read does not change over the course of this kind of interoperation event.
- **Asset encumbrance** - lock up asset bundle A on chain X and have locking conditions be dependent on activity on chain Y. Use cases include liens, collateral in financial derivatives, bankruptcy clawbacks, court orders and various use cases involving security deposits.
- **General cross-chain contracts** - for example, paying dividends on chain X to holders of an asset whose ownership is registered on chain Y.

Out of these use cases, the two that have received the most attention are cross-chain digital assets and cross-chain exchange (ie. payment-versus-payment and payment-versus-delivery); however, later

[1] The word "Fedcoin" is shorthand for a number of different schemes that revolve around the concept of "fiat currency issues and secured using (some) cryptocurrency technology"; some proposals include David Andolfatto's Fedcoin proposal (http://andolfatto.blogspot.ca/2015/02/fedcoin-on-desirability-of-government.html) with cash-like privacy properties, as well as central bank-based digital currency (CBDC) schemes run on top of permissioned ledgers that would present an interface to consumers more similar to the traditional banking system; see the recent Bank of England papers for explanations of such a scheme's possible benefits. The concept of Fedcoin itself implies little about the details; it could be run on a public chain or a consortium chain, have varying levels of access restriction or KYC, etc.

sections will describe solutions to the problems of interoperability in generic terms that are applicable to all applications and then zero in on security and other concerns related to cross-chain portable digital assets and cross-chain asset exchange specifically. However, it's worth noting that cross-chain asset portability is only one way to accomplish its desired objectives; in the world of unbacked cryptographic-only assets such as BTC and ETH, it is indeed the only way to achieve the desired objective, but if we are dealing with issuer-backed assets then alternative approaches (eg. separately issuing assets on multiple chains, and having the implied convertibility be at the legal layer rather than the blockchain layer) are also possible.

The word "**sidechain**" is often used in reference to cross-chain portable digital assets, although much of the use of this language is misleading and confusing in several ways. First of all, Blockstream's formal definition of a "sidechain" is that "a sidechain is a blockchain that validates data from other blockchains."[2] However, this language is very expansive; under this definition, thanks to BTCRelay (described in a later section), Ethereum is a sidechain to Bitcoin already[3]. In normal discourse, the term "sidechain" is more frequently used to refer to what Blockstream calls a "pegged sidechain", where the functionality of a blockchain reading data from other blockchains is used to facilitate cross-chain asset portability. This is a more advanced step; it requires either both chains to be sidechains of each other or the existence of a trustworthy federation (see the later section on notary schemes), as well as a scheme layered on top of such a cross-chain communication mechanism that actually implements the cross-chain asset portability logic.

Second, the phrase "chain A is a sidechain of chain B" implies a relationship of subservience that in many cases cannot be reasonably said to exist; to re-use an example from above, Ethereum is technically now "a sidechain to" Bitcoin, but Ethereum is clearly not subservient to Bitcoin in any meaningful way. The phrases "sidechain" and "pegged sidechain" originally came about in a context where a blockchain is home to a single dominant asset, and so it was natural to view a chain *containing a pegged token* as being a "*pegged sidechain*". In reality, however, "pegged sidechaining" is a property of individual assets on top of blockchains much more than it is a property of blockchains themselves.

For example, the Ethereum blockchain contains ether, an asset native to Ethereum itself, but if other blockchains implement the appropriate required protocol changes (or if a trustworthy federation emerges), then Ethereum may well end up containing an asset "e-BTC", backed 1-for-1 by BTC on the Bitcoin blockchain, e-DOGE backed 1-for-1 by DOGE on the Dogecoin blockchain, etc; hence, Ethereum would be a "sidechain" to all of these blockchains simultaneously. However, it is clear that the *Ethereum blockchain* itself is not "pegged" to anything in any meaningful sense as a result of these applications existing. Hence, it is arguably better to use phrases such as "chain B can read chain A", "a relay of chain

[2] http://blockstream.com/sidechains.pdf

[3] Even more trivially, a sofa is also a sidechain:
http://www.ic.unicamp.br/~stolfi/EXPORT/projects/bitcoin/posts/2015-06-10-my-sofa-is-a-sidechain/main.html

A exists on chain B" or "D is a cross-chain portable digital asset with home ledger A that can also be used on chain B" rather than talking about whether or not a given chain is a "sidechain".[4]

Notary schemes

The technologically simplest way to facilitate most cross-chain operations is through the use of notary mechanisms. In a notary mechanism, a trusted entity or set of entities that is trusted as a group is used in order to claim to chain X that a given event on chain Y took place, or that a particular claim about chain Y is true. Such entities may be active, listening and automatically acting based on events in some chain, or reactive, issuing signed messages only when asked. The most advanced effort that has taken steps in this direction is the Interledger project[5] developed by Ripple. Interledger, at least in what it describes as "atomic mode", uses a Byzantine-fault-tolerant consensus algorithm in order to achieve consensus among a set of notaries on whether or not a given event took place, and then issues a signature that can be used to finalize payments conditional on this consensus[6].

Interledger also envisions the notion of a *payment chain* where this notary mechanism can be composed. If parties X and Y desire to make an exchange of digital asset bundles, but these bundles exist on chains A and F, where A and F have no direct link, then one can find intermediaries on intermediate ledgers B, C, D, and E, where each pair of adjacent intermediate ledgers does have a direct link (ie. there exist notaries and exchange opportunities between A and B, B and C, etc), then one can determine a bundle of exchanges that will satisfy A and F's preferences while at the same time earning small arbitrage revenues for the intermediaries, and then use a single consensus process for the entire exchange, ensuring that either all transfers happen or none do.

Note that this is not the only way to accomplish such atomicity; hash locking (described in a later section) also accomplishes this function, and is used for that role for cross-chain transfers in the Lightning Network[7]. However, it is relatively technologically simple, and accomplishes its desired goal under its trust model (namely, that less than some fraction of chosen notaries are Byzantine). The set of

[4] A further concept that often gets confused with sidechains is merge-mining, where one chain leverages another chain's consensus. This *is* arguably a relationship of subservience, but is completely independent from the discussion about cross-chain digital asset portability. Chains A and B can be merge-mined but have no cross-chain portable assets between each other, or they may share many cross-chain portable assets, perhaps even some home-based on A and some home-based on B, without any kind of merge-mining relationship.

[5] Note: many statements on Interledger come from old/outdated information; Interledger's latest protocols are based entirely on the hash-lockcing trust model rather than M-of-N oracles.

[6] Similarly to BTCRelay with Ethereum (see later sections), the Interledger team is looking at implementing Ripple-Bitcoin cross-chain trade as an initial test case; More recent developments in the Interledger project have favored a direction more similar to what is described in this document as "hash locking".

[7] http://lightning.network

notaries could be determined individually for a particular exchange; one can imagine a negotiation protocol where all participants submit their trust list and the intersection of all parties' trust lists is agreed upon as the notary set for that exchange[8]. One can also imagine notaries being used in combination with schemes described below, creating an inter-chain exchange protocol where the security of the exchange tries to achieve the highest security level possible, but falls back to notary schemes if the underlying chain does not yet support a more trustless relay mechanism.

Another related scheme is the notion of a *federated pegged sidechain*. The intention here is to have a cross-chain-movable asset, where pegging in one or both directions is accomplished by means of a multisig scheme. This has been implemented in Liquid, a BTC-backed sidechain created by Blockstream[9], where one can move BTC into a multisig address controlled by a federation of participants and then receive a token which we can call "L-BTC", conjured into existence on the Liquid chain once the Liquid chain consensus sees that the BTC transaction has taken place. L-BTC can then be freely traded on Liquid, and one can also destroy L-BTC, at which point the federation that controls the multisig will send an equivalent amount of BTC from the multisig to the party that destroyed the L-BTC (ie. 1-for-1 backing).

Note that one can have a federated peg between two public chains, or between a public chain and a consortium chain. In the latter case, it is arguably simplest to have the entities that make up the consortium also be the entities that control the multisig. In the former case, the scheme arguably only has value if the "sidechain" is also home to applications independent of that particular pegged asset; otherwise, the use of public chain consensus only adds expense with no corresponding improvement to the system's actual security properties, which inherently rely on the honesty of the federation as part of the trust model.

Relays

Relays are a more "direct" method for facilitating interoperability[10], where instead of relying on trusted intermediaries to provide information about one chain to another, the chains effectively take on the task of doing that themselves. The general approach is as follows. Suppose that a smart contract executing on chain B wants to learn that either a particular event took place on chain A, or that some

[8] There are subtleties in designing such protocols; for example, one may want to require the intersection of the trust lists to make up a majority of each individual trust list, as otherwise an adversary that knows a small portion of corrupt notaries within their counterparty's trust list could issue a small trust list and thereby try to trick their counterparty into accepting an exchange that is adjudicated by only those notaries.

[9] https://blockstream.com/2015/11/02/liquid-recap-and-faq/

[10] See http://btcrelay.org/ for the first live running example of a relay, between Bitcoin and Ethereum.

particular object in the state of chain A contained some value at some particular time. Suppose also that chain A is designed similarly to Bitcoin or Ethereum in that it has a notion of “blocks” and “block headers”, where a “block header” is a compact piece of information that “represents” the block (and possibly state data) in some cryptographically authenticated way, most likely using Merkle trees[11].

We can create a contract on chain B that takes one of these block headers of chain A, and uses the standard verification procedure for chain A's consensus algorithm to verify this block header - in proof of work, this would involve verifying that a sufficiently greater amount of proof of work has been generated for the given header than for any conflicting header[12], and in traditional Byzantine-fault-tolerant consensus algorithms that are popular in consortium chains it would consist of verifying that 2/3 of validators' signatures have signed the block header. Once the relay has verified that the block header has been finalized, the relay can then verify any desired transaction, or account/state entry, separately by verifying a single branch of the Merkle tree against the block header.[13]

This use of this so-called “light client verification” technology is ideal for relays because of how fundamentally resource constrained a blockchain is. In fact, it is impossible for a mechanism inside chain A to fully validate chain B and a mechanism inside chain B to fully validate chain A at the same time, for the same simple mathematical reason why two boxes cannot simultaneously contain each other: A would need to re-run the part of B that re-runs A, including the part of A that re-runs B, and so forth. With light client verification, however, a protocol where chain A contains small pieces of chain B and chain B contains small pieces of chain A that are pulled on-demand is entirely feasible. A smart contract on a relay on chain B that wants to verify a particular transaction, event or state information on chain A would, much like a traditional light client, verify a branch of the cryptographic hash tree of chain A, then verify the block header that the root of this branch is inside, and if both checks pass it would accept that the transaction, event or state information is correct (note that because blockchains are fully self-contained environments and have no natural access to the outside world, the relevant bits of chain A would need to be fed into chain B by a user; however, because the data is in a cryptographic sense "self-verifying", this user that feeds this information in need not be trusted)[14].

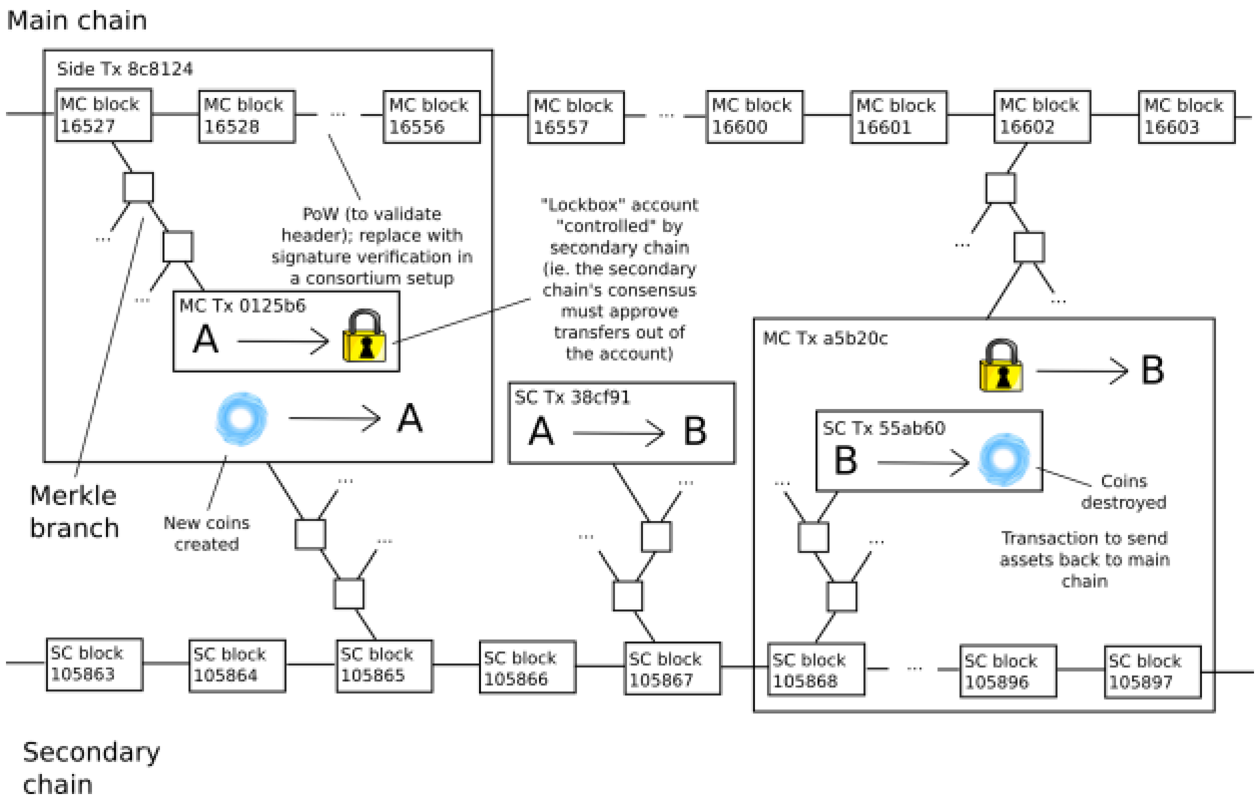
[11] See https://github.com/ethereum/wiki/wiki/Patricia-Tree, https://easythereentropy.wordpress.com/2014/06/04/understanding-the-ethereum-trie/, https://wiki.ripple.com/Hash_Tree and https://en.bitcoin.it/wiki/Protocol_documentation#Merkle_Trees for more information on Merkle trees as used in Bitcoin, Ripple and Ethereum.

[12] A “conflicting header” in this context means a header B which is not an ancestor or a descendant of header A; this implies that A and B are on different forks, and so in a proof of work model, the process of verifying A involves verifying that forks containing A have a much higher “total difficulty” than any forks not containing A.

[13] This use of Merkle tree technology was originally pioneered by Satoshi Nakamoto in the Bitcoin protocol design, the original use-case being facilitating rapid validation of transactions by resource-constrained *clients*; see the section on “simplified payment verification” in https://bitcoin.org/bitcoin.pdf for details.

[14] The security of “light client verification” is not absolute; see later sections.

Relays are very powerful; they can be used for asset portability, atomic swaps or any other more complex use case essentially without restriction. An asset-portability system on top of a relay would look as follows:



Note that in general, the complicated cryptographic verifications behind relays can easily be abstracted and made invisible to developers. Event verification itself can be made into a smart contract that other contracts can call as an event verification oracle. Event reading can be abstracted into an asynchronous operation: one can imagine a cross-chain smart contract programming language containing a primitive createEvent(destinationChain, params), which registers an event and assigns it a unique ID, and a function onReceiveEvent(senderChain, params) { ... }, which can be called only if a cryptographic proof of the event is passed in, and when such a proof is provided it saves a record in storage preventing the function from being called with the same event again.

One may note that this asynchronous event reading model can be viewed as a hybrid of an Ethereum 1.0-style stateful blockchain scheme and an "unspent transaction output" model similar to Bitcoin, where the equivalent of a UTXO is an unconsumed event record; the above asynchronous event architecture assumes that events are consumed quickly, but one can also imagine programming languages where events are not consumed until needed, and unconsumed events become a very meaningful part of long-term state.

A sketch of the cross-chain portable “Fedcoin” code might look as follows:

```
function sendCrossChain(destChain, to, value) {
        if (balances[msg.sender] < value) throw;
        createEvent(destChain, {name: SEND, to: to, value: value});
        balances[msg.sender] -= value;
        crossChainBalances[destChain] += value;
}

function onReceiveEvent(senderChain, params) {
        if (params.name == SEND) {
                if (crossChainBalances[senderChain] < params.value) throw;
                balances[params.to] += params.value;
                crossChainBalances[senderChain] -= params.value;
        }
        ...
}
```

This contract would be initialized on both the main Fedcoin chain and on secondary chains. The logic of the code is simple. The sendCrossChain function first checks if the sender has enough Fedcoin to send; if they do not then it exits with an error. If the sender does have sufficient funds, it creates an event stipulating that the coins should be created on the destination chain, subtracts the sender's balance, and increases the destination chain's balance. The part of the code that manages the destination chain’s balance is a safety feature: even if the consensus of the destination chain is in some way broken, the destination chain can only send back to the main chain as much Fedcoin as the main chain sent it, preventing errors in potentially untrusted sub-chains from allowing attackers to create unbounded quantities of assets on chains that are still intact.

The onReceiveEvent function makes sure that the sender chain can send the given amount of FedCoin, and if so it increments the balance of the recipient and decrements the balance of the sender chain. Note that on each secondary chain, crossChainBalances[mainChain] (where mainChain is an identifier for the main Fedcoin chain) would be initialized to essentially infinity; this signifies that the secondary chains accept the main chain as the authoritative issuer of Fedcoin.

A relay contract has successfully been implemented between Bitcoin and Ethereum in the form of BTCRelay, a smart contract on Ethereum that can read the Bitcoin chain. However, note that the interoperability is one-way: Bitcoin cannot read the Ethereum chain, as its scripting language is not sophisticated enough to do so. BTCRelay is already seeing a small amount of usage; at the time of this writing there is an application called EthereumLottery.io where the lottery smart contract logic itself lives on the Ethereum public chain, but it uses Bitcoin block headers as a source of randomness because each Bitcoin block header has a larger reward and so is more expensive to manipulate.[15] This is a primitive, albeit technologically excellent, example of what can be described as a "cross-chain oracle" application, and the use of a relay plays an important role in increasing the application's security.

Relays for Cross-Chain Atomic Swaps

Going beyond asset portability and cross-chain oracles, the next natural use case for relays is cross-chain atomic swaps, ie. exchanging asset M on chain A for asset N on chain B. In the case of BTCRelay and Ethereum, the MakerDAO team is working on implementing this already; however, any such system unfortunately will necessarily have weaknesses of its own due to fundamental limitations of the Bitcoin protocol itself. One particular concern is race condition attacks. For example, suppose that party A wants to sell 50 ETH for 1 BTC, and deposits the 50 ETH into a contract that essentially says "whoever provides a proof that they sent 1 BTC to address X gets 50 ETH". Party B comes along and sends the 1 BTC. Party A can now themselves send 1 BTC into their own contract, and try to claim the ETH from themselves before party B can; with some probability they will succeed, leaving party A with 2 BTC and 50 ETH and party B with nothing.

If the Bitcoin blockchain had Ethereum-like smart contract capabilities, this would not be a problem: the target address on Bitcoin could simply be a contract that auto-refunds all incoming transfers except for the first one. Without such a capability, the best that we can do is to use semaphores on the Ethereum side: allow the buyer to reserve the exclusive rights to make a trade for some time, and then safely make the claim within that window; this is indeed the solution that the MakerDAO is using. However, the semaphores themselves may become a denial-of-service vector, as someone may try to repeatedly reserve trading rights and never claim them. Making reservations costly resolves this problem, but it does lead to the user-experience issue that one must already have ether in order to purchase more ether through this scheme (there are ways to use untrusted third parties to resolve this, but they are complex and require complex 2-of-2 escrow and locktime techniques on the bitcoin side). Regardless of interpretation details, however, the broader point is that race conditions are a legitimate security concern for applications that are based on relays, and developers should be mindful of them similarly to

[15] The cost of manipulating a lottery that is based on block header randomness is much smaller than the cost of 51% attacking a blockchain (see https://blog.ethereum.org/2016/07/27/inflation-transaction-fees-cryptocurrency-monetary-policy/ for some estimates for 51% attacking bitcoin: $50-150m USD now, $1.2-4m in a transaction fee-only model assuming the Bitcoin ecosystem stays constant). To manipulate a block header-based lottery, all that is required is for the miner who created a block that influences the randomness to refuse to publish the block if they see that the result is unfavorable, sacrificing a single block reward (~$7300 for Bitcoin, ~$60 for Ethereum as of the time of this writing; unfortunately Ethereum's fast block time makes its block headers very insecure as a source of randomness for secondary use cases). Bentov et al provide techniques to increase the expected cost of manipulating block header-based randomness to ~O(sqrt(n) * r) where r is the block reward and the system is allowed to sample n blocks.

how developers should be mindful of re-entrancy when writing single-chain Ethereum applications or SQL injection attacks when writing web apps.

A related weakness of relays is their asynchrony itself; especially if one or both chains are using a consensus algorithm that finalizes slowly, it takes a long time for one chain to verify that the other chain has consensus on some operation, and this limits the speed of cross-chain operations. This is a particularly unfortunate weakness for cross-chain atomic asset exchange, as the market exchange rate may well change over the course of an atomic swap operation. Hence, relays can work very well on chains that have rapid finality, but in those cases where they are too slow the approach may be problematic (although the use of deposits on a third chain with very fast block times may be one way to overcome any issues[16]).

Hash-locking

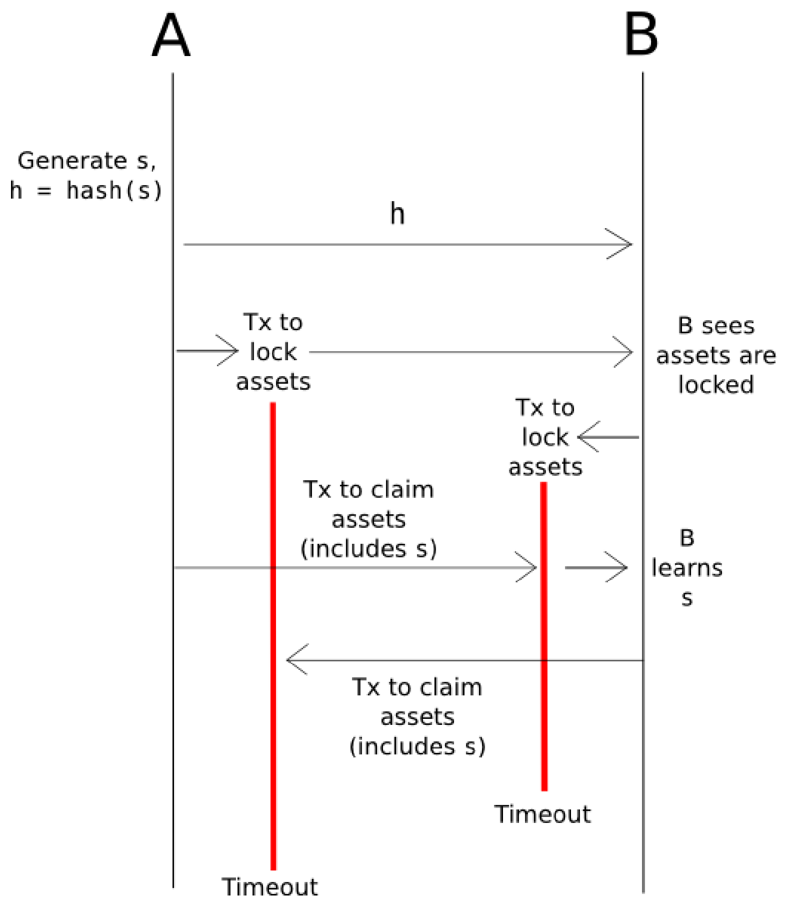
There is another well-known technique for achieving cross-chain atomic operations, one that requires the blockchains to know much less about each other: hash-locking. Hash locking is often associated with the Bitcoin forum user TierNolan[17][18] and is more recently being actively explored by the Interledger protocol as a means of removing the trust requirement of notaries. The simplest description for this mechanism, in the case of cross-chain digital asset exchange, is as follows:

1. A generates a random secret s, and computes the hash of the secret, hash(s) = h. A sends h to B.
2. A and B both lock their asset into a smart contract with the following rules (A locks first, B locks after seeing A's asset successfully locked). On A's side, if the secret is provided within 2X seconds, then the asset is transferred to B, otherwise it is sent back to A. On B's side, if the correct secret (ie. the value whose hash is h) is provided within X seconds, then the asset is transferred to A, otherwise it is sent back to B.
3. A reveals the secret within X seconds in order to claim the asset from B's contract. However, this also ensures that B learns the secret allowing B to claim the asset from A's contract.

[16] “Fast block time” in this context is relative; Ethereum is fast relative to Bitcoin and Dogecoin, but slow relative to well-implemented consortium chains. If sub-250ms finality times are required, the chain can even be a fidelity-bonded centralized server

[17] https://en.bitcoin.it/w/index.php?title=Atomic_cross-chain_trading&oldid=60657

[18] The Lightning Network team is designing their technology in a cross-chain fashion, so cross-chain hash lock trading may well be one of the use cases



Note that this is provably atomic. If A reveals s within X seconds, then this provides at least an X second window within which B can claim their asset. A could make a mistake and reveal s too late, preventing them from recovering their own asset, but this would be their own fault and is easily avoidable. If A reveals s between X seconds and 2X seconds, then A would not get their asset but B does, but once again this would be A's fault. If A reveals s after 2X seconds (or never), then both sides recover their own asset. If A never locks their asset, B doesn't lock their asset either. If B doesn't lock their asset (or locks it incorrectly, eg. with the wrong deadline for s), A can simply never reveal s, and thereby recover their asset.

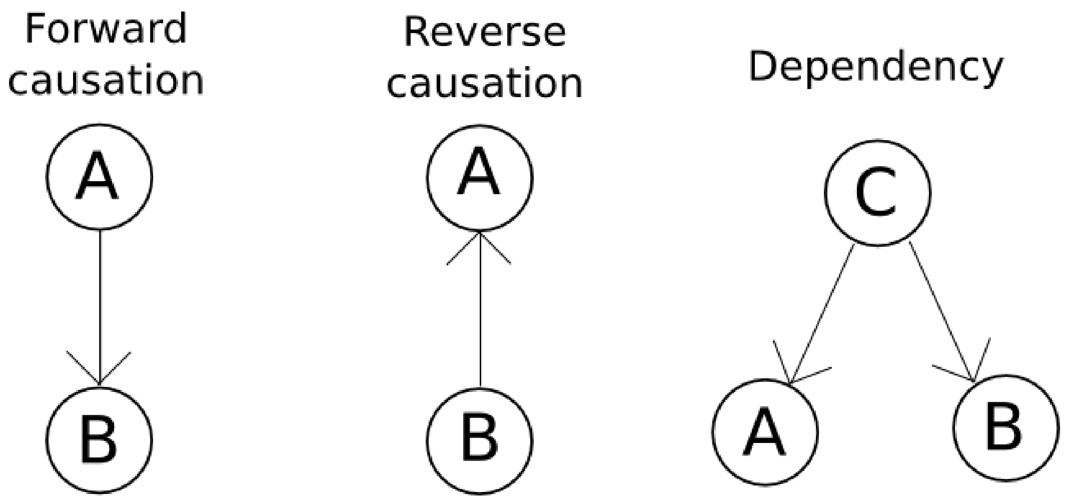
Note that in a world with exchange rate fluctuations, the financial safety guarantee is imperfect, as a malicious A can wait for X/2 seconds and only publish s if the exchange rate after that time moved in a favorable direction; by participating in this hash lock, B is effectively giving A a free option. One could argue that in equilibrium, we may expect B to simply expect A to act in such a way, and expect a fair premium in exchange for this option (a public chain decentralized swap exchange would be a great opportunity to test this hypothesis), though even this scenario is imperfect as it effectively means that much of the time someone wishing to exchange their assets will have to either try several times or pay a sufficiently high premium to reliably entice honest behavior from their partners (or be the offeror without claiming an option premium, once again an economic loss), and this weakens market participants' ability to efficiently hedge their exposure.

Cross-chain hash locking can also be combined with state channel techniques in order to create faster exchanges that mitigate this problem by taking advantage of state channels' greater speed compared to the base blockchain layer; this is still an active area of research and development.
Note that hash locking is only useful for atomic operations; it is *not* useful for asset portability or cross-chain oracle use cases. The reason why it is not useful for cross-chain oracle use cases is simple: accessing a cross-chain oracle is inherently a passive operation with respect to the chain that is being read, and hash-locking is an inherently active operation on both sides. The reason why it is not useful for asset portability is somewhat more subtle: the hash-lock atomic swap protocol preserves the invariant that the total supply of each asset on each chain remains the same, and hence it cannot actually move assets from one chain to another. However, hash-locking can be combined with relays as an efficiency improvement: the possibility of using relays to move assets from one chain to another ensures that exchange rates of one asset for the same asset on a different chain remain very close to 1:1 as any other rate would create an arbitrage opportunity, but ordinary users simply use hash-locking mechanisms to move their coins from one chain to the other by trading with someone who wants to move in the opposite direction.[19]

Theory and implementation

As described earlier, there are many different situations in which interoperability is desirable. From a high-level application standpoint, the use cases range from finance to identity verification to potentially any application that is deemed blockchain-worthy. From a computer-science-theoretic statement, however, we can create a much simpler classification of interoperability types. We can define these as cause-effect graphs, and refer to them as *forward causation*, *backward causation* and *dependency*; we can view them as follows:

[19] This technique was proposed by Blockstream as a way to overcome the inefficiencies of the claim process in their model for Bitcoin sidechains: “Fortunately, they are not necessary for most transfers: holders of coins on each chain may exchange them directly using atomic swap” https://blockstream.com/sidechains.pdf



Those with a basic statistics background may recognize this chart as representing the three possible categories of causes of correlation, not including coincidence, and this is for good reason: interoperability is, to some extent, fundamentally about (reliable) correlation. Forward causation is simple: chain A can cause an event on chain B (this can also be expressed as "chain B can read chain A"). Reverse causation is when chain B can cause an event on chain A - a similar relationship, but one that is worth considering as separate (intuitively, think of this as causation that goes in the opposite direction from the direction you *want* causation to be possible). Dependency is when actions on both chains can arise dependent on some additional outside event. Relays can provide forward and reverse causation, depending on the capabilities of the chains. Notaries can provide any kind of causation. Hash locking, however, can only provide cross-dependency, using the revelation of a hash value as a “common cause”; this is another way of understanding why hash locking is fundamentally weaker than relay techniques.

If we have some of these forms of causality and not others, we could ask the question: is there perhaps some way to convert any one of the three into something equivalent to the others? As it turns out there is one approach, relying on one or more parties that are incentivized by an on-chain security deposit (or possibly, in a consortium chain setting, the strength of a binding legal contract[20]), but its capabilities are limited and it carries important caveats. The argument is that if chain A can read chain B, then one can create a smart contract on chain A containing a deposit that pays that deposit back only if an event is created on chain B every time a corresponding trigger is created on chain A. One possible example use case of this is a kind of “ghetto two-way peg”, where there exists a multisig that stores BTC on the

[20] Legal contracts are not absolute; they depend on both the reliability of the underlying legal system and the solvency of the participants, and so some approach will be required for evaluating their reliability. Arguably, a large part of the value proposition of “collateralized smart contracts” in general is that while purely economic guarantees are in some ways less powerful than legal ones, it is much easier to see and formally evaluate exactly how strong they are, so techniques that rely *purely* on the “wet code” of law should be used carefully. Combining legal agreements with decentralization (requiring multiple parties to act in some way and relying on only a majority of them to be honest) is another possible middle road.

bitcoin blockchain where the participants also hold their own ETH collateral on ethereum, and the participants are required to send transactions from this multisig as needed in order to allow conversion from an “e-BTC” IOU token on ethereum back into BTC on the bitcoin chain (conversion from BTC into e-BTC is easier: just use BTCRelay)[21]. Note that such a scheme does have high ETH collateral requirements to be “cryptoeconomically safe”, as if ETH/BTC rate falls too much there may not be enough collateral to disincentivize the multisig from stealing the BTC[22].

Hash locking can also be to some extent converted into direct causality, although the possibilities are much more limited and difficult to implement. A simple scheme involves a series of hashes being placed both on chain A and chain B, where on chain A the preimages for the hashes are needed to claim a security deposit when a cross-chain message is triggered, and on chain B the preimages trigger the receiving side of the cross-chain message. When a cross-chain message is triggered on chain A, the party with the deposit has no choice but to reveal the preimage, thereby also triggering the event on chain B. However, note that this requires the specific event data to be pre-arranged. An approach that has better properties involves switching from hash-locking to signature-locking: mechanisms on chain B treat messages from some oracle as cross-chain messages, but then these messages can be re-imported by third parties into a mechanism on chain A, which checks that the correct message and only the correct message was signed, destroying the oracle’s security deposit in the event of nonfeasance or malfeasance. But even still, the complexities involved in the above discussion only serve to show that these approaches are inferior substitutes at best, and optimum interoperability can be achieved by having all forms of causality be available between all chains directly.

If we want to satisfy the maximum possible set of use cases with these primitives, we may want to come up with a high-level cross-chain programming language that expresses these concerns. Causation can be expressed through event creation and event listener primitives; cross-dependency can also be expressed through the use of events, though these events would be triggered by a hash commitment being revealed rather than an event on a chain. The ideal would be for a programmer to write an application with components on chain A and chain B, where both contain a line of code that says, for example, onPreimageReveal(0x172db5b4...), and the programmer can then be confident that if the preimage is revealed, then the code triggered by the event will be run; the compiled version of this code would output not just smart contract code but also a daemon (or a plugin for a daemon) that would check both

[21] Much more discussion on the economic safety and limits of such “ghetto two-way peg” schemes in general can be found in the discussions around “stablecoins”. See https://github.com/rmsams/stablecoins, https://makerdao.com/, https://blog.ethereum.org/2014/11/11/search-stable-cryptocurrency/ and https://bitshares.org/technology/price-stable-cryptocurrencies/ for examples.

[22] One way of addressing this is via a “hybrid trust model”, where the parties involved are selected to be maximally trustworthy as well as economically incentivized. This is similar to the approach taken by Rootstock for its blockchain security, where the chain is secured by a trusted federation if less than 50% of Bitcoin miners are participating, and control slowly turns over to Bitcoin miners as enough of them become involved for a merge-mining model to be secure.

blockchains and cross-post the event if it is published to one chain. The daemon can be run by anyone and does not need to be trusted; a copy could be run by the parties that are beneficiaries to the contract, or may also be organized by the users or developers or a given chain as a public service.

Formal modeling and Security

Formal verification of these cross-chain scripts could be done using a model that assumes that some notion of timestamp on chain A is correct to within some error margin δ_1, a timestamp on chain B is correct to within δ_2, and that message-passing latency between the chains is less than δ_3 (these deltas would incorporate block times, risk of short-range 51% attacks, short-range censorship, etc), and thereby reason that, for example, an onPreimageReveal event will happen on chain A and chain B with maximum time discrepancy $\delta_1 + \delta_2 + \delta_3$, or that an event created by a contract on chain A will be read by chain B with some other maximum time discrepancy. This would then allow for, say, a cross-chain payment-versus-delivery contract to have a formal proof of atomicity given those assumptions[23].

Note that in a cross-chain context, incorporation of such assumptions into a formal model is required. In a single-chain context, modelling the protocol that the application is running on is much less necessary, with the possible exception of a censorship-resistance assumption ("if A really wants to get transaction X into the chain, they will be able to do so within 10 minutes"). This is because, from the point of view *inside* the decentralized computer, the decentralized computer is perfectly secure: every single operation is 100% guaranteed to work as planned. In a multi-chain context, however, chain A and chain B's security models are separate, cross-chain information transmission needs to be modelled, verification of chain A inside of chain B needs to be modelled, etc. Hence, complexities arise, and these complexities become more interesting once we get into the discussion on failure modes in the next section.

Additionally, in the case of public blockchains particularly, it is impossible to avoid discussing economics. For example, 51% attacks, transaction spam attacks, and other similar measures. Such attacks are always a possibility, but they also have a cost, and so a richer formal model may even produce claims of the form "this algorithm with these parameters is safe for up to $200,000, but not more, because at those scales attacks become economically feasible and you need to increase the time windows"[24].

---

[23] To see why these timing assumptions are necessary, consider the case of hash-locked atomic swaps described in a previous section. In that protocol, A can recover their asset if the secret is revealed and pushed into A's chain within X seconds, and B can recover theirs if the secret can be pushed into B's chain within 2X seconds. However, if there are ways to attack the chains to create a time lag greater than X, then A may be able to claim their asset, and prevent B from claiming theirs long enough for A to be able to withdraw it.

[24] See http://www.ofnumbers.com/2015/11/02/integrating-mining-and-attacking-analyzing-the-colored-coin-game/ for an example of an attempt at modeling security concerns of applications on top of a public blockchains in the asset issuance ("colored coin") use case.

However, this is not always that easy to model, one particular reason being the possibility of attacks that affect multiple applications at the same time. For example, censorship through transaction spam is generally ineffective for stealing funds from high-value applications, as the legitimate sender only needs to get one transaction in in order to be safe whereas the attacker needs to fill every block, and in a public chain context the sender can just add to their transaction a very high fee that would only set them back slightly but would require an attacker to potentially burn through hundreds of thousands of dollars per hour in order to block them. However, such an attack harms all applications on the chain simultaneously, and so if a chain has near-full usage under normal circumstances, and if these operations are time-sensitive (eg. state channel settlements), then a transaction spam attack may well cause enough of those operations to fail simultaneously to be worth the cost[25]. 51% attacks have a similar property; even if every single application is designed in such a way that losses from a 51% attack never exceed $50,000, if there are a thousand such applications at the same time then a 51% attack at a cost of $5 million that attacks all of these applications in parallel may well be profitable. Hence, a form of risk modeling similar to that used for financial collateral and risk exposure management may be appropriate in public chain settings.

Governance and Failure Modes

The above discussions make sense in a context when both chains involved in an interoperating application are working normally. However, what happens when one or both of those chains either fails or experiences an irregular governance event?

A quick listing of possible irregularities, both positive and negative, that may happen to a chain is as follows:

1. 51% attacks leading to reversion of transactions
2. 51% attacks leading to successful (total or partial) censorship
3. 51% attacks leading to the creation of an invalid chain that is not accepted by “full nodes” on the chain, but that is accepted by “light nodes” and relays. This invalid chain may contain invalid state transitions (eg. you do not need to actually have 10 trillion Fedcoins in order to trick a relay into thinking you do you have the ability to conduct 51% attacks).
4. "Soft forks" changing functionality (if undesirable, this is arguably equivalent to (2))
5. "Hard forks" where all or nearly all of the user base of a chain effectively agrees to switch to using a new chain with either a rule change or an irregular state transition, and the

[25] This is one argument against the combined philosophy of “full blocks” being a default state, a fixed vertical supply curve for transaction space in blocks and the use of the lightning network as an exclusive scaling paradigm in the context of public blockchains: the three factors together may create a “perfect storm” of incentives for attackers to steal funds by manipulating channel settlement.

level of coordination is sufficient that the name, brand and community of the old chain is essentially entirely moved over to the new chain.

6. Network splits leading to a chain "splitting" into two and the split persisting for some time.[26]
7. A chain using a consistency-favoring consensus algorithm losing so many nodes that no new blocks can be produced and the chain "fail-stops" (this can be viewed as equivalent to (2), but also would in almost all circumstances lead to (4)).

How would a cross-chain application deal with one of these kinds of events happening on one or both of the chains that it "lives" on? If no failure mode handling exists, then the effects differ by situation. For example:

- A transaction reversion event can lead to loss of atomicity for otherwise atomic cross-chain operations, as one part of a trade may be reversed but not the other.
- A replay spoofing event can lead to loss of atomicity for otherwise atomic cross-chain operations, as one part of a trade may not actually happen.
- A censorship event can sometimes lead to loss of atomicity for atomic cross-chain operations, as it could prevent withdrawal transactions long enough for a counterparty to withdraw from an agreement. However, the ability to do this depends on each application, and it is not always the case that such problems exist.
- A hard fork of one chain can potentially lead to "relays" tracking that chain breaking, although this is not nearly always the case, and in fact depends heavily on the specifics of both the fork and the implementation of the relay mechanism.
- A network split is a complicated case, because depending on the details of the consensus algorithm and the relay it may resemble either a reversion event or a fail-stop event. Proof of work network splits lead to reversions; network splits on traditional byzantine fault tolerant consensus algorithms lead to fail-stops; Ethereum’s Casper proof of stake model is a hybrid, allowing the relay design to choose which "degree of confirmation" it waits for and thereby choose between the two fault modes on a per-application basis.

In general, one can make the observation that consistency-favoring consensus algorithms[27] lead to "safer" consequences for cross-chain operations. To see why, consider the example of "Fedcoin" from

[26] Aside from accidental splits caused by network failures, in a consortium-chain context splits may also happen for geopolitical reasons (eg. sanctions against Iran, Brexit); see below for more details.

[27] “Consistency-favoring consensus algorithms” is a term used by Ethereum researcher Vlad Zamfir to differentiate between some properties of traditional Byzantine-fault-tolerant consensus and Bitcoin-style consensus (which he calls “availability-favoring”). Consistency-favoring algorithms tend to require a supermajority of validators (eg. ⅔ + 1) to finalize a transaction, and so require ⅓ Byzantine failures to experience a fail-stop event or ⅔ Byzantine failures to experience a reversion event or a light-client/relay-spoofing event. Note that these probabilities can be adjusted; for example, we can achieve ¾ robustness against reversion or spoofing failures at the cost of reducing robustness against fail-stops events to ¼. Availability-favoring algorithms can tolerate any number of crash faults

above, which is based on some home ledger and which has M extant coins on some external chain A. Suppose that chain A suffers a repeated reversion or spoofing attack. Then, chain A can send an arbitrary number of SEND messages to the home chain by sending a SEND message, waiting for the message to be confirmed in the relay on the home ledger, reverting to a block before the message was sent, and repeating. The home chain would accept M coins' worth of such messages from chain A, and credit M coins to the attacker, but then it would stop; the home chain's accounting of child chain balances protects it from suffering undue losses, though the loss of M coins is unavoidable. Note that a converse attack also applies: if the home chain can suffer reversion attacks, then an attacker can fill all child chains with an arbitrary number of coins, effectively making them worthless.

Now, suppose that chain A suffers either a censorship attack or a fail-stop - the failure mode that would be more likely if chain A was consistency-favoring rather than availability-favoring. Actors on chain A may now be *prevented* from moving their Fedcoins to the home chain, but an attacker cannot steal anything - hence, the failure mode is less harmful and there is no gain to the attacker in making the attack[28].

Another important consideration is the type of ledger environment in question. If the home ledger is a consortium chain, then taking emergency action (eg. a remedial hard fork, freezing the attacker's assets) to mitigate the damage would be easier, and so the total damage may be much less. If a public chain is used, then the damages may depend on other considerations, including the degree of "permissionlessness" of the asset itself. However, no technology can prevent attackers from profiting entirely, as they may be able to exchange the funds into assets based in other jurisdictions or even permissionless cryptocurrencies more quickly than any administrators can detect and stop a theft.

Note that going from a single chain to a model with cross-chain portable assets, under normal assumptions, changes little about a process except for adding asynchrony, but under failure modes it weakens the security model in a subtle way: reversion attacks become theft vulnerabilities. This is similar to the way that the security model is changed by state channel technologies, although the claim in that case is subtly different: *censorship* attacks become theft vulnerabilities. This has particular relevance in a public chain context: a common argument for why miners do not collude to 51% attack their own chain is that they do not benefit from destroying the ecosystem that supports them; however, in the case of a cross-chain movable asset, it is entirely possible to come up with excuses for why

among validators, but at the cost of not having such a strong notion of finality; they are vulnerable to reversion or spoofing failures at the ½ Byzantine failure level.

[28] This is true in the case of movable assets, but in other cases such as financial derivatives there are ways for attackers to gain by triggering consistency failures; however, (i) those gains are much smaller than the gains from literally being able to steal all assets from chain A, and (ii) this is why the bias toward favoring consistency is a general observation and not an absolute rule.

attacking one particular “sidechain” actually helps the main chain[29], and so sidechains find themselves in an uneasy political relationship with the miners of the home chain that their asset lives on.

Interoperability and Lifecycle Events

One class of event that is particularly interesting to analyze particularly in a consortium chain setting is lifecycle events. This includes:

- Change of economic boundaries (eg. Brexit)
- Implementation and removal of sanctions (eg. as against Russia and Iran)
- Currencies ceasing to exist (eg. introduction of the Euro in 1999)
- Currencies “forking” (eg. the Iraqi Swiss dinar)

This may lead to several kinds of events:

- A chain winding down because it’s raison d’etre (one particular currency) ceases to exist.
- A chain starting up ex nihilo to accommodate a new asset
- A chain literally forking, in a similar style to the ETH / ETC fork (literally splitting an asset into two in this way has precedent mainstream finance only in the fork of spinoffs, as in the case of Ebay and Paypal, but it may well end up being used for currencies in secession scenarios)
- Certain kinds of interoperability between two chains being blocked because of a sanction. There is also the possibility of “de-facto soft sanctions” where a particular kind of interoperation event suddenly carries a much higher compliance burden due to legal changes[30].

Special care should be taken in scenarios similar to the 1999 Euro introduction: a scenario that is best avoided is one where a new asset is created that has no single home ledger but instead has multiple chains that are all authoritative over a subset of balances. It is arguably a large part of the whole advantage of blockchain technology that the complexities associated with such scenarios can be prevented. One possible introduction path is for the home ledgers of the old currencies to remain, but become subordinate to a new home ledger (ie. equal status to “chain A” in our previous Fedcoin examples), and for balances to be simply multiplied by the exchange rate in-place[31].

[29] See Paul Sztorc for an example argument: “Peter Todd emphasizes a key sidechain characteristic: miners can destroy/steal-from any sidechain that they merged-mine …at no direct cost. But with the bad comes the good: miners (as a group) can and should censor any sidechain which threatens the value of the “sidechain portfolio” (or affects it superfluously)” http://www.truthcoin.info/blog/contracts-oracles-sidechains/

[30] See FATCA and Swiss banks’ response to the law for an instructive example

[31] Technically, this multiplication-in-place does not even require a hard fork: the balances could remain exactly the same, but different fixed exchange rates with the home ledger would be introduced for each chain; however, in

Preventing interoperation for regulatory reasons is in some ways more difficult; the reason is that anyone can create a verifier for one chain inside another chain, and hash locking is even easier; it may well even be possible to hide a hash lock inside of something that looks like a mathematically complex financial derivative. However, there are still obvious actions that can be taken: for example, if the central bank of country A wishes to freeze access to its currency for country B, it can simply freeze the balances associated with any chains in that country. Softer measures like restricting cross-chain claims to $X per person per year are also possible. In general, nearly arbitrary controls on movement of assets from one chain to another may be implemented; however, preventing *exchange* between assets on the same chain is much more challenging and would likely be dealt with only through heuristic pattern-detection techniques similar to those employed today to deal with fraud[32] and structuring[33].

One highly desirable goal in all of these scenarios is that regardless of what unexpected events occur, the transition for applications on the chain, including contracts on the chain, should be maximally smooth. This can happen through a combination of advance notice, use of maximally "clean" techniques for implementing the change, and the use of appropriate mitigation and recovery techniques within any financial or other contracts on the chain. Technically chaotic splits, freezes and other black swans should only happen during rare exigencies such as wars (or possibly lower-grade political retaliations, one possible example being a government disrupting financial access to a region during a rapidly unfolding secession scenario), in which case damage prevention and recovery techniques at contract level are likely the only form of mitigation.

Mitigation and Recovery from Failure Modes

From experience we know that the concept of everlasting, unchanging and immutable protocols that never experience either attacks or governance events is an unrealistic fantasy. Needs change, priorities change, the unexpected happens, and implementations themselves often end up containing bugs. If the notion, held by many in the Bitcoin community[34], that the Bitcoin Core C++ code *literally is* the specification of the Bitcoin protocol was true, then the attacker who created 93 billion BTC out of thin

the user interface this could still be presented as a single currency. This is in fact a cleaner and better way to do it, as it ensures that balances stored inside applications on the chain are also converted correctly (eg. if a conversion is done through protocol-level multiplication-in-place, then one possible risk is that a financial contract that contains some formula dealing with asset quantities or prices would see the formula not get converted along with the rest of the currency, leading to unexpected results).

[32] http://www.ibmbigdatahub.com/blog/how-improve-bank-fraud-detection-data-analytics

[33] http://www.prnewswire.com/news-releases/what-is-structuring-and-how-do-banks-detect-this-form-of-money-laundering-56903762.html

[34] See Bitcoin's own wiki page for an example: "The Bitcoin protocol is specified by the behavior of the reference client, not by this page" https://en.bitcoin.it/wiki/Protocol_documentation

air through an integer overflow attack should never have been denied his rightful gains through the coordinated soft fork that ended up reverting 12 hours of history and removing the vulnerability. Given these inevitable realities, how do we manage the risks?

One possible response is to limit interaction to being between chains that have similar governance policies and consensus mechanisms, or at least governance policies and consensus mechanisms that trust each other. Some prefer to stay within the realm of private and consortium chains for this reason, seeing "anarchic" public chains as being the oil to traditional finance's water. But then, different consortium chain governance mechanisms may also end up disagreeing with each other, or even trying to actively attack each other; deliberate attempts to prevent interoperability have precedent in the form of financial sanctions, eg. those against Russia and Iran. A government of one country may try to commandeer a chain with the majority of its nodes based in that country, and conduct attacks against services of hostile countries that try to interface with that chain as a form of cyber-warfare. Hence, if we want to have a maximally globally applicable model, a model where each chain only trusts itself and assumes the possibility of arbitrary behavior from other chains, ie. where *every* chain is "anarchic" seems like a reasonable place to start; after all, describing geopolitics itself as literally anarchic is not at all inaccurate. Hence, we want to deal with the possibility that chains will fail.

At this point, you may ask, if chains can fail then what is the difference between analyzing chain interoperability and analyzing interoperability between centralized systems? What is it that blockchain technology brings into the discussion that creates fundamentally new considerations that do not exist between, say, SWIFT and national banking systems? One argument is that what we call "blockchain technology", or "crypto 2.0", is in fact *a package of related technologies and philosophies*, including digital signatures, hashes, Merkle trees, peer-to-peer networking, and a general notion of "mistrust first" and "push, not pull", and a consequence of these philosophical choices taken together is that, even in the context of various forms of 51% attacks, the blockchain environment presents unique properties that we can take advantage of in order to design safer systems.

Technologically speaking, we can start the discussion with one particular property: *a blockchain leaves a hash and signature-based cryptographically verifiable trail of every transaction that it processes and every state change that it makes*. Even if a 51% attack takes place, the old chain still exists, and is cryptographically - not just cryptoeconomically or socially, but also cryptographically in the sense of being secured by hard math - immutable. The fact that the attack did take place can be seen by all, and if a given actor (or relay) viewing a blockchain has its own memory and its own block it can even see which branch came first. If no new blocks get created for some time, this can be seen too.

Some of the consequences of this are fairly mundane and include what we have already discussed: it is only because of the heavy use of hashes and signatures in blockchains for cryptographic authentication that we can construct relay-style interoperability mechanisms as well as constructs like hash-locking. Neither of these techniques is remotely a possibility in the world of traditional centralized APIs. The ability to detect failure modes, however, is even more interesting. For example, suppose that we wanted to have an event, onChainReverted(chainID, k). The event could automatically be triggered in a proof of work context if a relay sees that at some point, block M was the head of the chain, and then the head switches to block N, where N and M only share a common ancestor more than k generations behind each other - a k-block revert.

Another example would be an onFailStop(chainID, s) event that triggers when a blockchain fails to create a new block in s seconds. A more complex one to implement would be a censorship detection oracle where anyone could submit a transaction and the oracle would reply yes if the transaction *appears* worthy of inclusion in a block but does not get included within, say, 30 blocks. A protocol hard fork event could be triggered in a way that voluntarily includes a state change to a “flag contract” signifying a fork, allowing an onFork event to be automatically triggered. A more centralized solution to all of the above is, of course, a multisignature notary oracle that signs a message when it detects a particular chain undergoing an irregularity.

When an application receives one of these messages, what should it do? There is naturally a limit to how far you can go; it is fundamentally impossible to stop attackers from draining a sidechain by repeatedly reverting it, so attackers breaking our hypothetical "chain A" in the previous section would be able to steal M Fedcoins no matter what kind of programming-language computer science magic is done at the high level, although it is possible to use smart contracts together with reversion oracles to create a kind of low-trust “blockchain failure insurance”[35]. However, in other cases (eg. fail-stops), it *is* possible to achieve better properties where no one need bear any risk.

One strategy is simple: have each contract appoint an "emergency curator", where the curator only gains powers over the contract in the event that a fork oracle claims that one of the chains that the contract in question is interacting with experienced an irregularity. This curator could then individually attempt to execute contracts to the best of its ability given their original intent. The curator could be a bank, a regulatory body, an NGO or even itself a consortium consisting of any combination of the three; even though the curator will not be leaned on often, it is a role requiring trust and so a trustworthy body is strongly preferred to fill the role. This strategy is easiest to code, simple for human beings to understand (including understanding why it introduces no additional centralization in the normal case), and so is most likely to be implemented in the short term.

Another approach is to try to handle certain cases automatically; for example, if a given chain experiences a fail-stop event, timeouts can be extended until it recovers. Orderly hard forks can carry flags in the onFork event arguments that specify the level and type of intrusiveness, and actions can be

[35] One side benefit of such an insurance market is that it may also serve as a decentralized way of preventing or mitigating combination attacks, as the existence of such a market encourages participants to effectively report their exposure to a fork at any particular time, allowing the total exposure to be estimated.

taken based on the type in question. In some cases, fallbacks may work: if identity information from chain A is sought by a contract on chain B, and chain A is unavailable, then it could try to get the same information from other sources, or appoint a “fallback KYC curator”. In some cases (eg. cross-chain interest coupon payments), one can simply register a payment as failed and try again some time later.

Altogether, we can see that interoperability adds a whole new layer of complexity to applications. This complexity brings great promise, and in many cases even with this added complexity, cross-chain smart contracts may well still be far less complex, and far less vulnerable, than the processes that govern inter-system transfers, contracts and exchanges today. However, it also brings its own unique challenges, its own unique risks, and a need to understand what these risks, and the mitigation strategies to resolve them, are. In the long term, we hope to see research in interoperability be designed around this notion of cross-chain programmable languages so that the challenges of implementing the basic building blocks of interoperability securely, and the challenges around taking advantage of such features and implementing them into real-world applications, can be handled separately; however, we are still in the early stages and there remains a long way to go.

The Road to Interoperability in Practice

	Notaries	Relays	Hash-locking
Interoperability types	All	All (if relays exist on both chains; otherwise one-way causality only)	Cross-dependency only
Trust model	Majority of notaries honest	Chains do not fail or get “51% attacked”	Chains do not fail or get “51% attacked”
Usable for cross-chain exchange?	Yes	Yes	Yes
Usable for cross-chain asset portability?	Yes (but requires universal long-term notary trust)[36]	Yes	No
Usable for cross-chain oracles?	Yes	Yes	Not directly
Usable for cross-chain asset encumbrance?	Yes (but requires long-term notary trust)	Yes	In many cases, but with difficulty

[36] “Universal” here means that every user of an asset must trust the same notaries; this is distinct from the asset encumbrance scenario where different users can still trust different notaries

Ease of implementation	Medium	Hard	Easy

Use cases involving chain interoperability are among the few that may take the longest to come to fruition, simply because the set of dependencies is so large. Such a use case requires (i) some chain A that is mature enough to build off of, (ii) some chain B that is mature enough to build off of, and (iii) some application or need that cannot be served by implementing it on a single blockchain. (iii) itself arguably can only be met if (i) and (ii) themselves both have reasonably mature applications on top of them that have a need to interoperate.

Hence, short term interoperability use cases will likely have to do with public blockchain cryptocurrency exchange and possibly public blockchain cryptocurrency derivatives. Projects such as MakerDAO may be most interested in the cryptocurrency derivatives use case, as they deliberately intend to diversify their holdings across many assets based on many blockchains. Consortium chain use cases will take much longer, because Liquid is perhaps the only consortium chain that is seeing any significant usage, and even that has a primary use case that is entirely dependent on public blockchain cryptocurrencies; the use of consortium chains in mainstream finance is still very far away. Hence, public chain cryptocurrency financial systems will likely be the primary testbed for cross-chain interoperability mechanisms in the short term, giving ample time for the technology to become mature once it can be applied for other use cases.

In the longer term, building interoperable applications is something that should be done on a basis of responding to actual need; while the underlying technology can certainly be developed in the abstract, it would likely be a waste of resources to try to build applications that “connect the chains together” without a clear underlying use-case-driven reason for why such a thing would be useful. In the meantime, it seems clear that notaries, relays and hash locking schemes are excellent building blocks to start building on top of, and a possible next step is to continue working on developing these tools while proving their applicability in some simple example test-cases, including payment-versus-payment, delivery-versus-payment, asset portability and cross-chain oracles. If the technology is built to be sufficiently generic, then its applicability in whatever applications end up being optimal will emerge naturally over time.