Atomically Trading with Roger: Gambling on the Success of a Hardfork

Patrick McCorry[1(✉)], Ethan Heilman[2(✉)], and Andrew Miller[3,4(✉)]

[1] University College London, London, UK
p.mccorry@ucl.ac.uk
[2] Boston University, Boston, USA
heilman@bu.edu
[3] University of Illinois at Urbana-Champaign, Champaign, USA
soc1024@illinois.edu
[4] Initiative for Cryptocurrencies and Contracts, Berkeley, USA
http://www.initc3.org/

Abstract. We present atomic trade protocols for Bitcoin and Ethereum that can bind two parties to swap coins in the event that two blockchains emerge from a single "pre-fork" blockchain. This work is motivated by a bet between two members of the Bitcoin community, Loaded and Roger Ver, to trade 60,000 bitcoins in the event that Bitcoin Unlimited's planned hardfork occurs and the blockchain splits into two distinct forks. Additionally we study several ways to provide replay protection in the event of hardfork alongside a novel mechanism called migration inputs. We provide a detailed survey and history of previous softforks and hardforks in Ethereum and Bitcoin.

1 Introduction

Bitcoin [29] is the world's first successful and most valuable cryptocurrency. In June 2017, it reached a market cap of $ 43 bn USD [10] and processed ≈250,000 transactions per day [4]. However, Bitcoin's future is uncertain; it is reaching its capacity limits, and so far the community has failed to reach consensus on how best to increase its capacity.

One proposed approach for increasing capacity, called Bitcoin Unlimited (BU), involves removing the 1-megabyte-per-block parameter that most directly effects the capacity limit [35]. A competing approach, the Core Roadmap [26], calls for a technical upgrade called SegWit [24], followed by deployment of the overlay payment network, Lightning [30]. Both approaches require changing the network's consensus rules; however there is a critical difference between them, BU is implemented as a hardfork upgrade, whereas Core relies on softforks. These two approaches are mutually incompatible: unlike a hardfork, a softfork is "forward-compatible" in the sense that blocks mined using the new rules can still be processed by non-upgraded clients (for additional details see Sect. 2.3).

If the community remains divided on which approach to support, then the result may be a schism, where each faction maintains a distinct fork of

J. Garcia-Alfaro et al. (Eds.): DPM/CBT 2017, LNCS 10436, pp. 334–353, 2017.
DOI: 10.1007/978-3-319-67816-0_19

Bitcoin with mutually incompatible consensus rules.[1] Both blockchains will diverge post-fork, but share the same pre-fork transaction history. We denote the non-upgraded fork as Fork-1 and the fork with new consensus rules as Fork-2. As both forks share a common history, a party holding X coins in the pre-fork blockchain will, after the hardfork, hold X coins in Fork-1 and hold X coins in Fork-2.

In this paper we consider the scenario where, prior to a hardfork, Alice and Bob decide to bet on which of the two forks will be most valuable. After the hardfork, Alice's coins in Fork-1 are sent to Bob, and Bob's coins in Fork-2 are sent to Alice. Remarkably, this gambling scenario is inspired by real-world events: two wealthy members of the Bitcoin community, Loaded[2] and Roger Ver, have expressed the desire to arrange a 1:1 trade of coins in the event that Bitcoin Unlimited performs a hardfork from Bitcoin [23]. Roger wants to exchange 60,000 of his coins on Fork-2 for 60,000 of Loaded's coins on Fork-1. After the trade Loaded would have 120,000 coins on Fork-1 and Roger would have 120,000 coins on Fork-2 (this trade was roughly $120 million USD when proposed).

There are two previously known approaches we could employ for cross-chain trades, though both have drawbacks in this scenario. In the first approach, both parties escrow funds with a third party who facilitates the trade; several protocols have been outlined by Goldfeder et al. [14] that could mediate such a trade. The second approach, an atomic cross-chain swap smart contract, was proposed by TierNolan [33] (see Appendix A for details). Unfortunately the first approach requires a trusted third party and the second does not allow users to commit to the bet prior to the hardfork.

In this paper we introduce a novel atomic cross-chain trade where the trade can be committed prior to the activation of a hardfork, but executed after the hardfork. We construct protocols for both Bitcoin and Ethereum (the second most popular cryptocurrency with a market cap of $29 bn USD as of June 2017 and which had four hardforks in 2016). It is worth mentioning that our protocol for Bitcoin *does not require a fix for transaction malleability*, but relies on the hardforked blockchain Fork-2 implementing replay protection[3]. On the other hand, our protocol for Ethereum leverages a Hardfork Oracle contract that can detect if it is on Fork-1 or Fork-2. Our contributions are:

- The first atomic cross-chain trade protocols for Bitcoin and Ethereum that can transfer coins across both sides of a hardfork
- A novel mechanism which we call migration inputs that provides replay protection in the event of a Bitcoin hardfork.
- A detailed history of hardforks and softforks in Bitcoin and Ethereum.

[1] A schism has previously occurred in the case of Ethereum, whose *TheDAO* hardfork precipitated a split into Ethereum and Ethereum Classic.

[2] Loaded is a pseudonym used by a person on the bitcointalk forums.

[3] The user can choose which blockchain can accept their newly signed transaction.

2 Background

In this section we cover technical background for our protocols, a history of soft/hardforks in Bitcoin and Ethereum and a survey of replay protection proposals for Bitcoin including *migration inputs* a novel replay protection mechanism.

2.1 Bitcoin

Bitcoin is a digital currency that facilitates trading the ownership of a single asset (i.e. bitcoins). Users send bitcoins to other users by publishing transactions. All transactions are stored in a globally replicated data structure called the blockchain. A computationally expensive process called mining (i.e. Proof-of-Work) is responsible for periodically electing a leader to create and append a new block of recently authorised transactions to the blockchain. To understand our protocol we focus on a Bitcoin transaction's scripting and lock time capability.

A Bitcoin transaction contains a list of inputs and outputs. Inputs specify the source of bitcoins along with evidence that the spender is authorized to spend these bitcoins. Outputs specify the conditions that must be satisfied before its associated bitcoins can be spent. Inputs and outputs are controlled using a limited forth-like language called *script*. The most popular script is the *pay-to-pubkey-hash* script which requires a digital signature σ_A from the corresponding secret key of a specified Bitcoin address (i.e. hash of the public key $H(PK_A)$).

Scripts can include a function `CHECKLOCKTIMEVERIFY` [34] to prevent spending an output until time t. This lock time t is compared against the median time of the previous 11 block's timestamps [21]. It is worth mentioning that a block's timestamp must be greater than the median timestamp computed over the 11 previous blocks and it must not be greater than 2 hours from a node's network time. As a result, the median time is loosely-bound with current time.

2.2 Ethereum

The motivation for Ethereum (and Ethereum Classic[4]) is to store and execute expressive *smart contracts* on a peer-to-peer network as opposed to simply trading a single asset. Similar to Bitcoin, users must authorize transactions using an Ethereum account (i.e. public-secret key pairs) and miners are responsible for appending new blocks to the blockchain. Unlike Bitcoin, the transaction payload contains the code/execution instructions for the contract and the transaction's destination is the contract address[5]. Here we focus on the capability of smart contracts and how coins can be locked for a pre-determined period of time.

Ethereum smart contracts are written in Solidity which is a Javascript-like language. Prior to being stored in the blockchain this code is compiled from Solidity to EVM (Ethereum Virtual Machine) code. Transactions that contain

[4] Emerged in July 2016 after Ethereums's TheDAO hardfork.

[5] The hash of the transaction's nonce and the creator's Ethereum accounts address.

Table 1. Previous forks. A list of significant softforks, hardforks and blockchain splits in Bitcoin, Ethereum and Ethereum Classic.

Name	Date	Softfork	Hardfork	Split
	Bitcoin			
1MB Block Size	12th Sep 2010	✓	✗	✗
182 Billion Coins	15th Aug 2010	✓	✗	✓
BIP30	15th Mar 2012	✓	✗	✗
BIP16	15th Apr 2012	✓	✗	✗
Bitcoin Core 0.8	11th Mar 2013	✗	✗	✓
BIP34	24th Mar 2013	✓	✗	✗
BIP50	16th Aug 2013	✗	✓	✗
BIP66	4th Jul 2015	✓	✗	✓[a]
BIP65	8th Dec 2015	✓	✗	✗
BIP68/112/113	4th Jul 2016	✓	✗	✗
	Ethereum			
Homestead	14th Mar 2016	✗	✓	✗
TheDAO	20th Jul 2016	✗	✓	✓
Tangerine Whistle	18th Oct 2016	✗	✓	✗
Spurious Dragon	22nd Nov 2016	✗	✓	✓[b]
	Ethereum Classic			
Gas Reprice	25th Oct 2016	✗	✓	✗
Die Hard	13th Jan 2017	✗	✓	✗

[a] It was discovered that a significant portion of miners who signaled for the activation of BIP66 were not fully validating blocks (i.e. spv mining). This led to a temporary blockchain split and the invalid fork was eventually discarded.
[b] The blockchain split occurred on the 24th November 2016.

EVM code are propagated throughout the network and deterministically executed by all peers using their copy of the EVM. The transaction is stored in the blockchain to ensure the contract's state is no longer reversible.

It is worth mentioning that locking coins until time t can be expressed in a straight-forward manner. Solidity supports accessing a block's timestamp using `block.timestamp` or a block's height using `block.number`. Furthermore, there is a tighter-bound on a block's timestamp as it must be greater than the previous block and strictly less than the user's local clock [28]. Next, we discuss soft and hardforks that have occurred in Bitcoin and Ethereum.

2.3 History of Forks

Cryptocurrencies have clearly defined consensus rules on which all network peers (including both miners and relay nodes) must agree in order to deterministically

validate scripts, transactions and blocks. These rules define a transaction's format, the semantics of its scripting language, the rate at which new coins are minted, parameters such as the maximum block size, and many more constraints. Changing these consensus rules to upgrade a cryptocurrency requires community-wide co-ordination and approaches generally fall into two categories:

- A **softfork** introduces new rules such that a new block conforming to the changed consensus rules is considered valid by non-upgraded nodes, i.e. the proposed change is "forward compatible."
- A **hardfork** introduces new rules such that a new block conforming to the changed consensus rules is not considered valid by non-upgraded nodes.

In both cases, a fork proposal typically has a "flag day" activation time and built-in activation conditions, such as requiring a threshold limit of miners and/or validators to indicate support before the change is activated. This provides ample time for the entire community to upgrade their nodes to support the new consensus rules. However, the difference between a softfork and hardfork is how non-upgraded nodes are impacted. In the former, non-upgraded nodes will follow the majority of miners, whereas in the latter non-upgraded nodes will find themselves in a partitioned network. In practice, Table 1 highlights that Bitcoin has performed softforks (with the exception of one hardfork due to BerkeleyDB's misconfiguration), whereas Ethereum (and Ethereum Classic) have used hardforks. Next, we explore the new consensus rules introduced in Bitcoin, Ethereum and Ethereum classic.

Bitcoin. So far, Bitcoin has implemented over six softforks. These softforks range from introducing rules to prevent miners creating coinbase transactions with duplicated identification hashes [1,38], requiring all ECDSA signatures to strictly enforce DER coding [39], and introducing both absolute [34] and relative lock times [13] for individual transaction outputs. In terms of implementation, this involves storing new information in the `scriptsig` of the coinbase transaction, constraining transaction validation rules or re-defining the use of special `OP_NOP` function.

On the other hand, Bitcoin has experienced two accidental (and temporary) splits (i.e. Fork-1, Fork-2 emerged) that required miner intervention to remedy. The first split permitted a user to exploit an integer overflow bug and create 184 billion coins. This required miners to co-operatively extend a new blockchain without the coin creation transaction [5] and to enforce a soft-fork to prevent this exploit. The second split involved miners who upgraded to Bitcoin Core 0.8 accidentally creating blocks that were invalid for Bitcoin Core 0.7. Unfortunately, BerkeleyDB's configuration in Bitcoin Core 0.7 was non-deterministic and as a result was not compatible with LevelDB's configuration in Bitcoin Core 0.8. Resolving this fork required miners to immediately downgrade to Bitcoin Core 0.7 and abandon the forked blockchain. Next, the developers released Bitcoin

Core 0.8.1 that enforced the activation of a hardfork[6] after a two-month grace period for miners and users to upgrade [2,27].

Ethereum. Ethereum has executed four hardforks in response to community demand and to reduce the impact of network spam attacks. Homestead modified the gas cost for creating transactions and EVM operation codes [36], TheDAO fork reversed a theft of approximately $40 m worth of ether [16], Tangerine Whistle reduced long-term gas changes for IO-heavy operations [7] in response to a spam atack and Spurious Dragon enabled transactions to delete empty accounts by *touching* them [37]. All hardforks required peers on the network to upgrade their software to continue participating in the network.

TheDAO hardfork precipitated the creation of Ethereum Classic (market cap of $2 bn, June 2017) as a distinct fork of Ethereum [16]. One of the reasons this split occurred was that a faction of the community disagreed in principle with modifying TheDAO smart contract in order to reverse the theft. An accidental split also occurred after the Spurious Dragon hardfork as both Geth and Parity (i.e. distinct implementations of the Ethereum protocol) failed to identically implement the new consensus rules. Geth was updated to fix a bug in order to resolve the fork and of course the forked blockchain was abandoned [17].

Ethereum Classic. There have been two hardforks in Ethereum Classic. Gas-Reprice replicated Ethereum's hardfork to increase the cost for underpriced EVM operation codes in order to prevent future spam attacks [18]. Die Hard removed the *difficulty time-bomb* that was hard-coded into Ethereum [19]. So far, there have been no accidental splits.

In the next section, we highlight that Ethereum's inclusive hardfork for TheDAO allowed an attacker to perform replay attacks against unprepared exchanges before presenting a survey of replay protection proposals for Bitcoin.

Other cryptocurrencies. We briefly note that other cryptocurrencies besides Bitcoin and Ethereum, such as Litecoin and Monero, have also endured softforks and hardforks. Monero notably has committed to regularly scheduled every six months (and therefore predictable) hardforks [32].

2.4 Replay Protection

A replay attack is when the sender signs a transaction with the intention that it is accepted into one blockchain (i.e. Fork-1), but it can also be accepted into an alternative blockchain (i.e. Fork-2). Thus, the purpose of replay protection is to permit users to decide which blockchain can accept their newly signed transactions. Unfortunately, the lack of replay protection after Ethereum's *TheDAO* hardfork caused some companies to lose a substantial number of Ethereum Classic coins (ETC). For example, a Chinese exchange YUNBI lost 40k ETC as a single transaction was unexpectedly accepted in both blockchains.

[6] The community disputes whether BIP50 (deployed in response to BIP34's accidental split) should be considered a hardfork, and therefore to what degree Bitcoin governance has established a precedent of avoiding hardforks.

Table 2. An overview of the replay protection proposals. ◐ highlights that this feature depends on whether the proposal was introduced via a softfork or hardfork.

Proposal	Any Fork	Fork-1 First	Prior to HF	Tx Format	Softfork
Transaction version	◐	◐	✗	✗	✓
Check block at height	✓	✗	✓	✗	✓
Sighash Enum	✓	✗	✗	✓	✗
Migration Input	✓	✗	✗	✗	✗
Chain ID	✓	✗	✗	✓	✗

In Ethereum, this incident led to the Spurious Dragon hardfork which introduced `chain_id` [8]. The sender is responsible for updating the transaction's `chain_id` to state which blockchain can accept it. On the other hand, several companies in Bitcoin have co-operatively signed a letter [20] to request replay protection in any future hardfork. We provide a survey on four approaches for replay protection from the community before proposing migration inputs below.

Transaction Version. All transactions have a version number that can be incremented to inform clients that a new feature is supported. For example, a recent softfork incremented the version number from 1 to 2 when the developers introduced relative lock times.[7] Both Harding [15] and Lau [22] proposed that a single bit in the transaction version can be re-purposed as an *opt-in/opt-out bit*. The sender can update this bit to dictate which blockchain can accept this transaction. However, the Fork-1 blockchain cannot respect this new consensus rule without a softfork. As a result the sender must first create a transaction that is only valid in Fork-1 before creating a second transaction for Fork-2.

Check Block At Height. Dashjr proposed a new Bitcoin script function `OP_CHECKBLOCKATHEIGHT`. This allows the sender to specify that a block hash (at a given height) must exist in the blockchain before this transaction can be accepted [11]. It was originally proposed to prevent double-spending and blockchain re-organization attacks. However, it can conceivably be used to decide whether a transaction can be accepted into Fork-1 or Fork-2. Although, the function must be introduced via a softfork for Fork-1 and a block hash after the hardfork must be known before transactions that spend "pre-fork" coins can be signed.

Sighash Enum. One approach proposed by Zander was to change the hash-type enum (i.e. `SIGHASH`[8]) to begin with 10 instead of zero [41].

[7] `OP_CHECKSEQUENCEVERIFY` [13] and a new consensus rule was introduced to only check for relative lock times if the transaction version number is two or higher [3].

[8] A marker in the transaction input to specify how to construct the transaction's hash before verifying the signature. For example, the transaction hash can contain no transaction outputs, all transaction outputs, or a 1:1 mapping of inputs/outputs.

The purpose is to change the transaction format such that all signed transactions are only valid in the forked blockchain Fork-2. The full proposal can be found here [12].

Chain ID. In a similar style to Ethereum it is feasible to incorporate a `chain_id`. This value can be included explicitly in the transaction as an additional field which allows all validating peers for Fork-1 to reject the transaction as its format is not valid, whereas peers for Fork-2 can confirm that `chain_id` is part of the signed message.

Migration Input. We propose a new consensus rule for the forked blockchain Fork-2 to include an additional transaction input when a transaction is spending "pre-fork" coins. Technically, it is a sentinel `41 byte` transaction input of zeros.[9] Of course, the previous transaction hash can be reduced from `32 bytes` to `1 byte` if structural changes to the transaction are implemented in the forked blockchain, and if so the overall cost per transaction is `10 bytes`. Peers conforming to the previous consensus rules will reject this transaction, while peers with the new set of consensus rules will accept it.

Compatibility. Table 2 presents a comparison of the proposals. The criteria is based on whether the sender can dictate if a transaction is accepted into Fork-1 or Fork-2, if the sender must first sign a transaction for Fork-1 before Fork-2, if a transaction must be stored in Fork-1 prior to the hardfork, if the transaction format must be changed or if a softfork in Fork-1 is necessary.

We highlight that Sighash Enum, Migration Input and the Chain ID proposals are compatible with our protocol as no new consensus rules is required for Fork-1 while the sender can explicitly dictate if a transaction is accepted into Fork-1 or Fork-2. As well, transaction version can be used if Fork-1 performs a softfork. On the other hand, `OP_CHECKBLOCKATHEIGHT` is not compatible as the block hash immediately after the hardfork must be available and thus prevents both parties setting up the atomic trade prior to the hardfork.

3 Bitcoin Hardfork Atomic Cross-Chain Trade

To set the scene, both Alice and Bob publicly commit to the atomic trade by depositing coins into a single transaction. Next, both parties co-operatively set up the atomic trade by signing off-chain transactions before the hard-fork activation time Δ_{FORK}[10]. After the hardfork has occurred, one party (i.e. Alice) is responsible for triggering the trade. If she fails to trigger the atomic trade, then Bob can claim all coins in both Fork-1 and Fork-2. Next, we present the Bitcoin's hardfork atomic trade protocol.

[9] Previous transaction hash as `32 bytes`, the previous transaction output index as `4 bytes`, the length of the script as `1 byte` and the sequence number as `4 bytes`.

[10] The activation time can be determined by a publicly announced flag day (i.e. similar to Bitcoin Cash [31]). There is a signalling process outlined in BIP9 [40], but this is designed for softforks. If the signalling process is used, then this atomic trade must be set up after the new rules are locked in.

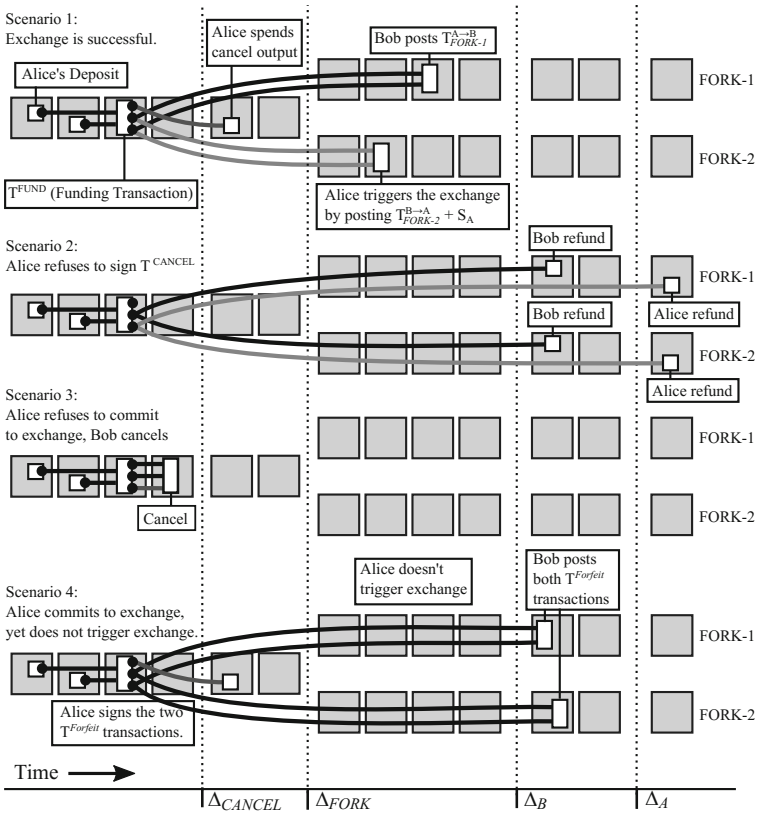

Fig. 1. High-level overview. Our protocol has four outcomes: (1). both parties successfully perform the trade, (2). Alice aborts the protocol shortly after T^{Fund} is accepted into the blockchain and both parties are refunded, (3). Bob cancels the atomic trade and both parties are immediately refunded, and (4). Alice forfeits her coins in both blockchains to Bob by not triggering the atomic trade

3.1 Proposed Protocol for Bitcoin

Table 6 presents the atomic trade protocol that permits two parties to exchange coins in the event of a hardfork. We present the establishment, off-chain setup and atomic trade aspects of the protocol below.

On-chain Establishment. Alice computes the secret S_A and hashes it $h_A = H(S_A)$ before both parties co-operatively deposit coins into a Funding Transaction T^{Fund}. This transaction has an output for Alice's deposit, Bob's deposit and an auxiliary output that we denote as Cancel Timer[11]. Both deposit outputs can be redeemed if either condition is satisified:

[11] Both parties deposit a sufficient number of coins in this output to cover a future transaction fee.

1. **Refund.** Each party is refunded their deposit if the trade times-out after time Δ_A for Alice or Δ_B for Bob.
2. **Transfer.** One party can claim the deposit if both parties have signed the transaction and S_A is revealed.
3. **Cancel.** A sentinel condition that cancels the atomic trade if it is redeemed simultaneously with the Cancel Timer output of this transaction.
4. **Forfeit (Alice Deposit Only).** Alice forfeits her deposit if she does not trigger the transfer by Δ_B.

Alice's refund time Δ_A must be after Bob's refund time Δ_B such that $\Delta_A > \Delta_B$. As well, both timers must be after the hardfork activation time Δ_{FORK} such that $\Delta_A, \Delta_B > \Delta_{FORK}$. This provides a grace period for Alice to reveal S_A (i.e. trigger the trade) and for Bob to find S_A to claim his coins.

The Cancel Timer output has a Cancel condition that can cancel the atomic trade if it is signed by both parties before Δ_{CANCEL}. Otherwise the output also has a Commit condition that allows Alice to single-handedly sign this output after Δ_{CANCEL}. The lock time Δ_{CANCEL} must expire before the hardforks activation time Δ_{FORK} such that $\Delta_{FORK} > \Delta_{CANCEL}$. This is to ensure the atomic trade is set up within a timely manner and before the hardfork. Finally the Funding Transaction T^{Fund} must achieve sufficient depth in the blockchain before both party's can co-operatively begin the off-chain setup.

Set up cancellation. Alice signs and sends Bob T^{Cancel}. This transaction satisfies the Cancel condition for all three outputs[12] of the Funding Transaction T^{Fund} and sends both parties their deposits The purpose of this transaction is to allow Bob to cancel the atomic trade if it is not set up before Δ_{CANCEL}. Alice can sign and broadcast T^{Commit} after Δ_{CANCEL} that spends the Cancel Timer's output (i.e. Commit condition) in order to invalidate T^{Cancel} and prevent Bob cancelling the atomic trade.

Set up trade. Alice signs and sends Bob $T^{A \to B}_{FORK1}$. This transaction spends both deposit outputs using the Transfer condition and sends all coins to Bob if the pre-image S_A is revealed and the transaction is accepted into the blockchain Fork-1 before Δ_B. Next, Bob signs and sends Alice $T^{B \to A}_{FORK2}$. This transaction spends both deposit outputs using the Transfer condition and will send all coins to Alice if the pre-image S_A is revealed and the transaction is accepted into the blockchain Fork-2 before Δ_A. As well, this transaction must incorporate relay protection such that it is only valid for the forked blockchain Fork-2. It is worth mentioning that the atomic trade can be performed after the hardfork activation time Δ_{FORK} if Alice broadcasts T^{Commit} to invalidate T^{Cancel}. However, Alice currently has an unfair advantage as she can abort the protocol (i.e. not reveal the pre-image S_A) and cancel the atomic trade without a penalty.

[12] Table 6 highlights that each transaction output in the Funding Transaction has a Cancel condition. For example, Alice's deposit can be spent if Cancel(PK_{A_3}, PK_{B_3}) is satisfied and the Cancel Timer can be spent if Cancel(PK_{A_2}, PK_{B_2}). is satisfied.

Set up forfeit. To overcome this fairness issue, Alice must sign and send Bob $T^{Forfeit}_{FORK1}$, $T^{Forfeit}_{FORK2}$. Both transactions spend Alice's deposit using the Forfeit condition and Bob's deposit using the Transfer condition. Of course, these transactions will send all coins to Bob in both Fork-1 and Fork-2. This allows Bob to penalize Alice for aborting the protocol (i.e. not triggering the trade before Δ_B). Furthermore, Alice must sign and send Bob both transactions before the lock time Δ_{CANCEL}. Otherwise, Bob is expected to cancel the atomic trade by signing and broadcasting T^{Cancel}.

Commit to Atomic Trade. Alice signs and broadcasts T^{Commit}. This transaction spends the Cancel Timer output using the Commit condition after Δ_{CANCEL} in order to invalidate the cancellation transaction T^{Cancel}. Thus, both parties are committed to performing the atomic trade.

Trigger Trade. After the hardfork activation time Δ_{FORK} Alice can claim both deposits in Fork-2 using $T^{B\rightarrow A}_{FORK2}$. This reveals S_A in Fork-2 and allows Bob to claim both deposits in Fork-1 using $T^{A\rightarrow B}_{FORK1}$.

Forfeit. As we mentioned previously Bob can penalise Alice if she does not trigger the transfer. He can broadcast the transactions $T^{Forfeit}_{FORK1}$, $T^{Forfeit}_{FORK2}$ after Δ_B to claim all coins in both blockchains Fork-1, Fork-2.

3.2 Distinct Keys

We highlight that the protocol is only secure if each condition in a transaciton output has a unique signing key i.e. $PK_{A_1}, ..., PK_{A_4}$. The core issue is that the message signed for a transaction output is the same regardless of the condition the signer intends to satisfy. This insecurity can be highlighted if we assume all conditions for Alice's deposit output rely on a single signing key PK_{A_1}.

Alice signs the transaction output that represents her deposit during the trade setup phase. She intends for her signature to satisify the Transfer(PK_{A_1}, PK_{B_2}, h_A) condition that sends her deposit to Bob if S_A is revealed. Unexpectedly, Bob can re-use her signature to also satisfy the forfeit condition Forfeit($PK_{A_1}, PK_{B_4}, \Delta_B$). This guarantees that he receives both deposits in the non-forked blockchain Fork-1 after Δ_B and thus he has no motivation to continue following the protocol.

4 Ethereum Hardfork Atomic Cross-Chain Trade

The key insight for this protocol is that both parties can deposit their coins into a smart contract. After the hardfork has occurred the contract can use a `Hardfork Oracle` to determine whether it is on the blockchain Fork-1 or Fork-2 before sending each respective party their coins. In this section, we discuss how to construct `Hardfork Oracles` before presenting the protocol.

Table 3. Ethereum Hardfork Atomic Cross-Chain Trade. Both parties deposit coins into the `Trade` Contract. This contract can detect if it is on Fork-1, Fork-2 using `Hardfork Oracle` contract before sending the deposits.

1. Alice and Bob agree on the refund lock time Δ_{REFUND} based on the fixed hardfork time Δ_{FORK}
2. Alice creates the `Trade` contract that specifies the deposits d_a, d_b required by both parties, the refund lock time Δ_{REFUND} and `Hardfork Oracle` contract's address σ
3. The contract locks both parties into the exchange once the deposits d_a, d_b are confirmed
4. **Both parties wait for the hardfork at time Δ_{FORK}**
5. Alice signs $T^{B \rightarrow A}_{FORK2}$ and claims both deposits in the forked blockchain Fork-2 before Δ_{REFUND}
– Contract communicates with `Oracle Hardfork` contract to confirm this is the forked blockchain Fork-2
6. Bob signs $T^{A \rightarrow B}_{FORK1}$ and claims both deposits in the non-forked blockchain Fork-1 before Δ_{REFUND}
– Contract communicates with `Oracle Hardfork` contract to confirm this is the non-forked blockchain Fork-1

4.1 Hardfork Oracle

We propose that a `Hardfork Oracle` contract can be used to distinguish whether it is on Fork-1 or Fork-2 without the need for a trusted third party. There are two approaches to realize this oracle:

Detection within contract. As mentioned in Sect. 2.4, Ethereum has implemented replay protection in the form of a `chain_id`. The simplest approach is for the contract to query `tx.chain_id` to determine if the transaction was accepted into Fork-1 or Fork-2. Unfortunately, the `chain_id` cannot yet be programmatically accessed by the contract's code.

The Ethereum Community have also proposed the concept of an oracle contract that can detect the activation of a hardfork and have provided an example for TheDAO hardfork [25]. This contract checks `TheDAO`'s contract balance after the publicly announced hardfork time Δ_{FORK} to determine if the contract is in Fork-2 (i.e. the balance is reverted to reverse the theft) or Fork-1 (i.e. the coins remain stolen and the balance has not changed).

Detection outside contract. One approach is that the user can provide the contract evidence that a transaction with the desired `chain_id` was accepted into the blockchain after the Δ_{FORK}. This evidence can be a confirmed transaction alongside its patricia tree branch and the respective block's header. The contract can verify that the transaction is accepted in the respective block before confirming that it is in the blockchain's most recent 256 blocks. Finally, the contract

can extract the `chain_id` from the transaction and determine if this blockchain is Fork-1 or Fork-2.

Future Hardforks. Ethereum have recently approved changes that will be included as a hardfork in the future. This includes EIP96 [6] that proposes extending `block.blockhash` to return hashes that are more than 256 blocks deep and EIP98 [9] that proposes removing the intermediate state value from a transaction's receipt. We highlight that a hardfork for EIP96 can be detected within a contract as `block.blockhash(257)` will either return 0 for the oracle contract on Fork-1 or the respective block hash for the oracle contract on Fork-2. On the other hand, a hardfork for EIP98 can be detected in a similar manner to `chain_id` by providing a transaction receipt, patricia tree branch and the respective block header. The contract can verify if the intermediate root's value is removed (or set to 0) to decide if it is on Fork-1 or Fork-2. We leave it for future work to determine if oracles can be built to detect gas changes or new functions (i.e. opcodes).

4.2 Proposed Protocol for Ethereum

Given a `Hardfork Oracle` we can perform an Atomic Cross-Chain Trade in Ethereum and the protocol is presented in Table 3. We briefly explain how to establish the atomic trade prior to the hardfork, how to perform the trade using the hardfork oracle and why the trigger is no longer necessary.

Establishment. First, Alice establishes the `Trade` contract and specifies the required deposits d_a, d_b, the timers $\Delta_{FORK}, \Delta_{REFUND}$, and the `Hardfork Oracle`'s address. Finally, both parties deposit their coins into the contract before the hardfork activation time Δ_{FORK}.

Atomic Trade. Both Alice and Bob must claim both deposits from the `Trade` contract during the grace period between Δ_{FORK} and Δ_{REFUND}. Otherwise, either party can withdraw their deposit from the contract after Δ_{REFUND}. Notably, at the time of withdrawal, the `Trade` contract contacts the `Hardfork Oracle` to determine if this blockchain is Fork-1 or Fork-2.

Triggering Trade. The Bitcoin protocol's trigger served two purposes. The first was to ensure both deposits could not be spent until the activation of the hardfork, and the second was to ensure the trade was only conducted if the hardfork occurred. The `Trade` contract can enforce both purposes without a trigger as the contract can detect which fork it is on after the hardfork activation time Δ_{FORK}. Most importantly, this also removes the requirement for a synchronised clock for both Fork-1, Fork-2 in order to perform the atomic trade.

5 Discussion

In this section, we discuss the requirement for a synchronised global clock, the potential for miner censorship and bribery attacks, and the impact of transaction malleability for designing our protocols.

Synchronised Time. Unlike the TierNolan protocol, the Bitcoin atomic trade protocol in this paper does not rely on both blockchains having a synchronised block height or timestamp in order to co-ordinate and enforce the atomic trade's fair exchange. We highlight that Bob is guaranteed to receive his coins in Fork-1 after Δ_B using the forfeiture transaction $T^{Forfeit}_{FORK1}$. The only crucial timer is Δ_B in Fork-2 that dictates when Alice should reveal S_A to claim both deposits. As a result, both the block height and the median time of the previous 11 blocks is suitable for our protocol.

In Ethereum, no single party is responsible for triggering the trade and the Δ_{REFUND} timer used by the `transfer` contract is independent for both blockchains. It is feasible for miner's to slow the passage of time although this simply increases the affected party's grace period to claim both deposits.

Miner Censorship. A cartel of miners have the authority to censor transactions in both Bitcoin and Ethereum. This censorship permits miners to interfere with the atomic trade and coerce either party to share a portion of their deposit. To illustrate for the Bitcoin protocol, it is feasible for miners in blockchain Fork-2 to simply censor $T^{B\rightarrow A}_{FORK2}$ if Alice refuses to pay a bribe. At the same time, Bob can agree to pay this bribe by sending the miners a new bribery transaction which is only valid if the forfiture transaction $T^{Forfeit}_{FORK2}$ is accepted into Fork-2 after Δ_B. In Ethereum, miners can simply stop the atomic trade by preventing both parties depositing or withdrawing the contract's coins. It is worth mentioning that bribery and censorship attacks also violate the security guarantees for timelock based atomic cross-chain trade protocols/off-chain payment channels.

Hardfork Time. The hardfork's activation time Δ_{FORK} must be fixed to permit both parties to agree suitable lock times for the atomic trade. Alice must only sign the forfeit transactions if she is confident the hard-fork activation time Δ_{FORK} will not be delayed. Otherwise, the delay can result in $\Delta_{FORK} > \Delta_B$ for the Bitcoin protocol. This allows Bob to claim the deposits in both blockchains Fork-1 and Fork-2 using the forfeiture transactions. On the other hand, in Ethereum, both parties mutually agree upon a single Δ_{REFUND} and if the hard-fork is delayed until after this time then both parties are refunded.

Transaction Malleability. The atomic trade protocol for Bitcoin is designed to account for transaction malleability which is why both parties are required to co-operatively sign cancellation, trade and forfeit transactions after T^{Fund} is stored in the blockchain. If transaction malleability is fixed, then it is feasible to simplify the protocol such that only the trade transactions $T^{A\rightarrow B}_{FORK1}$, $T^{B\rightarrow A}_{FORK2}$ need to be signed off-chain before both parties co-operatively sign and broadcast the funding transaction (protocol in full paper). (see Appendix B for details). On the other hand, the Ethereum protocol is not impacted by transaction malleability as the contract can store the current state of the atomic trade and parties are not required to co-operatively authorise transactions.

Nature of the Bet. It is important to distinguish if both parties are betting that the hardfork activates at Δ_{FORK}, or if both parties are betting whether Fork-1 or Fork-2 will be more valuable if the hard-fork occurs. Our protocol is focused

on the former bet as Alice only signs the forfeit transactions once she is confident the hardfork will activate at time Δ_{FORK}. If the hardfork does not occur at time Δ_{FORK}, then she forfeits her deposit to Bob. It is feasible to perform the latter bet (i.e. refund both parties if the hardfork does not occur) if Alice does not perform the final forfeiture step. However, this has a fairness issue as Alice can evaluate whether to perform the trade or to abort the protocol (i.e. not to reveal S_A) and cancel the atomic trade.

6 Conclusion

In this paper, we propose the first protocol that can commit two parties to swapping "pre-fork" coins before a hardfork activates, and then enforce the swap after the hardfork has occurred without the assistance of a trusted third party. Our protocols are inspired by real-world events as Loaded and Roger voiced interest in atomically trading 120 k bitcoins (i.e. approximately $120 m USD at the time) to effectively gamble on the success of a future hardfork in Bitcoin.

We show how to realize the atomic trade protocols in Bitcoin and Ethereum. The former relies on the hardfork deploying replay protection and a global clock, whereas the latter simply leverages a `Hardfork Oracle` contract that allows another contract to detect if it is in blockchain Fork-1 or Fork-2. Finally, also we provided a detailed survey on the history of soft/hard forks for Bitcoin, Ethereum and Ethereum Classic, and a survey on proposed replay protection mechanisms in Bitcoin.

Acknowledgements. We thank Nick Johnson for bringing to our attention hardfork oracles, Tadge Dryja for his comments and criticisms, Roger Ver for allowing us to use his name in the paper's title, Iddo Bentov for insightful discussions and #bitcoin-wizards IRC channel for answering questions regarding forks. Patrick McCorry is supported by EPSRC grant EP/N028104/1, Ethan Heilman is supported by NSF 1350733.

A TierNolan's Atomic Cross-Chain Trading Protocol

Table 4 presents the protocol proposed by TierNolan in 2013. The TierNolan protocol allows two parties to atomically exchange coins across two blockchains. Such protocols are called Atomic Swaps because the two transactions happen *atomically i.e.,* either swap occurs or it does not. It requires both parties to deposit coins in separate blockchains and for one of the parties to trigger the exchange.

Brief overview. Consider two blockchains, the first blockchain we denote as Fork-1, the second blockchain we denote as Fork-2. Alice wants to trade her coins on blockchain Fork-1 for coins which Bob controls on blockchain Fork-2. First, Alice picks a random value S_A, hashes it to $H(S_A)$ and deposits her coins into a transaction T_1 which is confirmed on Fork-1. Bob can claim the funds in T_1 if and only if he learns the value S_A. However if the coins in T_1 are still unspent by Δ_A Alice can refund the coins in this transaction back to herself.

Table 4. Atomic Cross-Chain Trading Protocol. This protocol allows two parties to atomically exchange coins across two distinct blockchains

1. Alice and Bob agree on lock times Δ_A, Δ_B

2. Alice picks random S_A, hashes it $h_A = H(S_A)$ and constructs T_1 with her deposit. This transaction has a single output:
   – **Alice's deposit:** Output script is (PK_{A_1}, Δ_A)[a] OR $(H(S_A), B)$

   This transaction is signed by Alice and published to the first blockchain Fork-1

3. Bob has knowledge of $H(S_A)$ and creates T_2 with his deposit. This transaction has a single output:
   – **Bob's deposit:** Output script is (PK_{B_1}, Δ_B) OR $(H(S_A), A)$

   This transaction is signed by Bob and published to the second blockchain Fork-2
   It is important that $\Delta_A > \Delta_B$ by a sufficient margin

4. Alice must claim Bob's deposit before Δ_B. She signs a transaction that spends T_2, includes S_A and publishes it to the second blockchain Fork-2

5. Bob must claim Alice's deposit before Δ_A. He learns S_A, signs a transaction that spends T_1 and publishes it to the first blockchain Fork-1

[a] The original protocol had an explicit refund transaction that Bob was required to sign. It is possible to remove this step using `CHECKLOCKTIMEVERIFY`.

B Proposed Bitcoin Protocol if Transaction Malleability is fixed

Table 5 presents the Bitcoin hard-fork atomic cross-chain protocol if transaction malleability is fixed. It assumes that both parties can use replay protection to dic-

Table 5. Bitcoin's Hard Fork Atomic Cross-Chain Trade if transaction malleability is fixed. Both parties authorise the atomic trade transactions prior to signing and broadcasting the funding transaction T^{Fund}. Both trade transactions can be accepted into their respective blockchain after the hard-fork activation time Δ_{FORK}.

1. **Funding Transaction.** Alice and Bob agree on lock times Δ_A, Δ_B that should be sufficiently after the hardfork activation time Δ_{FORK}

2. Alice constructs a transaction we call *the Funding Transaction* or T^{Fund}. This transaction requires a deposit from each parties and has a single output:
   – **Deposits:** Trade$(PK_{A_1}, PK_{B_1}, \Delta_{FORK} + 1)$

3. Alice signs and sends Bob $T^{A \to B}_{FORK1}$. This transaction is only valid in Fork-1 and cannot be accepted into the blockchain until after the hard-fork activation time Δ_{FORK}

4. Bob signs and sends Alice $T^{B \to A}_{FORK2}$. This transaction is only valid in Fork-2 and cannot be accepted into the blockchain until after the hard-fork activation time Δ_{FORK}

5. Both parties co-operatively sign and broadcast T^{Fund} for acceptance into the blockchain

6. **Atomic Trade.** Both parties can broadcast $T^{A \to B}_{FORK1}$, $T^{B \to A}_{FORK2}$ after the hard-fork activation time Δ_{FORK}

Table 6. Bitcoin's Hard Fork Atomic Cross-Chain Trade. Our proposed protocol commits both Alice and Both to the trade prior to the hardfork's activation

1. **Funding Transaction.** Alice and Bob agree on lock times Δ_A, Δ_B that should be sufficiently after the hardfork activation time Δ_{FORK}

2. Alice picks random S_A, hashes it $h_A = H(S_A)$ and constructs a transaction we call *the Funding Transaction* or T^{Fund}. This transaction requires a deposit from each parties and has three outputs:

– **Alice's deposit:** Refund(PK_{A_1}, Δ_A) OR Transfer$(PK_{A_2}, PK_{B_2}, h_A)$ OR Cancel(PK_{A_3}, PK_{B_3}) OR Forfeit$(PK_{A_4}, PK_{B_4}, \Delta_B)$

– **Bob's deposit:** Refund(PK_{B_1}, Δ_B) OR Transfer$(PK_{A_2}, PK_{B_2}, h_A)$ OR Cancel(PK_{A_3}, PK_{B_3})

– **Cancel timer:** Commit$(PK_{A_1}, \Delta_{CANCEL})$ OR Cancel(PK_{A_2}, PK_{B_2})

This transaction must be accepted into the blockchain before Δ_{FORK} and achieve sufficient depth before performing the next step

3. **Set up cancellation.** Alice signs and sends Bob T^{Cancel}. This transaction spends all three outputs using the Cancel condition and sends both parties their deposit. He can sign and broadcast T^{Cancel} before Δ_{CANCEL} to cancel the atomic swap

4. **Off-chain setup.** Alice signs and sends Bob $T^{A \to B}_{FORK1}$, and Bob signs and sends Alice $T^{B \to A}_{FORK2}$. Both transactions spend Alice's and Bob's deposit outputs using the Transfer condition

5. **Set up forfeit:** Alice signs and sends Bob two transactions $T^{Forfeit}_{FORK1}$ and $T^{Forfeit}_{FORK2}$. Both transaction's spend Alice's deposited coins using the Forfeit condition and is valid after time Δ_B

6. **Commit to Atomic Trade.** If Alice does not sign and send the forfeit transactions before time Δ_{CANCEL} then Bob must sign and broadcast T^{Cancel}. Otherwise, she signs and broadcasts T^{Commit} after time Δ_{CANCEL}. This transaction effectively invalidates T^{Cancel} by spending the Cancel Timer output using the Commit condition

7. Both parties wait for the hardfork at time Δ_{FORK}

8. **Trigger Trade.** If Alice triggers the trade:

(a) Alice signs $T^{B \to A}_{FORK2}$, reveals S_A and claims both deposits in the forked blockchain Fork-2 before Δ_B

(b) Bob finds S_A, signs $T^{A \to B}_{FORK1}$ and claims both deposits in the non-forked blockchain Fork-1 before Δ_A

8. **Forfeit.** If Alice does not trigger the trade by Δ_B:

(a) Bob signs $T^{Forfeit}_{FORK1}$ and $T^{Forfeit}_{FORK2}$ claims both deposits in Fork-2 and Fork-1

tate if a transaction can be accepted into Fork-1 or Fork-2. As we will soon see this variation is significantly simpler compared to the protocol outlined in Sect. 3.

Briefly, both parties co-operatively create a Funding Transaction T^{Fund} and the two trade transactions $T^{A \to B}_{FORK1}$, $T^{B \to A}_{FORK2}$. Next, both parties must exchange signatures for the trade transactions before signing and broadcasting the funding transaction. Finally, both parties wait until after the hard-fork activation time Δ_{FORK} to claim both deposits in their respective blockchain.

This approach follows a similar style to payment protocols such as Duplex Micropayment Systems and Lightning as the off-chain's transactions are signed prior to the funding transaction. The order of signing off-chain transactions does not necessarily matter as these transactions are only valid if the funding transaction is accepted into the blockchain. Furthermore, it is worth highlighting that this approach does not require one party (i.e. Alice) to reveal a pre-image S_A or to sign cancel/forfeit transactions.

References

1. Andresen, G.: Block v2, Height in Coinbase, July 2012. https://github.com/bitcoin/bips/blob/6925e66aa092d97f8273e4bab15bb0d4c63f9ac9/bip-0034.mediawiki
2. Andresen, G.: March 2013 Chain Fork Post-Mortem, March 2013. https://github.com/bitcoin/bips/blob/master/bip-0050.mediawiki
3. BitcoinCore. Accept Transaction version 2 or more, June 2017. https://github.com/bitcoin/bitcoin/blob/1088b02f0ccd7358d2b7076bb9e122d59d502d02/src/consensus/tx_verify.cpp#L45
4. blockchain.info. Blockchain charts, March 2017. https://blockchain.info/charts
5. Buterin, V.: Bitcoin Network Shaken by Blockchain Fork, March 2013. https://bitcoinmagazine.com/articles/bitcoin-network-shaken-by-blockchain-fork-1363144448/
6. Buterin, V.: Blockhash refactoring, April 2016. https://github.com/ethereum/EIPs/issues/98
7. Buterin, V.: Long-term gas cost changes for IO-heavy operations to mitigate transaction spam attacks, September 2016. https://github.com/ethereum/EIPs/issues/150
8. Buterin, V.: Simple replay attack protection, October 2016. https://github.com/ethereum/eips/issues/155
9. Buterin, V.: Removal of intermediate state roots from receipts, February 2017. https://github.com/ethereum/EIPs/pull/210
10. coinmarketcap.com. CryptoCurrency Market Capitalizations: Bitcoin, March 2017. https://coinmarketcap.com/currencies/bitcoin/
11. Dashjr, L.: OP_CHECKBLOCKATHEIGHT, September 2016. https://github.com/luke-jr/bips/blob/bip-cbah/bip-cbah.mediawiki
12. Deadalnix. Add spec for UAHF, June 2017. https://github.com/Bitcoin-UAHF/spec/blob/master/replay-protected-sighash.md
13. Friedenbach, M., BtcDrak, Dorier, N., Kinoshitajona: Relative lock-time using consensus-enforced sequence numbers, May 2015. https://github.com/bitcoin/bips/blob/master/bip-0068.mediawiki
14. Goldfeder, S., Bonneau, J., Gennaro, R., Narayanan, A.: Escrow protocols for cryptocurrencies: how to buy physical goods using bitcoin. In: Financial Cryptography and Data Security. Springer (2017)

15. Harding, T.: [bitcoin-dev] Proposal: Hard fork opt-out bits, July 2016. https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2016-July/012917.html
16. Hertig, A.: Ethereum Executes Blockchain Hard Fork to Return DAO Funds, July 2016. http://www.coindesk.com/ethereum-classic-explained-blockchain/
17. Hertig, A.: How Developers Are Responding to Ethereum's Unexpected Fork, December 2016. http://www.coindesk.com/developer-response-ethereum-fork/
18. Hertig, A.: The Blockchain Created By Ethereum's Fork is Forking Now, October 2016. http://www.coindesk.com/ethereum-classic-blockchain-fork-ddos-attacks/
19. Hertig, A.: Ethereum Classic Freezes 'Difficulty Bomb' With 'Diehard' Fork, January 2017. http://www.coindesk.com/ethereum-classic-diehard-fork/
20. S. Higgins. Bitcoin Exchanges Unveil Hard Fork Contingency Plan, March 2017. http://www.coindesk.com/ethereum-executes-blockchain-hard-fork-return-dao-investor-funds/
21. Kerin, T., Friedenbach, M.: Median time-past as endpoint for lock-time calculation, August 2015. https://github.com/bitcoin/bips/blob/master/bip-0113.mediawiki
22. Lau, J.: [bitcoin-dev] Anti-transaction replay in a hardfork, January 2017. https://github.com/bitcoin/bips/blob/master/bip-0068.mediawiki
23. Loaded. @RogerVer lets make a deal. At least 60k, my BTU for your BTC, March 2017. https://bitcointalk.org/index.php?topic=1836672.0
24. Lombrozo, E., Lau, J., Wuille, P.: Segregated Witness (Consensus layer), December 2015. https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki
25. Maersk, N.: The DAO Hard Fork Oracle, July 2016. https://github.com/veox/solidity-dapps/blob/TheDAOHardForkOracle-v0.1/TheDAOHardForkOracle/TheDAOHardForkOracle.sol
26. Maxwell, G.: Capacity increases FAQ, December 2015. https://bitcoin.org/en/bitcoin-core/capacity-increases-faq#roadmap
27. Maxwell, G., Wilcke, P.W.: Conversation on Bitcoin Wizards, April 2017. https://botbot.me/freenode/bitcoin-wizards/2017-04-20/?msg=84304042&page=3
28. McCorry, P., Shahandashti, S.F., Hao, F.: A smart contract for boardroom voting with maximum voter privacy. In: Financial Cryptography and Data Security. Springer (2017)
29. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008). https://bitcoin.org/bitcoin.pdf
30. Poon, J., Dryja, T.: The bitcoin lightning network, February 2015. https://lightning.network/
31. Song, J.: Timeline, The Bitcoin Cash: What Will Happen When. https://www.coindesk.com/bitcoin-cash-what-expect-fork-10000-foot-view/
32. Spagni, R.: A formal approach towards better hard fork management (2015). https://forum.getmonero.org/4/academic-and-technical/303/a-formal-approach-towards-better-hard-fork-management
33. Tier, N.: Re: Alt chains and atomic transfers, May 2013. https://bitcointalk.org/index.php?topic=193281.msg2224949#msg2224949
34. Todd, P.: OP_CHECKLOCKTIMEVERIFY, October 2014. https://github.com/bitcoin/bips/blob/master/bip-0065.mediawiki
35. ViaBTC. Miner guide: how to safely hard fork to bitcoin unlimited, October 2016. https://medium.com/@ViaBTC/miner-guide-how-to-safely-hard-fork-to-bitcoin-unlimited-8ac1570dc1a8
36. Wilcke, J.: Homestead Release, February 2016. https://blog.ethereum.org/2016/02/29/homestead-release/
37. Wood, G.: State trie clearing (invariant-preserving alternative), October 2016. https://github.com/ethereum/EIPs/issues/161

38. Wuille, P.: Duplicate transactions, February 2012. https://github.com/bitcoin/bips/blob/6925e66aa092d97f8273e4bab15bb0d4c63f9ac9/bip-0030.mediawiki
39. Wuille, P.: Strict DER Signatures, January 2015. https://github.com/bitcoin/bips/blob/6925e66aa092d97f8273e4bab15bb0d4c63f9ac9/bip-0066.mediawiki
40. Wuille, P., Todd, P., Maxwell, G., Russell, R.: Version bits with timeout and delay, October 2015. https://github.com/bitcoin/bips/blob/master/bip-0009.mediawiki
41. Zander, T.: Replay Protection Guidance, March 2017. https://bitco.in/forum/threads/replay-protection-guidance.1930/