Catena: Preventing Lies with Bitcoin

Alin Tomescu
MIT CSAIL
alinush@mit.edu

Srinivas Devadas
MIT CSAIL
devadas@mit.edu

***Abstract*—We present *Catena*, an efficiently-verifiable Bitcoin witnessing scheme. Catena enables any number of thin clients, such as mobile phones, to efficiently agree on a log of application-specific statements managed by an adversarial server. Catena implements a log as an `OP_RETURN` transaction chain and prevents forks in the log by leveraging Bitcoin's security against double spends. Specifically, if a log server wants to equivocate it has to double spend a Bitcoin transaction output. Thus, Catena logs are as hard to fork as the Bitcoin blockchain: an adversary without a large fraction of the network's computational power cannot fork Bitcoin and thus cannot fork a Catena log either. However, different from previous Bitcoin-based work, Catena decreases the bandwidth requirements of log auditors from 90 GB to only tens of megabytes. More precisely, our clients only need to download all Bitcoin block headers (currently less than 35 MB) and a small, 600-byte proof for each statement in a block. We implemented Catena in Java using the *bitcoinj* library and used it to extend CONIKS, a recent key transparency scheme, to witness its public-key directory in the Bitcoin blockchain where it can be efficiently verified by auditors. We show that Catena can be used to secure many systems today such as public-key directories, Tor directory servers or software transparency schemes.**

I. Introduction

Security often depends on systems not being able to lie, a property known as *non-equivocation*. For example, when a Certificate Authority (CA) lies by signing fake certificates, it can impersonate websites and compromise users' privacy. In fact, this has happened many times in the past [1]–[7]. To prevent lies, Certificate Transparency (CT) [8] has been introduced as a way of publicly-logging all CA-issued certificates. However, a CT log server can still lie about the log of issued certificates and, together with a colluding CA, can launch impersonation attacks. While gossiping [9] about the log can help detect these lies, detection can be slow or not happen at all, as gossip messages can be delayed indefinitely. Another example is the Tor [10] anonymity network, where if directory servers lie then Tor users can be tricked to use malicious Tor relays and get deanonymized [11]. Thus, we believe non-equivocation is an important security requirement in many systems today.

Unfortunately, without online trusted parties, achieving non-equivocation is impossible [12]. To deal with this impossibility result, systems resort to enforcing a weaker property called *fork consistency* [12]. Fork-consistent systems essentially make lies "permanent" and thus easier to prove later when clients are able to communicate or "gossip" out-of-band. However, as illustrated above, for many systems such as public-key directories or Tor directory servers, undetected

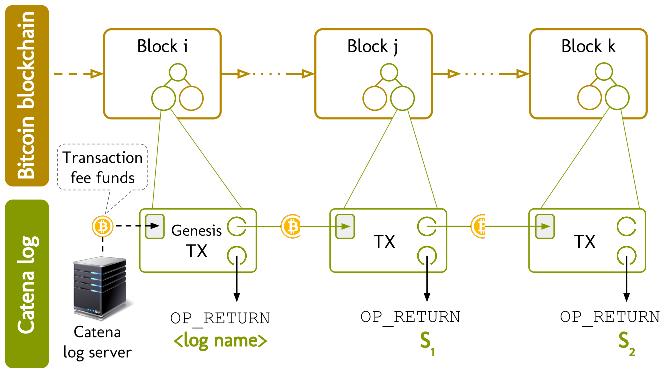

Figure 1. A Catena log is a chain of Bitcoin transactions. Each Catena transaction has two outputs: (1) a continuation output, which will be spent by the next Catena transaction, thus creating a chain and (2) an `OP_RETURN` output, which commits some application-specific statement. The server pays Bitcoin transaction fees for each issued statement. For applications that publish statements often, batching can be used to keep the fee per statement low.

lies can seriously impact users' security. Thus, we believe a more proactive approach [13] to security is desirable for such systems.

To prevent equivocation proactively, recent work [14]–[16] uses the Bitcoin blockchain [17], [18], as a witness. We believe this *Bitcoin witnessing* approach, though currently inefficient, is promising for three reasons. First, this approach makes equivocation as hard as forking the Bitcoin blockchain itself, which has proven resistant to forking attacks. Second, this approach only relies on a single global witness, namely the Bitcoin blockchain, obviating the need for users to obtain correct cryptographic identities of multiple trusted entities such as log providers and auditors as in CT [8], or witnesses as in CoSi [13]. It also has the advantage of not requiring the witness to keep any secrets, which if compromised would result in equivocation. Third, the Bitcoin blockchain's open, decentralized and censorship-resistant nature makes deployment of witnessing schemes easy and interference with them hard. Unfortunately, the main drawback of Bitcoin witnessing has been that verifiers have to download the entire Bitcoin blockchain, which at the time of this writing is almost 90 GB [19] in size.

This paper presents *Catena*, an efficient Bitcoin-based witnessing scheme which dramatically reduces auditors' bandwidth overhead. At a high-level, Catena is a tamper-evident log [20] built on top of the Bitcoin blockchain. Catena prevents adversarial log servers who cannot fork the Bitcoin blockchain from equivocating about a log of application-specific statements. Importantly, auditors who run Catena clients can check the log for non-equivocation efficiently via Simplified Payment Verification (SPV) [17]. This drastically decreases the bandwidth requirements of log auditors from 90 GB [19] to only tens of megabytes, as Catena clients only need to download Bitcoin block headers and small Merkle proofs under some of those headers. Furthermore, after all block headers are downloaded, the bandwidth for auditing decreases to less than 1 KB of data every 10 minutes.

A. Efficient Non-equivocation via Bitcoin

Previous Bitcoin witnessing schemes [14], [15] are inefficient because they cannot prove non-membership of inconsistent statements unless auditors download all the transactions in the Bitcoin blockchain. Our design addresses this issue by allowing Catena clients to skip downloading all irrelevant transactions while still guaranteeing non-equivocation. *The key idea behind Catena is that Bitcoin's mechanism for preventing double spends can actually be regarded as a non-membership proof.* Specifically, Bitcoin proves that no transactions double spending a previous transaction's output exist. That is, if a client verifies blockchain membership for a transaction tx_2 which spends a previous transaction output $tx_1[0]$, that client has also implicitly verified that no other transaction tx'_2 which spends $tx_1[0]$ exists in the blockchain[1].

Catena turns this idea into a non-equivocation scheme. Each *Catena transaction* stores exactly one statement and spends the previous Catena transaction, creating a chain of statements as shown in Figure 1. This implies that if an auditor sees a statement s_i in the blockchain whose transaction correctly spends the transaction for the previous statement s_{i-1}, then *that constitutes a non-membership proof that no other inconsistent statement s'_i exists*. Looked at differently, if an adversarial log server wants to equivocate about s_i, it has to double spend the previous Catena transaction for s_{i-1}, which can only be done by forking the Bitcoin blockchain.

1) Root-of-Trust: Catena guarantees that once a client correctly obtains a log's *genesis transaction*, the server cannot equivocate about that log unless it forks the Bitcoin blockchain. The genesis transaction is the first transaction in the log and acts as the root-of-trust or "public key" for a Catena log (see Section IV-A). Once clients obtain the correct genesis transaction they can efficiently verify that every issued statement comes from a transaction that spends coins originating from the genesis transaction. In Section IV, we explain how this implicitly prevents equivocation in a Catena log. Our design is simple, efficient and obviates the need for log servers and clients to download the full Bitcoin blockchain while ensuring the consistency of the log.

2) Bitcoin-friendly: To embed log statements in Bitcoin transactions, Catena uses provably-unspendable `OP_RETURN` transaction outputs [21] which, unlike previous work [15], [22], does not harm Bitcoin by polluting the unspent transaction output (UTXO) set on Bitcoin nodes. To avoid putting stress on the Bitcoin P2P network, Catena clients query the log server directly to find out about statements and also query a *header relay network* (HRN) to obtain the latest Bitcoin block headers (see Section IV-C). Nodes for the header relay network can be run by anybody and are only trusted for availability and freshness. This is because lying about headers requires forging them, which is computationally prohibitive (see Bitcoin mining in Section II-B4).

3) Applications: Due to Bitcoin's 10-minute block rate, Catena can only issue a statement every 10 minutes. Still, even at this slower rate of issuing statements, Catena can help secure many applications which depend on non-equivocation. Catena can prevent equivocation attacks in key transparency systems such as CT [8], ECT [23], DTKI [24], ARPKI [25], AKI [26] and CONIKS [16]. Catena can also be used to implement Tor Consensus Transparency [11], to implement a simple yet resilient software transparency scheme [27], or as consensus protocol for n mutually distrusting servers. We discuss these applications in more detail in Section II-A and discuss Catena's application-agnostic nature in Section VII-3.

4) Implementation: To demonstrate the feasibility of Catena, we implemented a prototype in 3000 lines of Java using the bitcoinj Simplified Payment Verification (SPV) library [28]. The Catena log server starts with some Bitcoin funds and issues statements to the Bitcoin blockchain for clients to audit. Catena clients connect to the Bitcoin P2P network and use Bloom filtering [29] to hear about relevant transactions issued by the Catena server. Importantly, clients verify that transactions are backed by sufficient proof-of-work and correctly chained together, thus preventing equivocation. We also analyze the Bitcoin transaction fees the server has to pay per issued statement and show they could be anywhere between 7 to 12 cents per statement. Since existing systems like Keybase [14] already pay close to 7 cents per transaction, we believe this cost is practical. In the future, we plan on scaling our Catena prototype by allowing clients to fetch statements directly from the log server and to fetch block headers from a header relay network. Finally, we used our prototype to add Bitcoin witnessing to CONIKS [16], a recent key transparency scheme (see Section VI-D), and demonstrate the ease of using Catena.

B. Contributions and Organization

To summarize, this paper makes the following contributions:

- A new, *efficient* approach to transparency based on witnessing in the Bitcoin blockchain.
- Catena, an append-only log built on top of Bitcoin that is efficiently verifiable by thin clients, obviating the need to download the full Bitcoin blockchain.
- A prototype implementation of Catena in Java that can be used by applications today.

Organization. We motivate Catena and present the Bitcoin background necessary to understand our design in Section II. We describe our system model, threat model and goals in

[1]Here, we use $tx_1[0]$ to refer to output #0 of transaction tx_1 (see Section II-B5 for background on Bitcoin transactions)

Section III. We present Catena's design in Section IV and we discuss attacks and countermeasures in Section V. We discuss our prototype implementation, its overheads and our extension of CONIKS in Section VI. We discuss remaining issues and future work in Section VII. We go over related work in Section VIII and we conclude in Section IX.

II. Background and Motivation

In this section we discuss our motivation for designing Catena and give the necessary background on Bitcoin needed to understand Catena's design.

A. Motivation

Our main motivation for designing Catena is to provide proactive security to many applications that depend on it. We describe these applications and how Catena can secure them in this subsection. At the same time, we wanted to improve previous blockchain-based transparency schemes [14], [30] whose shortcomings we describe in Section II-A2. Finally, we wanted a non-equivocation scheme that does not require many trustworthy parties to come into existence as in [13], [16] and can be deployed today. In that sense, we only assume the Bitcoin blockchain is a trustworthy witness and show how to leverage it efficiently to prevent equivocation.

1) Key Transparency: Catena can prevent equivocation attacks in current key transparency work [8], [16], [23]–[26] and, as a result, thwart man-in-the-middle (MITM) attacks. Key transparency schemes bundle public key bindings together into a directory implemented using authenticated data structures [31]. Users are presented with commitments of the directory as it evolves over time and can verify someone's public key against a commitment of the directory, preventing equivocation with respect to that commitment. The remaining problem for key transparency schemes is to prevent equivocation about the commitments themselves. For this, current schemes rely on users gossiping between themselves [9], [16], [23], [24], users gossiping with trusted validators [26], federated trust [16], any-trust assumptions [25] or non-collusion between actors [25], [26].

With Catena, we propose using the Bitcoin blockchain as a hard-to-coerce, trustworthy witness that can vouch for directory commitments. For example, in Certificate Transparency (CT), a log server would directly witness *signed tree heads* (STHs) in Bitcoin via a Catena log. Users can efficiently lookup new STHs in the Catena log and be certain that the log server has not been coerced or compromised to equivocate about them. We believe this approach could be more resilient to attacks, as a compromised log server cannot equivocate without forking the Bitcoin blockchain. Also, because most transparency schemes publish commitments of the directory periodically, we believe they are amenable to being secured by Catena.

2) Blockchain-based Transparency: Blockchain-based transparency schemes [14], [15] are promising due to their simplicity and resilience to forks, but the overhead of downloading all blockchain data makes them unusable on many devices. Catena can decrease the overhead of these schemes from currently 90 GB [19] to around 35 MB. For example, Catena can enable thin clients running on mobile phones to efficiently audit the Bitcoin-witnessed Keybase [14] public-key directory. Currently, Keybase publishes commitments of their public-key directory in Bitcoin by creating a transaction which spends coins from a fixed address. Keybase clients recognize transactions from this address [32] and read the arbitrary data committed in the transaction. The problem with this approach is thin clients cannot securely use Bloom filtering to avoid downloading irrelevant transactions, as an adversary could selectively hide Keybase transactions and equivocate about the directory (we explain this attack in Section IV-D). Catena prevents this attack and also has the advantage of not polluting Bitcoin's unspent transaction output (UTXO) set [22].

Catena can also be used to build thin clients for blockchain-based applications such as Blockstack [15]. Currently, to benefit from Bitcoin's resilience against forks, Blockstack clients need to download the entire blockchain and compute their own *consensus hash* over all Blockstack-related operations. Blockstack clients can choose to trust someone else's consensus hash and verify public key lookups against it efficiently using Simplified Name Verification (SNV) [15], [33]. However, Blockstack clients still have to download full Bitcoin blocks to update that consensus hash or continue trusting someone else to update it. As with Keybase, Bloom filtering cannot be used securely to filter Blockstack transactions. To fix this problem, we propose using a Catena log to keep track of Blockstack operations rather than scattering them through the blockchain. In this way, thin clients can efficiently download just the Blockstack operations and quickly compute their own consensus hashes. One disadvantage of this approach, according to one of the Blockstack co-founders [34], is that it requires a secret key to manage the Catena log and would thus "centralize" the system.

3) Certificate Authorities (CAs): As an alternative to current key transparency schemes, Certificate Authorities (CAs) may choose to publish their certificates in the Bitcoin blockchain using Catena. A CA would publish a periodically-updated Merkle tree of certificates, sorted by the subject name so as to provide non-membership proofs as in [16], [23]. Subjects who are issued certificates can efficiently check that they have not been impersonated, while users, relying on subjects to monitor their own certificates, can trust that every certificate seen in the log is valid. We believe this approach is promising because of its simplicity. Specifically, users can efficiently verify that a certificate has been published in the CA's log before using it. To bootstrap the system securely, browsers can include the genesis transaction (see Section IV-A) of a Catena-enabled CA next to the CA's public key.

4) Software Transparency: Catena can prevent equivocation in *software transparency* schemes [27] and thus thwart man-in-the-middle attacks that try to inject malicious software binaries on victims' machines [27]. In fact, Bitcoin developers were concerned in the past about these kinds of attacks on Bitcoin binaries [35]. To prevent these attacks, software vendors can publish digests of new versions of their software in a Catena log. Customers can then verify any version downloaded from a vendor's website against the vendor's log. Previous work [13] already highlights the necessity of software transparency in the face of insecure software update schemes [36], [37], key loss or compromise [38] and black markets for code-signing certificates [39].

5) Tor Directory Servers: Catena can be used to prevent Tor directory servers [10] from lying about the directory of Tor relays. Equivocation attacks are particularly concerning for Tor because they enable an attacker to easily deanonymize users by pointing them towards attacker-controlled Tor relays. In fact, Tor Transparency [11] plans to address these attacks by publicly logging the Tor directory consensus. In the same spirit, we propose using Catena to increase the resilience of Tor Transparency. With Catena, directory servers can publish the consensus in a Catena log by jointly signing it using a Bitcoin multisignature [40]. Since Tor consensus is updated every hour [41], we believe it is suitable for embedding in a Catena log.

Tor presents a small challenge though, because Tor users running Catena clients need to obtain Bitcoin block headers from Catena's header relay network. Thus, we need to ensure Tor users do not leak their IP addresses to adversarial servers in this network. As a solution, Tor users can initially sync the block headers via the Bitcoin P2P network and, later on, users can query the header relay network anonymously through Tor.

6) Consensus Amongst n Servers: Catena can be used by a set of n servers to reach consensus on a log of operations, where each server manages its own private key and does not necessarily trust the other $n - 1$ servers. In this scheme, each server submits its own operation to the log by creating a Catena transaction which is spendable by all n servers (see Section IV-B for more detail on our transaction format). To disincentivize the other servers from stealing the coins, small amounts in Bitcoin can be used to fund the log along with frequent refunds (see Section IV-F). This scheme allows all servers to reach consensus on a log of operations and relies on the Bitcoin miners to decide which server's operation gets included in the log. To prevent adversarial servers from paying a higher transaction fee and monopolizing the log with their operations, the servers can agree on an upper bound on the transaction fee.

B. Bitcoin Background

In this section, we describe Bitcoin in sufficient detail to understand Catena's design. Though not necessary for understanding this paper, additional background on Bitcoin can be found in [42]–[44].

1) Overview: Bitcoin is a peer-to-peer digital currency that allows users to mint digital coins called *bitcoins* and exchange them without a trusted intermediary. Bitcoin uses a novel permissionless Byzantine consensus protocol known as *proof-of-work consensus* [18] which allows all participants to agree on a log of transactions and prevent attacks such as double spending coins. The log of transactions is called a *blockchain* and is stored and managed by a peer-to-peer (P2P) network [45]. A special set of users called *miners* run Bitcoin's proof-of-work consensus protocol, extending the blockchain with new *blocks* made up of new transactions. This process, called *mining*, is computationally difficult and secures the Bitcoin network by allowing all participants to agree on the correct log of transactions while preventing Sybil attacks [46]. To incentivize Bitcoin miners to mine, a *block reward* consisting of newly minted bitcoins is given to a miner if he mines or "finds" the next block.

2) P2P Network: Bitcoin uses a peer-to-peer (P2P) network of volunteer nodes to store the blockchain [45], listen for new transactions or new blocks, and propagate this information throughout the network. Users, such as merchants and their customers, can download the blockchain by becoming part of the P2P network and then receive or issue Bitcoin transactions. Miners, who create blocks, are also part of the P2P network where they listen for new blocks and broadcast their own blocks.

3) Blockchain: Bitcoin's public log of transactions keeps track of all transactions in the system, allowing anyone to verify that no double spends have occurred. The transaction log, or blockchain, is implemented as a hash-chain of *blocks*, as depicted in Figure 2. A Bitcoin block is made up of a small *block header* (80 bytes), which contains a hash pointer to the previous block, and a set of transactions (up to 1 MB). The transactions in the block are hashed in a Merkle tree [47] and the tree's root hash is stored inside the block header. The Merkle tree allows Bitcoin *thin clients* (see Section II-B7) to obtain efficient *membership proofs* that a transaction is part of a block.

4) Decentralized Consensus: To solve the consensus problem in the decentralized or *permissionless* setting, where participants can enter and leave the protocol as they please, Bitcoin introduces a novel Byzantine consensus protocol called *proof-of-work consensus* [18]. In this protocol, consensus on the blockchain can be reached under the assumption that 51% of the computational power amongst participants is honest. Bitcoin's consensus protocol defeats Sybil attacks [46] and keeps the Bitcoin network open or "decentralized," though it does so at a high computational cost. Specifically, participants called *miners* race to solve computationally-difficult proof-of-work puzzles derived from the previous Bitcoin block. This process is referred to as *mining*.

If a miner finds a solution, the miner can publish the next block by announcing it along with the solution[2] to everyone else over the P2P network. The puzzle difficulty is adjusted every 2016 blocks based on the inferred computational power, or *network hashrate*, of the miners so that a new block is found or "mined" on average every 10 minutes. To incentivize miners to expend computational power and produce new blocks, each mined block gives a *block reward* in bitcoins to the miner who found the solution for that block.

[2]In reality, the solution is part of the block.

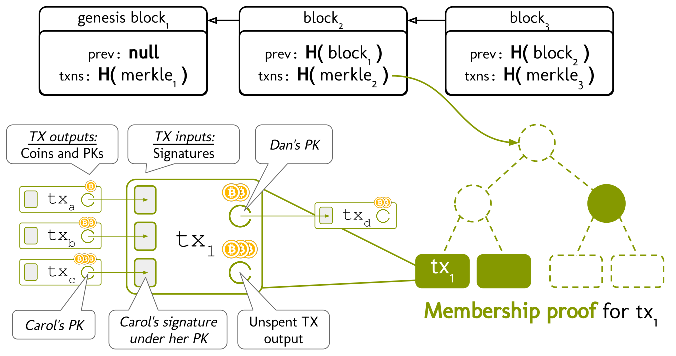

Figure 2. The Bitcoin blockchain is a hash chain of blocks. Each block has a Merkle tree of transactions. Efficient membership proofs of transactions can be constructed with respect to the Merkle root. In this example, Alice, Bob and Carol sign a transaction tx_1 transferring 6 coins: 2 coins go to Dan, 3 coins go to somebody else and 1 coin pays the transaction fee. Later on, Dan spends those coins by signing a new transaction tx_d.

When two miners find a solution at the same time, the Bitcoin blockchain forks into two chains. This is referred to as an *accidental fork*. To reach consensus in the presence of accidental forks, Bitcoin peers use the *heaviest chain rule* and select the heavier fork as the *main chain* that dictates the state of the system. The weight of a fork in Bitcoin is measured in terms of the amount of computational work expended to create that fork. If there are no difficulty changes, then the longest fork will be the heaviest fork. But across many difficulty changes, it could be that a fork with less blocks is heavier than a longer fork (though this never happens in practice).

When an accidental fork happens, both forks will have the same length and weight[3], so Bitcoin peers will adopt the fork they received first as the main chain. However, as more blocks are found by the network, eventually one of the forks becomes heavier than the other one and is accepted as the main chain by the whole network [18]. In this case, the other abandoned fork and its blocks are said the be "orphaned." In practice, accidental forks are infrequent and short: no more than one or two blocks get orphaned. To deal with accidental but also with malicious forks, most Bitcoin nodes only consider a block and its transactions *confirmed* if 6 or more blocks have been mined after it.

Proof-of-work consensus has been analyzed extensively in [18], [48]–[50].

5) Transactions: Bitcoin transactions facilitate the transfer of coins between users. A Bitcoin transaction is made up of an arbitrary number of *transaction inputs*, which specify amounts of coins to be transferred, and *transaction outputs*, which specify to whom those coins will be transferred to and in what amounts. Naturally, the number of coins locked in the outputs cannot exceed the number of coins specified in the inputs[4].

A transaction output specifies an amount of coins and their new owner, most commonly as a public key. A transaction input refers to or "spends" a previously unspent transaction output (UTXO) and contains a proof-of-ownership from that UTXO's owner which authorizes the transfer of those coins. For the purposes of this paper, we will only consider the case where outputs specify owners using public keys and inputs prove ownership using signatures.

Importantly, when assembling transactions into blocks, Bitcoin miners prevent double spends by ensuring that, across all transactions in the blockchain, for every transaction output there exists at most one transaction input that refers to or spends that output. This invariant is known as the *transaction output (TXO) invariant* and Catena leverages it to prevent forks.

Lastly, Bitcoin computes a transaction's fee as the difference in value between the coins spent in its inputs and the coins transferred by its outputs. The fee is awarded to the miner who includes that transaction in a mined block. In theory, the fee can be zero, but in practice recent contention for space in the blockchain requires users to pay transaction fees.

We show an example of a Bitcoin transaction in Figure 2. Here, tx_1 transfers coins owned by Alice, Bob and Carol to two users: Dan and somebody else. The difference between the inputs and the outputs is transferred to Bitcoin miners as a transaction fee. Alice authorizes the transfer of her coins by signing tx_1, which contains an input pointing to Alice's coins locked in the first output of tx_a. Bob and Carol do the same. Dan later spends the coins locked for him in tx_1's first output by signing a new transaction tx_d which has an input pointing to tx_1's first output.

6) Storing Data in Transactions: Bitcoin allows embedding of data in transactions through provably-unspendable `OP_RETURN` transaction outputs. An `OP_RETURN` output allows the user to specify up to 80 bytes of arbitrary data. Importantly, any coins specified in the output are forever unspendable or "burned". Catena uses `OP_RETURN` outputs to store application-specific statements in the Bitcoin blockchain (see Section IV).

7) Thin Nodes vs. Full Nodes: Bitcoin's P2P network is made up of two types of nodes: *full nodes* which download the entire blockchain and validate all the transactions and *thin nodes* which only download small 80 byte block headers and cannot fully validate transactions. Full nodes play an important role in the Bitcoin network as they validate new transactions and new blocks, relay them to other nodes and prevent double spends, helping the network reach consensus on the blockchain. However, not everyone can run a full node due to the high bandwidth, computation and space requirements. As a result, Bitcoin provides a *thin node* implementation for smaller devices with limited space and bandwidth, such as smartphones.

Thin nodes or Simple Payment Verification (SPV) nodes verify Bitcoin transactions more efficiently under a slightly stronger assumption about the Bitcoin network. Thin nodes assume that Bitcoin miners follow their incentives and create correct blocks, since otherwise miners would lose their block reward (see Section II-B4). As a result, a thin node considers

[3]Assuming the fork does not cross a difficulty recomputation point.
[4]With the exception of coinbase transactions, which mint new coins and have no inputs.

a transaction valid if it sees a correct Merkle proof of membership for that transaction in a block. Also, the more confirmations a transaction gets, the more confident a thin node can be that it is indeed valid. Importantly, thin nodes don't even verify signatures on transactions. The membership proof coupled with enough confirmations offers enough assurance that the transaction was verified by miners and is thus valid. We revisit this point later in Section IV-D when discussing Catena clients which run thin nodes but actually have to verify transaction signatures for security.

Finally, to avoid downloading unnecessary data, thin nodes use a Bitcoin feature called Bloom filtering [29]. This feature allows thin nodes to only receive transactions of interest by asking remote peers to filter out irrelevant transactions using a Bloom filter [51].

III. Model and Goals

In this section we describe our system model, our threat model and our design goals.

A. System Model

The main actors in our scheme are the log server, which appends statements to the log, Catena clients, which verify new statements and check for non-equivocation, and the header relay network (HRN) which helps scale Catena to support a large number of clients (see Figure 3).

1) Log server: A log server manages an append-only log of application-specific *statements*. The log server appends statements to the log by signing Bitcoin transactions with statement data embedded in them and broadcasting them to the Bitcoin P2P network. For now, we call these transactions *Catena transactions* and defer discussion on them until Section IV-B. In this paper, we will mostly talk about a single log server managing the log, but using Bitcoin multisignatures [40], Catena can support multiple servers who either jointly or separately append statements to the log. Also, although a log server can manage many different logs, for simplicity we restrain our discussion to a single server managing a single log.

2) Clients: Multiple clients connect to the log server and keep up with new log statements. As depicted in Figure 3, clients fetch Catena transactions from the log server and verify they have been included in the Bitcoin blockchain. This verification is done against block headers obtained from the header relay network, which we discuss next. The goal of Catena clients is to avoid being lied to about statements, a property known as non-equivocation. Specifically, a client who is shown a statement s_i wants to ensure there is no other contradictory statement s'_i in the log at position i.

3) Header Relay Network (HRN): Catena clients use a separate network to obtain Bitcoin block headers. This is simply a scalability measure we adopt due to the low connection capacity of the Bitcoin P2P network. Otherwise, if Catena clients were to connect to the Bitcoin P2P network to hear about block headers, our system would put unnecessary stress on Bitcoin's network and would not scale very well. We discuss this in more detail in Section IV-C.

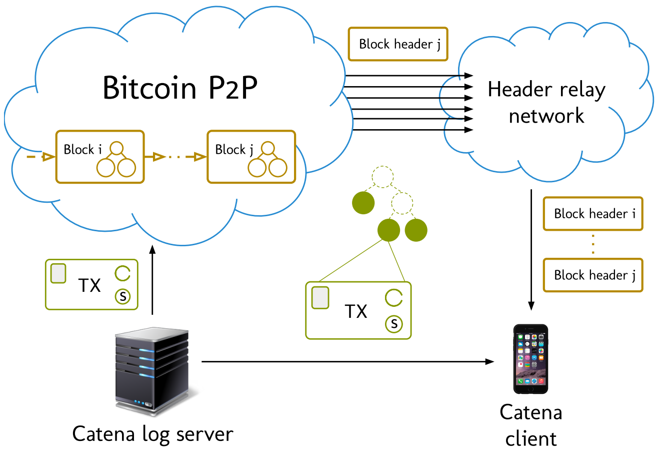

Figure 3. The log server broadcasts Catena transactions with statements embedded in them to the Bitcoin P2P network. Catena clients query the header relay network for block headers and the log server for statements and proofs that those statements were witnessed in the Bitcoin blockchain. The header relay network maintains good connectivity to the Bitcoin P2P network without depleting its connection pool.

B. Catena API

Our scheme can be succinctly described as a tuple $\langle \mathsf{CreateLog}, \mathsf{AppendStmt}, \mathsf{VerifyStmt} \rangle$ of API calls. For clarity, we prefix calls with S when they are made by the log server and with C when they are made by clients. Our API is summarized below:

$S.\mathsf{CreateLog}(d) \rightarrow (\mathsf{sk_{BTC}}, \mathsf{tx_{genesis}})$ Creates an empty log whose "public key" is the genesis transaction $\mathsf{tx_{genesis}}$ against which all future log statements can be verified. The empty log will also contain arbitrary data d, if specified by the log creator. New statements can be appended to the log by broadcasting new Bitcoin transactions signed using the secret key $\mathsf{sk_{BTC}}$ (see $S.\mathsf{AppendStmt}$ below).

$S.\mathsf{AppendStmt}(\mathsf{sk_{BTC}}, s_i) \rightarrow tx_i$ Appends the statement s_i to the log by signing a new Bitcoin transaction using $\mathsf{sk_{BTC}}$. Returns the signed Catena transaction tx_i which commits the statement s_i and spends the previous Catena transaction tx_{i-1}. If this is the first call to $S.\mathsf{AppendStmt}$, then tx_i spends the genesis transaction returned by $S.\mathsf{CreateLog}$.

$C.\mathsf{VerifyStmt}(\mathsf{tx_{genesis}}, tx_i) \rightarrow \{\mathsf{s_i}, \perp\}$ Verifies that the statement s_i in transaction tx_i is correctly appended to the log specified by $\mathsf{tx_{genesis}}$. If successful, returns the statement s_i embedded in the transaction. Otherwise, returns $\perp$ if the transaction is incorrectly signed or improperly formatted. If $tx_i = \mathsf{tx_{genesis}}$, then this call returns the arbitrary data d specified in $S.\mathsf{CreateLog}$.

To recap, a server creates a new log by calling $\mathsf{CreateLog}$ and appends statements to this log by signing and broadcasting Catena transactions using $\mathsf{AppendStmt}$. Clients keep up with the log by listening on the header relay network for new block headers, fetching new transactions tx_i from the log server and verifying them using $\mathsf{VerifyStmt}$.

C. Threat Model

1) Adversarial Log Server: We assume the Catena log server is compromised or coerced and wants to equivocate about statements. We assume Catena clients can correctly obtain the log's genesis transaction which acts as the log's "public key" (see Section IV-A). We note that both Catena and previous work [8], [13], [16], [23], [25], [26] all rely on some sort of initial public-key distribution. However, unlike previous work, Catena can make guarantees about non-equivocation once a client has the "public key" or genesis transaction. Specifically, Catena prevents equivocation once the client has obtained the genesis transaction identifying the log. It's important to understand that, similar to how a signature can only be verified with respect to a public key, equivocation can only be prevented with respect to a log identified by some kind of information, in this case, the genesis transaction.

We stress that Catena's goal is to prevent equivocation given a log's genesis transaction and orthogonal techniques can be used for distributing the genesis transaction. For instance, the log's genesis transaction can be shipped with the application software that audits that log, similar to how browsers are shipped with public keys of Certificate Authorities (CAs). In fact, because the Bitcoin blockchain effectively timestamps transactions, we argue it might be easier for end-users to verify the genesis transaction if they have a rough idea about when the log was started. This is because users can inspect the Bitcoin blockchain efficiently via Bloom filtering [29] and make sure no other genesis transaction for the log exists around its creation time.

2) Proof-of-Work Consensus: Like all previous Bitcoin-based work [14], [15], [30], [52]–[57], we assume that adversaries cannot break Bitcoin's proof-of-work consensus and fork the blockchain. Specifically, we assume that a Catena transaction is immutable once it has been confirmed by a sufficient number of blocks, as configured by Catena clients individually (we recommend at least 6 blocks). We believe it is reasonable to assume that long malicious forks are unlikely to occur due to the computational difficulty and financial burden of such an attack. We also assume the Catena log server cannot collude with large Bitcoin miners, who are not likely to benefit financially from a forking attack. We discuss accidental forks, which have occurred in the past [58], [59], and malicious forks, which are not known to have occurred, in more detail in Section V.

3) SPV Assumption: To enable Catena logs to be verified efficiently via Simplified Payment Verification (SPV), we have to assume miners verify their own blocks and the blocks of other miners before mining, otherwise they could accidentally fork the blockchain. Fortunately, Bitcoin miners have a strong incentive to verify blocks, as miners would lose the block reward and TX fees if they create invalid blocks. However, recent work [60] shows that if block verification is expensive, then miners have an incentive to skip it. This route was indeed taken by at least two large Bitcoin miners in 2015. However, these miners were actually acting in an irrational manner and, as a result, suffered a significant financial loss [59]. We discuss this in more detail in Section V-C.

4) Bitcoin's P2P Network: Like Bitcoin, we have to assume its P2P network is reliable and broadcasts Bitcoin blocks timely, or else its proof-of-work consensus would be easily subverted [18], [61]. While so-called "eclipse attacks" [61] on the P2P network can lead to double spending attacks, mitigations for this attack have already been deployed in Bitcoin's P2P code. Moreover, networks such as the Bitcoin Fast Relay Network (FRN) [62], Falcon [63] or FIBRE [64] are already deployed to speed up block propagation and make the network more resilient to partitioning attacks.

5) Header Relay Network: We also trust Catena's header relay network to serve Catena clients with the latest Bitcoin block headers. Note that nodes in this network cannot equivocate about block headers unless a fork happens, in which case our proof-of-work consensus assumption would be broken anyway. In that case, Catena cannot prevent equivocation but can make it detectable later when the fork is resolved. Thus, we only trust the header relay network for availability and freshness. We can minimize trust in this network but only at the cost of putting some stress on Bitcoin's P2P network.

D. Goals

Succinctly, our goal is to prevent equivocation and to do so in an efficiently verifiable manner. Efficient verification is crucial as it enables each user to audit individually, minimizing trust in our applications such as public-key directories.

1) Non-equivocation: A log server should have a hard time equivocating about log statements. Catena makes equivocation in the log as hard as forking the Bitcoin blockchain, which we believe to be a reasonable amount of protection for many applications, including public-key directories. If our assumptions are broken and the Bitcoin blockchain forks, Catena cannot prevent equivocation but still makes equivocation detectable once the forks are resolved, similar to previous gossip-based approaches [9], [16], [23], [24].

It's important to understand what non-equivocation actually provides. Non-equivocation does not prevent the adversarial log server from issuing incorrect statements that break semantics at the application layer. Instead, non-equivocation simply guarantees that all clients will see all issued statements, including incorrect ones. This allows clients to detect attacks at the application layer. For example, a compromised public-key directory server could issue a fake statement attesting to a fake directory d that contains fake public keys for both client A and B. Non-equivocation guarantees that both A and B will see the same fake directory d and can quickly detect (at the public-key directory application layer) their own fake public keys in the directory. As a result, A and B can stop using the directory and whistleblow as in [8], [16], [23].

2) Publicly Verifiable: Given a log's public key as its genesis transaction $\mathsf{tx}_{\mathsf{genesis}}$, anyone can verify the full history of statements in that log. Specifically, a client can obtain all statements $\langle s_1, s_2, \ldots, s_n \rangle$ in the log and verify them with respect to $\mathsf{tx}_{\mathsf{genesis}}$. Verification here means that a statement is part of the log at some position i and no other inconsistent statement at position i exists (i.e., non-equivocation). In particular, for any statement s_i, the log server gives the client a publicly verifiable proof p with respect to the log's $\mathsf{tx}_{\mathsf{genesis}}$ which proves that s_i is indeed the only statement in the log at position i.

3) Efficiently Verifiable: Catena clients should be able to audit logs efficiently without downloading the entire Bitcoin blockchain. Recent blockchain-based transparency work [14]–[16] is inefficient, requiring clients to run full Bitcoin nodes that download the entire blockchain or risk being forked (see Section II-A2). This high space and bandwidth requirements raises the barrier to entry for log auditors who might have to outsource auditing or trust the log blindly. In contrast, the barrier for Catena clients is very low. A client only downloads small 80-byte block headers for each Bitcoin block and 600-byte Merkle membership proofs for each issued statement.

IV. Catena Design

At a high-level, Catena makes equivocation about a log statement as hard as double spending a Bitcoin transaction output. The key idea behind Catena is to embed statements in Bitcoin transactions and have each transaction spend the previous one. This is a simple but powerful idea because *it forces the log server to double spend a transaction output if it wants to equivocate*, which is notoriously difficult in Bitcoin. Thus, Catena can offer a strong guarantee to clients that they have not been lied to.

Catena operates very simply, as illustrated in Figure 1. The Catena log server creates a log by issuing an initial transaction called the *genesis transaction*. The server issues the first statement in the log by creating a new transaction which spends the genesis transaction and commits that first statement via an `OP_RETURN` transaction output (see Section II-B6). Finally, the server can append a new statement to the log by creating a new transaction which spends the previously-created transaction and commits the statement as before.

Catena clients first obtain the log's genesis transaction, which can be shipped with the higher-level application that Catena secures, such as a public-key directory client. Then, clients obtain and verify all Bitcoin block headers from the header relay network (discussed in Section IV-C). Finally, clients can ask the Catena log server for the statements and verify them against the genesis transaction and the Bitcoin block headers. Importantly, because Catena transactions are chained and Bitcoin prevents double spends, clients are assured that no inconsistent statements have been issued by the server.

Catena's overhead is small. For each statement, the server will send over a 235-byte[5] Catena transaction and a 350-byte Merkle path proving that the statement is part of the log. That amounts to around 600 bytes per statement plus the overhead of downloading all block headers (currently 35 MB), making Catena very cheap in terms of bandwidth. Our design is simple

[5]Assuming statements are 32-byte hashes or signatures.

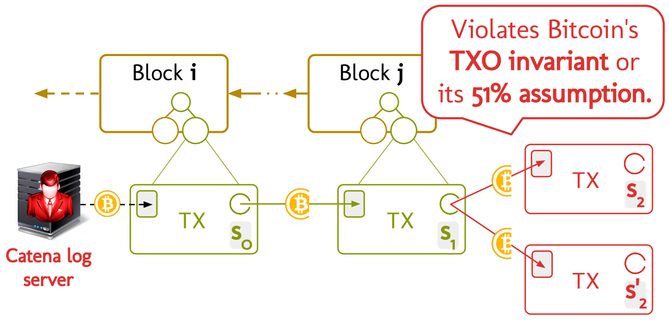

Figure 4. Equivocating in a Catena log is as hard as double spending in Bitcoin which requires forking the blockchain. This is because Catena's design requires a new Catena transaction to spend the previous one which linearizes the history of statements embedded in those transactions, thereby preventing equivocation.

and prevents equivocation as long as the Bitcoin blockchain does not fork.

A. Genesis Transaction

Catena logs are identified by a genesis transaction. This is the first transaction created by the log server when it starts the log. The genesis transaction effectively acts as the log's "public key": once a client has the log's genesis transaction, that client can verify log updates against it and prevent equivocation.

Note that some kind of information about the log is needed for any system which claims to prevent equivocation, whether it's a Merkle root hash as in [20] or a genesis block as in Bitcoin [17]. Without such information, clients would have to resort to accepting any statement without being able to check whether it belongs to the log, making equivocation attacks easy. Thus, Catena uses the genesis transaction as its log-identifying information against which clients can verify new statements.

B. Catena Transactions

A Catena log is just a chain of specially-crafted Bitcoin transactions, which we call *Catena transactions* (see Figure 1). Our transaction format is simple. First, a Catena transaction has one input, which spends the previous Catena transaction in the chain. Second, a Catena transaction has two outputs. The first output is an unspendable `OP_RETURN` output which commits the log statement and the second output is a *continuation output* which is spent by the next Catena transaction's input. The genesis transaction also has the same Catena transaction format.

Our transaction format leverages the fact that Bitcoin miners prevent double spends which, in turn, allows us to prevent equivocation about statements. We illustrate this in Figure 4. The key idea is that a Catena transaction has a single spendable output, which means Bitcoin miners will ensure *only a single future transaction will spend that output*. Thus, a Catena transaction can only be followed by another *unique* Catena transaction, which allows us to create a linear history of statements that all Catena clients agree on.

Catena transactions just transfer coins from the Catena log server back to itself, committing log statements and paying fees to Bitcoin miners in the process. Recall from Section II-B5 that a transaction output specifies a coin amount and a public key that "locks" those coins (i.e., is authorized to spend them later). In Catena, all transaction outputs are locked by the same key called the *statement key*, which is managed by the log server. This key signs all Catena transactions, including the statements embedded in them, authorizing the transfer of coins back to the server. Catena clients can easily obtain the statement key from the genesis transaction since it is specified in its continuation output. The server can choose to change the statement key in future transactions and clients can easily pick up the new key, but for simplicity we assume it remains the same across all Catena transaction.

As mentioned before, the log server has to pay fees to Bitcoin miners to get its transactions included in the blockchain. Due to recent contention for space in Bitcoin blocks, zero-fee transactions are not likely to be included in the blockchain. Thus, a Catena log server needs access to some initial Bitcoin funds and will have to occasionally "refund" its Catena log when it runs out of funds. We describe how refunding works in Section IV-F and we analyze the server's cost per Catena statement in Section VI-C1.

C. Header Relay Network

We want to avoid putting stress on Bitcoin's P2P network, which has a limited connection capacity that would be quickly depleted by Catena clients. There are currently around 5500 full Bitcoin nodes, each by default capable of handling up to 117 incoming connections [65], [66]. This means Bitcoin's P2P network currently supports at most $5500 \times 117 = 643,500$ incoming connections at a single point in time, some of which are already used up by Bitcoin thin clients for user wallets. Importantly, these connections need to be long-lived so as to allow thin clients to connect to a diverse set of Bitcoin peers. Thus, if each Catena client maintains 8 outgoing connections, this implies Catena cannot scale beyond tens of thousands of clients without putting a significant stress on the Bitcoin network.

To address this problem, we propose using a header relay network (HRN) that is well connected to the Bitcoin P2P network and can serve block headers to hundreds of thousands of Catena clients. A header relay network can act as an extension of the Bitcoin P2P network without introducing a weak link in our design or depleting Bitcoin's small connection pool. As described in our threat model (see Section III-C), this network is trusted only for availability and freshness: an adversary still has to fork the Bitcoin network to equivocate in a Catena log.

For an initial small deployment of Catena, Bitcoin's P2P network can serve as the header relay network, but as more Catena clients join, the system has to transition to specialized header relay nodes. These nodes would be part of the Bitcoin P2P network and contribute to its health but would also provide an interface to Catena clients for obtaining block headers fast. For example, current blockchain explorers [67]–[70] can already act as header relay nodes as they are well connected to the Bitcoin network and already provide public APIs for fetching block headers.

For some applications, a header relay network can be bootstrapped as a P2P network amongst Catena clients themselves. Catena clients in this network can occasionally fetch block headers from the Bitcoin P2P network and then distribute them amongst themselves. Each Catena client can query the Bitcoin P2P network with probability inversely proportional in the size of the header relay network, so as to avoid stressing Bitcoin P2P nodes. The header relay network size can be estimated by each Catena client using known techniques [71]. Sybil attacks [46] can be addressed by requiring Catena clients joining the network to "burn" bitcoins in a provable manner and tie their identities to those coins.

D. Auditing a Catena Log

To audit a log, clients download the Catena transaction chain and verify that transactions are signed and chained correctly using the statement key. Clients first download and verify block headers from the header relay network and then download and verify Catena transactions and their Merkle proofs from the log server. The log server can serve the transactions and Merkle proofs more efficiently than using Bloom filtering on the Bitcoin P2P nodes (see Section II-B7) which creates significant disk activity for Bitcoin full nodes. Finally, auditing is cheap for Catena clients as they only download small transactions and Merkle proofs (600 bytes) and not full Bitcoin blocks (1 MB).

When a client verifies a new Catena transaction tx_i, it checks that:

- tx_i is in the correct Catena format
- tx_i is correctly included in a Bitcoin block with a Merkle membership proof.
- The first input of tx_i spends the continuation output of the previous Catena transaction tx_{i-1}.
- tx_i is signed correctly with the statement key of the log.
- tx_i has a sufficient number of confirmations (we recommend at least 6).

It's important to understand that without clients verifying transaction signatures, a malicious log server can lie about statements in the log. For example, consider two Catena clients c_1 and c_2 which correctly obtained the genesis transaction $\mathsf{tx}_{\mathsf{genesis}}$ of the log but do not verify Catena transaction signatures. In this attack, the malicious log server issues two Catena transactions that commit two different statements s_1 and s'_1 respectively but, importantly, *do not spend* the genesis transaction. Instead they spend some other transactions and get included in the blockchain. The attack is straightforward: the log server simply shows client c_1 the transaction for s_1 but hides the one for s'_1. Similarly, client c_2 is shown the transaction for s'_1 and not the one for s_1. Thus, without verifying signatures, clients can be easily lied to as they have no way of checking that s_1 and s'_1 are truly part of the log.

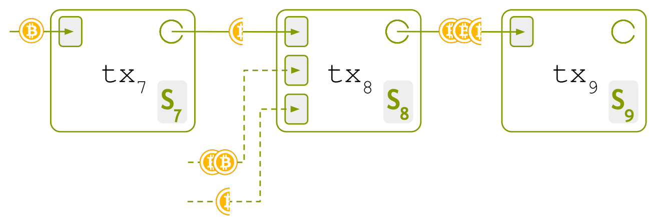

Figure 5. Refunding a Catena chain is done by allowing the next transaction in the chain to have additional inputs which lock extra coins in that transaction's continuation output. In this example, Catena transactions pay .5 BTC as a fee. Thus, to ensure tx_8 does not run out of coins, we refund it using extra inputs.

E. Blockchain Reorganizations

Like Bitcoin, Catena also needs to deal with small day-to-day blockchain reorganizations, such as when two miners both find a block and the blockchain forks for a short time (see Section II-B4). These small forks are automatically resolved by the Bitcoin network: as more blocks are found, eventually one of the forks overtakes the other one and becomes the main chain [18]. To be certain payments are not reversed by these small reorganizations, Bitcoin merchants only consider a block and its transactions confirmed if 6 or more blocks are built on top of it.

Catena also relies on confirmations before accepting a statement as final. Like Bitcoin, Catena allows clients to set their own application-specific number of required confirmations before accepting a statement. As a result, Catena makes a trade off between resilience to forks and latency of accepting statements. Additionally, as a security measure against longer accidental forks, Catena clients remember recently-issued statements. This way, if a statement is withdrawn due to a reorganization, Catena clients can ensure the reissued statement matches the previously seen one.

F. Paying for a Catena Log

A Catena log server must pay Bitcoin transaction fees to start a log and append statements to it. Initially, the Catena log server must obtain some bitcoins (BTC), perhaps from a Bitcoin exchange [72]. Then, the server can issue the log's genesis transaction and pay for its fee. The server will lock some coins in the genesis transaction's continuation output which can "fund" future log transactions. To issue the first statement, the server signs a new Catena transaction with the statement key. This transaction commits the statement (via an `OP_RETURN` output), transfers the genesis transaction's coins back to the log server and leaves a small fee for the miners. As before, the remaining coins will be locked in this new transaction's continuation output. The server repeats this process for every new statement, spending the coins locked in the previous Catena transaction, until it runs out of funds.

To "refund" the chain, we use a Catena transaction with additional inputs which locks extra coins in that transaction's continuation output (see Figure 5). Importantly, these inputs can only be used to add extra funds and cannot be used to maliciously join two different logs. This is because we restrict Catena transactions to only use their first input to spend a previous Catena transaction. Thus, clients can easily detect if a Catena transaction tries to point to two distinct previous Catena transactions by using additional inputs.

We analyze the costs of running a Catena log in terms of transaction fees in Section VI-C1.

V. Attacks

In this section we describe attacks on Bitcoin that can translate into attacks on Catena and explain what Catena clients can do to protect themselves. We touch upon attacks on Bitcoin's P2P network, accidental forks, as well as malicious forks. We also explain what can and cannot happen when a log server's secret key is stolen.

A. Stolen Key Attack

An attacker might compromise the Catena log server and steal its statement key. In this case, an attacker can issue his own statements, but he cannot fork the log and equivocate about statements. That is, all clients will see all attacker-issued statements and can check their correctness at the application layer. The attacker can also steal the server's Bitcoin funds. However, we stress that Catena's main goal is to prevent equivocation in the face of stolen key attacks and orthogonal techniques can be used to secure the Bitcoin wallet of Catena servers.

Once the attacker has the statement key, he could also abruptly "end" the log by issuing a transaction that is not in the correct Catena format. To recover from such an attack, the Catena log server has to abandon that log and start a new one with a new genesis transaction. In this sense, Catena performs no worse than previous systems which would also have to advertise a new public key to log clients if all log server secrets were compromised. However, unlike previous work, Catena provides non-equivocation in the face of a stolen key attack.

B. Freshness Attacks

The log server could hide away transactions from Catena clients. As a result, Catena clients would lose freshness and not be aware of the newest issued statements. However, as discussed in Section IV-D, Catena clients can never be lied to about statements as that would require double spending a transaction in Bitcoin. Similarly, the header relay network can hide block headers from Catena clients. However, hiding headers too only sacrifices freshness and does not lead to equivocation attacks unless a fork in the Bitcoin blockchain happens. To prevent freshness attacks, clients can query the Bitcoin blockchain directly and efficiently via Bloom filtering [29]. However, this defense comes at the cost of putting some stress on the Bitcoin P2P network.

C. Accidental Forks

Accidental forks in the Bitcoin blockchain pose a threat to Catena clients as adversaries can double spend Catena transactions across forks and equivocate. In the past, Bitcoin has had three major accidental forks. Two of them, in August 2010 and March 2013, were due to bugs in the `bitcoind` daemon [58], [73] and one of them, in July 2015, was caused by at least one irrational miner [59], which we expand on below. All of these forks orphaned a significant number of blocks, enough to unconfirm previously confirmed transactions. Moreover, during the March 2013 fork [58], an honest-but-curious user attempted a double spend attack on a Bitcoin exchange which succeeded. However, the attacker quickly returned the funds to the exchange [74].

We stress that accidental forks have been rare and are thus outside of our threat model. Also, an adversary cannot exploit accidental forks without also compromising the header relay network or the Bitcoin P2P network. Clients can be made aware about forks via the header relay network and refuse accepting statements until forks are resolved, which gives them an extra line of defense. As a last line of defense, Catena clients can wait for additional confirmations to protect themselves against accidental forks.

1) "SPV" Mining: The July 2015 fork was caused by at least one irrational miner who mined for over an hour on top of an unverified chain [75]. This strategy of mining before verifying is called *SPV mining*[6] and it hurts the Bitcoin network because it risks extending an invalid block and fork the blockchain. SPV mining is used by some rational miners as a way to start mining quicker and lower their rate of orphaned blocks [60]. However, when performed without a timeout, this strategy is actually irrational as it can leave miners mining on an invalid fork indefinitely. As we explain below, this is what happened in July 2015.

Instead of waiting to hear about a solved block on the P2P network, SPV miners obtain a solved block hash directly from other mining pools via their Stratum mining API [76]. Then, they start mining on top of that hash, assuming its corresponding block is correct and expecting to eventually receive the full block via the P2P network. Unfortunately, if the block is invalid, the P2P network will not waste bandwidth broadcasting it. Thus, SPV miners will never hear about an invalid block, which is why they need to time out after a while and switch to mining on the correct chain. Otherwise, SPV miners could be left mining on top of an invalid chain forever. This is exactly what happened in July 2015, when several miners did not implement timeout logic and went on to mine several invalid blocks, losing over $50,000 in profits [59].

SPV mining remains a concern for the larger Bitcoin network. However, future Bitcoin improvements should further decrease the orphan rate and steer miners away from this unhealthy mining strategy. These could be improvements in block propagation delay, block verification speed as well as new fast block relay networks such as Falcon [63] and FIBRE [64].

[6]SPV normally stands for Simple Payment Verification as discussed in Section II-B7, but here it is used to indicate that miners are not verifying the block they are mining on.

D. Network Partitioning

Eclipse attacks [61] and Sybil attacks [46] on the Bitcoin P2P network can partition nodes and lead to double spending attacks on the Bitcoin network *without the need for adversarial mining*. This in turn can lead to equivocation attacks on Catena logs. While countermeasures against eclipse attacks have already been adopted by the Bitcoin developers, full-fledged Sybil attacks on the P2P network would constitute a break of Bitcoin itself and, if practical, could be a concern for both Bitcoin and Catena. However, these attacks have not been observed yet in the wild and it is not clear that they could remain undetected for long. For instance, mining pool operators would quickly notice the fork by an increase in their fraction of mined blocks. We also stress that in applications such as key transparency schemes one fork is not sufficient. The adversary would need to maintain many forks in order to be able to impersonate multiple users, which further complicates his task. We plan on investigating to what extent a Sybil attack can partition the Bitcoin P2P network in future work.

E. Adversarial Mining Attacks

A sufficiently powerful adversary can mine his own side chain and fork the Bitcoin blockchain, enabling him to double spend transactions across the two forks. Unfortunately, thin clients are more vulnerable than full nodes to a generalized "Vector76" attack where the attacker mines a k-block long side chain that is at least one block longer than the main chain [77]. The side chain contains a transaction tx with k confirmations which the attacker will later replace with another transaction tx' double-spending the same output(s) as tx. When the attacker successfully mines the side chain, he shows the side chain only to the victim, who will accept tx. Then, the attacker will cease to mine, will issue tx' to the main chain, and let the main chain win the race, confirming tx' and unconfirming tx.

Full nodes are more resilient to this attack because they can relay the attacker's side chain to the rest of the network, while thin clients cannot. Thus, with full nodes, the attacker's side chain could be adopted by the network which would prevent the double spend. However, we stress that with proper timing, the attacker can also trick full nodes if he is able to propagate his side chain to the victim at the same time as the same-length main chain is propagating to the rest of the network [77].

Similar to previous work [14], [15], [30], [52]–[57], we exclude adversaries who can mine a sufficiently long side chain from our threat model because they are extremely powerful and so far they have not been observed in practice. These adversaries can break not only thin nodes but also full nodes with proper attack timing. The main countermeasure against these attacks is to simply wait for more confirmations, which makes the attacker's job more difficult. Another countermeasure, is for Catena clients to accept a block header only after hearing about it from multiple sources, so as to ensure the attacker's side chain is seen by the whole Bitcoin P2P network.

VI. Prototype and Evaluation

We implemented a Catena prototype in Java using the bitcoinj [28] library in 3000 lines of code[7]. Our code is open source and available on GitHub:

`https://github.com/preventinglieswithbitcoin/catena-java`

Our prototype implements a Catena log server and a Catena client, both operating as thin nodes on the Bitcoin network. In this first implementation, Catena clients use only the Bitcoin P2P network to fetch both block headers and Catena transactions with their associated Merkle proofs. In a future, more scalable implementation, we plan on fetching transactions and Merkle proofs from the log server and on using a header relay network for downloading block headers.

A. Catena Log Server

The log server manages the statement key used to sign new Catena statements (see Section IV-B) and a set of *funding keys* used to refund a Catena log (see Section IV-F). The server provides an $\mathsf{appendStatement(s)}$ API for issuing a statement s, which abstracts the Bitcoin layer away from applications. Though currently not implemented, the server should refund the chain automatically assuming there are sufficient funds controlled by funding keys.

B. Catena Log Client

The client connects to the Bitcoin P2P network and sets a Bloom filter [51] on all its connections to filter out irrelevant transactions. This way, Catena clients only receive server-issued Catena transactions (see Section II-B7) and save orders of magnitude in bandwidth. Currently, a Catena client only has to download approximately 35 MB worth of block headers and around 600 bytes for each statement and its Merkle proof. Thus, if one statement is issued per block (i.e., every 10 minutes), Catena clients have to download less than 100 KB per day. While additional bandwidth will be consumed by small blockchain reorganizations (see Section IV-E), this amount should be negligible.

Catena clients expose an $\mathsf{onStatementAppended(s)}$ API which notifies the higher level application of newly issued statements s which have sufficient confirmations. Applications are notified about statements in the order they were issued, making it easy to verify each statement for application-specific invariants (we discuss this in more detail in Section VII-3). If the Catena log is caught equivocating, the Catena client notifies the application via an $\mathsf{onEquivocation(s, s')}$ API which includes signatures on the two inconsistent statements s and s' and thus offers a publicly-verifiable non-repudiable proof of equivocation.

Certain applications might want to be made aware about the stability of Bitcoin's consensus. For this, we provide an $\mathsf{onReorganize()}$ API which notifies applications about blockchain reorganizations with information about forks and the number of orphaned blocks. Applications can use this information to infer whether the Bitcoin network is under attack, but we leave this to future work.

If bigger accidental or malicious forks should occur, they might unconfirm previously-confirmed Catena transactions. Even though such events are outside of our threat model, Catena still notifies applications about statements that were unconfirmed via an $\mathsf{onStatementWithdrawn(s)}$ API so they can decide how to proceed.

C. Costs and Overheads

In this subsection, we discuss the financial cost of running a Catena server, the overheads involved for clients and servers, and Catena's scalability.

1) Transaction Fees: Transaction fees in Bitcoin vary with contention for space in the blockchain (see Figure 6) and so far have not been prohibitive for Bitcoin users. For instance, Bitcoin transactions currently pay a fee of 70 satoshis per byte to get included in the blockchain within the next block [79]. For a 235-byte Catena transaction that commits a statement consisting of a 256-bit SHA256 hash, the fee would be 16,450 satoshis or 12 cents per statement[8]. If a statement is issued every 10 minutes, the cost per day would be less than 17.5 USD, which we believe is reasonable. For example, this cost is not much higher than Keybase's cost [14], which commits a Merkle root less often (every 6 hours) in the Bitcoin blockchain, paying a smaller fee of 10,000 satoshis or 7 cents per transaction [80].

2) Overheads: Catena's CPU overhead is insignificant. Catena servers can issue at most one statement per Bitcoin block, so they only have to perform one signature every 10 minutes. Similarly, Catena clients only verify a transaction every 10 minutes for each log they audit, which adds virtually no overhead. Finally, verifying the proof-of-work in block headers adds insignificant overhead.

Catena requires a small, constant amount of storage to recompute the Bitcoin difficulty and handle blockchain reorganizations. To recompute the difficulty every 2016 blocks, Catena clients and servers need to store the last 2016 block headers of the blockchain, which are 80 bytes each. To prevent lies about withdrawn statements during blockchain reorganizations (see Section IV-E), Catena clients also remember the past 100 statements issued by the server, which cannot exceed 80 bytes each due to the `OP_RETURN` payload limit (see Section II-B6). Here we assume that no Bitcoin fork, whether accidental or malicious, will be longer than 100 blocks. Thus, Catena's storage cost for both clients and servers is smaller than 200 KB.

Catena demands a small amount of bandwidth from clients and a larger amount from servers who have to serve statements to clients. First, servers and clients pay an initial cost to sync all the blockchain headers (currently 35 MB). Servers and clients need to download all the headers so as to ensure the chain is sufficiently "heavy" and is thus the correct chain (see Section II-B4). Once this is done, Catena clients need

[7]Measured using the `sloccount` tool.

[8]Currently, 1 BTC = $706.54.

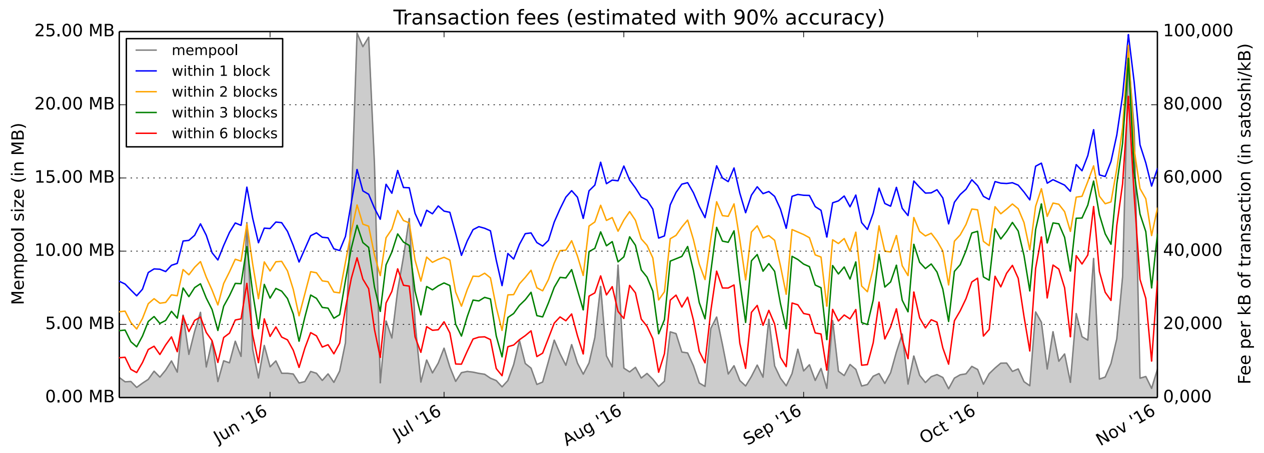

Figure 6. The transaction fees in this graph are estimated with 90% accuracy using Feesim [78]. This graph shows that fees tend to increase with contention for space in the Bitcoin "mempool" of unconfirmed transactions and that transactions with higher fees get included in the blockchain faster. The fee is displayed in "satoshis" per kilobyte, where 1 satoshi = 0.00000001 BTC = 10^{-8} BTC and 1 kilobyte = 10^3 bytes.

to sporadically connect to the header relay network to check for new block headers and connect to the log server to fetch new statements. The required bandwidth for clients is less than 1 KB every 10 minutes: 600 bytes for statements and Merkle proofs and 80 bytes for each block header requested. In contrast, the server will need more bandwidth to serve statements to all of its Catena clients.

3) Scalability: We believe Catena can scale easily if the header relay network distributes block headers and the Catena log server distributes statements and proofs. However, our current implementation based on Bitcoin's P2P network will not scale beyond tens of thousands of Catena clients without putting significant stress on Bitcoin. As discussed in Section IV-C, there simply aren't enough connections available in the Bitcoin network to support a large number of Catena clients. In addition, in our current implementation, Bloom filtering is somewhat expensive to perform for Bitcoin full nodes as they have to read blocks from disk and pass them through the filter. We stress that these are current, surmountable limitations of Bitcoin that all *thin* blockchain-based applications need to deal with, not just Catena.

D. Preventing Lies in CONIKS

To demonstrate Catena's applicability to key transparency schemes, we modified CONIKS [16] to publish directory commitments in a Catena log so as to prevent a malicious provider from equivocating about its public-key directory. Our modified CONIKS is as hard to fork as Bitcoin, which we believe makes CONIKS more resilient to attacks. Our changes to CONIKS are minimal, consisting of 66 new lines of code for the CONIKS server and 89 new lines of code for the CONIKS test client[9].

A typical CONIKS provider advertises the root hash of a prefix Merkle tree periodically to CONIKS clients. This root hash is signed and is referred to as a Signed Tree Root (STR). To prevent impersonation, clients have to gossip STRs amongst themselves or with different providers. Our modification of CONIKS removes the need for gossiping by witnessing STRs in the Bitcoin blockchain using a Catena log. This allows all CONIKS clients to agree on the same history of STRs. We lowered the frequency at which providers publish STRs from 1 minute to 10 minutes to coincide with the frequency of Bitcoin blocks. We also modified the CONIKS test client to listen for Bitcoin-witnessed STRs. However, because the provided test client is not fully implemented to keep track of STRs, more changes to CONIKS, not Catena, are needed to actually prevent equivocation.

Catena does not change CONIKS's public-key distribution assumptions. CONIKS assumes that clients have a way of obtaining the public keys of providers. Similarly, our Bitcoin-witnessed CONIKS assumes that clients have a way of obtaining the "public keys" for the Catena logs of providers. Specifically, our "public key" is the log's genesis transaction (see Section IV-A). We commit the old public key of the provider in the auxiliary data of the genesis transaction (see Section III-B). CONIKS needs this actual public key to sign server replies to clients.

VII. Discussion and Future Work

1) Building Catena on Top of Bitcoin: Our main reason for designing Catena on top of Bitcoin is the resilience of its proof-of-work consensus [18], which we leverage to prevent equivocation in Catena logs. Another reason is that Bitcoin is already deployed, which makes Catena-enabled applications easy to deploy, without having to wait for trustworthy parties to come into existence. We note that Catena could also be built on top of other blockchains such as Ethereum [81], but we believe Bitcoin's security currently outmatches the security of all other blockchains.

In particular, we avoided Ethereum for a few reasons. First, thin client support in Ethereum is currently not implemented [82], which breaks our efficiency goals (see Section III-D3).

[9]We changed Java source files, project files and configuration files.

Second, Ethereum plans on transitioning to a proof-of-stake consensus algorithm [83] which would change the trust assumptions behind thin nodes. Third, we believe Bitcoin is a more mature ecosystem to base applications on, given all current blockchain-based apps built on top of it [14], [15], [84]–[86].

2) Censorship: Catena's liveness depends on the censorship-resistance of the Bitcoin network. Malicious miners can censor Catena transactions and exclude them from the Bitcoin blockchain which reduces the liveness of a Catena log. We stress however, that Bitcoin's decentralized consensus does provide some degree of censorship-resistance by allowing *any* honest miner to join the protocol, eventually resulting in an honest, non-censoring, majority. We also stress that censorship attacks have not been observed in practice and, if performed, could affect miner profits by turning honest miners against censoring ones. We leave a more careful analysis of Bitcoin's censorship-resistance to future work.

3) Historical Consistency: Catena is application-agnostic and does not guarantee application-specific *internal consistency* of statements [20], which needs to be checked at the application layer. Instead, Catena only guarantees *historical consistency* of statements as in [20], enabling applications to later check the correct semantics of statements. As an example, Catena ensures that all clients of a key transparency scheme such as Certificate Transparency (CT) [8] see the same history of signed tree heads (STHs). However, applications still have to check the internal consistency of the STHs to detect impersonation. For instance, Bob's client will want to make sure that across all STHs, his public key has not been changed maliciously, and thus he hasn't been impersonated.

It is important to understand that without historical consistency, any guarantees of internal consistency are meaningless. This is exactly why we designed Catena. For instance, a malicious CT log server [8] can equivocate, giving Alice an STH with her real public key and a fake public key for Bob, while giving Bob a different STH with his real public key and a fake public key for Alice. Alice and Bob both verify their own STHs as being internally consistent and believe they were not impersonated. However, because Alice and Bob have no historical consistency, they are looking at different STHs which means the internal consistency guarantees they have are essentially useless. In this case, Alice and Bob are being impersonated even though internal consistency tells them they are not. Thus, without historical consistency, an adversary can equivocate and present inconsistent histories to different users such that each user's history verifies locally as internally consistent, even though it would not verify globally with respect to all the other "hidden" histories.

VIII. Related Work

Tamper-evident logging [20] allows log auditors to challenge a log and ensure its correct behavior. A *history tree* is used to store events in the log, check their membership and prove that a new version of the log is consistent with a past version (i.e., no past events have been removed or modified). Unfortunately, tamper-evident logging does not fully address equivocation attacks, and assumes auditors can gossip to detect forks. Catena offers the same semantics as tamper-evident logging (i.e., membership proofs, consistency proofs) but also prevents equivocation. However, because the Bitcoin blockchain is implemented as a hash-chain, Catena's membership and consistency proofs are linear, not logarithmic, in the log's size. In practice, a Catena log can commit root hashes of a history tree and prevent equivocation about the tree, while preserving logarithmic membership with respect to a root hash and logarithmic consistency proofs between two consecutive root hashes.

Keybase [14] and Blockstack [15], [30] use the Bitcoin blockchain to prevent equivocation but do so inefficiently, requiring clients to run a full Bitcoin node. Catena instead uses `OP_RETURN` transaction chains to allow thin clients to efficiently check for non-equivocation. CommitCoin [57] uses the Bitcoin blockchain to "timestamp" commitments and prove they were made at a certain time. Unlike Catena though, CommitCoin does not prevent an adversary from committing different data twice in the blockchain and equivocating to thin clients about the committed data.

CoSi [13] prevents equivocation by requiring statements to be signed by a threshold number of verifiers known as "witnesses." Depending on the application, these witnesses could also check the internal consistency of statements (see Section VII-3). Both CoSi and Catena make assumptions about connectivity of participants. CoSi requires a relatively well connected set of witnesses for its tree broadcast while Catena requires the Bitcoin P2P network to not be easily partitioned. One drawback of CoSi is that it requires an admission control process for witnesses in order to prevent Sybil attacks [46]. As a result, finding witnesses who are reputable, trustworthy entities could be hard. In contrast, Catena could be easier to deploy since it only relies on the Bitcoin blockchain as a single trustworthy witness and on a header relay network that can be bootstrapped using existing blockchain explorers or between Catena clients themselves.

Like CoSi, Catena can offer both "proactive" and "retroactive" security [13]. In particular, Catena can be used retroactively by clients to confirm that previously accepted statements were not lies, or it can be used proactively before accepting a statement as valid. Unlike CoSi, Catena suffers a higher delay when used proactively because clients have to wait for sufficient confirmations before accepting a statement. This is a cost Catena pays for using a decentralized consensus network as its only witness. However, we stress that not all applications will incur this cost. In particular, Catena is suitable for key transparency schemes, Tor directory servers and software transparency schemes which all perform batching and update their state infrequently.

Key transparency schemes can detect equivocation using gossip amongst users [9], [16], [23], [24], gossip between users and trusted validators [26], federated trust [16], anytrust assumptions [25] or non-collusion between actors [25], [26]. Catena instead relies on the resilience of Bitcoin's proof-of-work consensus to prevent, not just detect, equivocation. Our approach can provide proactive security [13] at the cost of publishing new statements every 10 minutes with an average 60-minute confirmation latency (if clients wait for 6 confirmations). Alternatively, Catena can provide retroactive security with no latency. We believe Catena can strengthen key transparency schemes because it enables anyone to audit efficiently for non-equivocation. We also believe Catena's approach to non-equivocation is simpler and more trustworthy due to the decentralized nature of Bitcoin's consensus protocol.

EthIKS [87] uses the Ethereum blockchain [81] to prevent equivocation in CONIKS [16], a key transparency scheme that enables users to efficiently monitor their own public key bindings. EthIKS implements CONIKS as a "smart contract" in the Ethereum blockchain and relies on Ethereum miners to enforce CONIKS security invariants. This is similar to how Catena relies on Bitcoin miners to prevent double spends and thus prevent equivocation in the log. Like Catena, EthIKS also makes equivocation in CONIKS as hard as forking the Ethereum blockchain. However, due to the lack of support for thin nodes in Ethereum [82], EthIKS cannot yet be audited efficiently by clients.

IX. Conclusion

We designed and implemented Catena, an append-only log that is as hard to fork as the Bitcoin blockchain but efficient to verify by thin clients such as mobile phones. Specifically, in Catena, an attacker can equivocate if and only if he can double spend Bitcoin transactions, which is notoriously difficult due to Bitcoin's proof-of-work consensus. The key idea behind Catena is to chain `OP_RETURN` transactions together, making equivocation in the log as hard as double spending in Bitcoin.

Catena can be used to prevent equivocation in key transparency schemes, paving the way for more trustworthy public-key directories. Catena can also be used as a public log for Tor Consensus Transparency [11], as a software transparency scheme to prevent malicious software updates or as a consensus log for mutually distrusting participants. Catena's overheads are small. Clients only need to download 80-byte block headers and 600-byte statements, a significant improvement over previous blockchain-based transparency schemes [14], [15], [30] which currently require auditors to download 90 GB of blockchain data [19]. We developed a first prototype of Catena in Java and we applied it to CONIKS, a key transparency scheme, demonstrating its feasibility. Next, we plan on extending our prototype to scale for popular applications.

Our main reason for designing Catena was to prevent online services from lying. In that sense, we believe Catena can bring Bitcoin's non-equivocation guarantees to many important applications today. In particular, we hope Catena can be adopted by existing secure messaging apps such as Signal [88] or public-key directories such as Keybase [14], allowing end users to get stronger guarantees about non-equivocation.

References

[1] J. B. Andre Niemann, "A Survey on CA Compromises," https://www.cdc.informatik.tu-darmstadt.de/fileadmin/user_upload/Group_CDC/Documents/Lehre/SS13/Seminar/CPS/cps2014_submission_8.pdf, Accessed: 2016-05-15.

[2] "Further improving digital certificate security," https://googleonlinesecurity.blogspot.com.au/2013/12/further-improving-digital-certificate.html, Accessed: 2015-11-06.

[3] R. Mandalia, "Security breach in CA networks - Comodo, DigiNotar, GlobalSign," http://blog.isc2.org/isc2_blog/2012/04/test.html, Accessed: 2015-08-22.

[4] "Enhancing digital certificate security," https://googleonlinesecurity.blogspot.co.uk/2013/01/enhancing-digital-certificate-security.html, Accessed: 2015-11-06.

[5] D. O'Brien, "Web users in the United Arab Emirates have more to worry about than having just their BlackBerries cracked." http://www.slate.com/articles/technology/webhead/2010/08/the_internets_secret_back_door.html, Accessed: 2015-08-22.

[6] H. Adkins, "An update on attempted man-in-the-middle attacks," http://googleonlinesecurity.blogspot.com/2011/08/update-on-attempted-man-in-middle.html, Accessed: 2015-08-22.

[7] C. Soghoian and S. Stamm, "Certified lies: Detecting and defeating government interception attacks against ssl (short paper)," in *Proceedings of the 15th International Conference on Financial Cryptography and Data Security*, ser. FC'11. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 250–259. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-27576-0_20

[8] B. Laurie, A. Langley, and E. Kasper, "RFC: Certificate Transparency," http://tools.ietf.org/html/rfc6962, Accessed: 2015-5-13.

[9] L. Chuat, P. Szalachowski, A. Perrig, B. Laurie, and E. Messeri, "Efficient gossip protocols for verifying the consistency of certificate logs," in *Communications and Network Security (CNS), 2015 IEEE Conference on*, Sept 2015, pp. 415–423.

[10] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The Second-generation Onion Router," in *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, ser. SSYM'04. Berkeley, CA, USA: USENIX Association, 2004, pp. 21–21. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251375.1251396

[11] L. Nordberg, "Tor Consensus Transparency, take two," http://archives.seul.org/tor/dev/Feb-2016/msg00099.html, Accessed: 2016-05-23.

[12] J. Li, M. Krohn, D. Mazières, and D. Shasha, "Secure untrusted data repository (sundr)," in *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, ser. OSDI'04. Berkeley, CA, USA: USENIX Association, 2004, pp. 9–9. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251254.1251263

[13] E. Syta, I. Tamas, D. Visher, D. I. Wolinsky, P. Jovanovic, L. Gasser, N. Gailly, I. Khoffi, and B. Ford, "Keeping authorities "honest or bust" with decentralized witness cosigning," pp. 526–545, 2016. [Online]. Available: http://dx.doi.org/10.1109/SP.2016.38

[14] Keybase.io, "Keybase," http://keybase.io, Accessed: 2016-05-15.

[15] M. Ali, J. Nelson, R. Shea, and M. J. Freedman, "Experiences with Building a Global PKI with Blockchains," in *2016 USENIX Annual Technical Conference (USENIX ATC 16)*. Denver, CO: USENIX Association, Jun. 2016. [Online]. Available: https://www.usenix.org/conference/atc16/technical-sessions/presentation/ali

[16] M. S. Melara, A. Blankstein, J. Bonneau, E. W. Felten, and M. J. Freedman, "Bringing deployable key transparency to end users," in *24th USENIX Security Symposium (USENIX Security 15)*. Washington, D.C.: USENIX Association, Aug. 2015. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/melara

[17] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[18] R. Pass, L. Seeman, and abhi shelat, "Analysis of the blockchain protocol in asynchronous networks," Cryptology ePrint Archive, Report 2016/454, 2016, http://eprint.iacr.org/2016/454.

[19] blockchain.info, "Bitcoin blockchain size over time," https://blockchain.info/charts/blocks-size, Accessed: 2016-11-11.

[20] S. A. Crosby and D. S. Wallach, "Efficient data structures for tamper-evident logging," in *Proceedings of the 18th Conference on USENIX Security Symposium*, ser. SSYM'09. Berkeley, CA, USA: USENIX Association, 2009, pp. 317–334. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855768.1855788

[21] bitcoin.org, "Null Data (OP_RETURN) Transaction," https://bitcoin.org/en/glossary/null-data-transaction, Accessed: 2016-10-13.

[22] "Use OP_RETURN to store merkle root in bitcoin blockchain," https://github.com/keybase/keybase-issues/issues/1104, Accessed: 2016-05-23.

[23] M. D. Ryan, "Enhanced certificate transparency and end-to-end encrypted mail," Cryptology ePrint Archive, Report 2013/595, 2013, http://eprint.iacr.org/.

[24] J. Yu, V. Cheval, and M. Ryan, "DTKI: a new formalized PKI with no trusted parties," *CoRR*, vol. abs/1408.1023, 2014. [Online]. Available: http://arxiv.org/abs/1408.1023

[25] D. Basin, C. Cremers, T. H.-J. Kim, A. Perrig, R. Sasse, and P. Szalachowski, "ARPKI: Attack Resilient Public-Key Infrastructure," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14. New York, NY, USA: ACM, 2014, pp. 382–393. [Online]. Available: http://doi.acm.org/10.1145/2660267.2660298

[26] T. H.-J. Kim, L.-S. Huang, A. Perring, C. Jackson, and V. Gligor, "Accountable Key Infrastructure (AKI): A Proposal for a Public-key Validation Infrastructure," in *Proceedings of the 22Nd International Conference on World Wide Web*, ser. WWW '13. New York, NY, USA: ACM, 2013, pp. 679–690. [Online]. Available: http://doi.acm.org/10.1145/2488388.2488448

[27] B. Ford, "Apple, FBI, and Software Transparency." Freedom To Tinker, Mar. 2016.

[28] "bitcoinj," https://bitcoinj.github.io/, Accessed: 2016-10-10.

[29] bitcoin.org, "Bloom Filter," https://bitcoin.org/en/glossary/bloom-filter, Accessed: 2016-10-19.

[30] J. Nelson, M. Ali, R. Shea, and M. J. Freedman, "Extending existing blockchains with virtualchain," https://www.zurich.ibm.com/dccl/papers/nelson_dccl.pdf, 2016, Accessed: 2016-08-01. [Online]. Available: https://www.zurich.ibm.com/dccl/papers/nelson_dccl.pdf

[31] S. A. Crosby and D. S. Wallach, "Authenticated dictionaries: Real-world costs and trade-offs," *ACM Trans. Inf. Syst. Secur.*, vol. 14, no. 2, pp. 17:1–17:30, Sep. 2011. [Online]. Available: http://doi.acm.org/10.1145/2019599.2019602

[32] Keybase.io, "Keybase is now writing to the Bitcoin blockchain," https://keybase.io/docs/server_security/merkle_root_in_bitcoin_blockchain, Accessed: 2016-05-15.

[33] Blockstack, "Light clients," https://blockstack.org/docs/light-clients, Accessed: 2016-05-22.

[34] R. Shea, Personal communication, Jun. 2016.

[35] bitcoin.org, "0.13.0 Binary Safety Warning," https://bitcoin.org/en/alert/2016-08-17-binary-safety, Accessed: 2016-10-21.

[36] A. Bellissimo, J. Burgess, and K. Fu, "Secure software updates: Disappointments and new challenges," in *Proceedings of the 1st USENIX Workshop on Hot Topics in Security*, ser. HOTSEC'06. Berkeley, CA, USA: USENIX Association, 2006, pp. 7–7. [Online]. Available: http://dl.acm.org/citation.cfm?id=1268476.1268483

[37] J. Cappos, J. Samuel, S. Baker, and J. H. Hartman, "A look in the mirror: Attacks on package managers," in *Proceedings of the 15th ACM Conference on Computer and Communications Security*, ser. CCS '08. New York, NY, USA: ACM, 2008, pp. 565–574. [Online]. Available: http://doi.acm.org/10.1145/1455770.1455841

[38] T. Mendelsohn, "Secure Boot snafu: Microsoft leaks backdoor key, firmware flung wide open," http://arstechnica.com/security/2016/08/microsoft-secure-boot-firmware-snafu-leaks-golden-key/, Accessed: 2016-10-21.

[39] L. Kessem, "Certificates-as-a-Service? Code Signing Certs Become Popular Cybercrime Commodity," https://securityintelligence.com/certificates-as-a-service-code-signing-certs-become-popular-cybercrime-commodity/, Sep. 2015, Accessed: 2016-10-21.

[40] bitcoin.org, "M-of-N Multisig, Multisig Output," https://bitcoin.org/en/glossary/multisig, Accessed: 2016-10-13.

[41] Tor, "Tor Metrics about page," https://metrics.torproject.org/about.html, Accessed: 2016-10-12.

[42] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, "Sok: Research perspectives and challenges for bitcoin and cryptocurrencies," in *2015 IEEE Symposium on Security and Privacy*, May 2015, pp. 104–121.

[43] Narayanan, Arvind and Bonneau, Joseph and Felten, Edward and Miller, Andrew and Goldfeder, Steven, "Bitcoin and cryptocurrency technologies," https://d28rh4a8wq0iu5.cloudfront.net/bitcointech/readings/princeton_bitcoin_book.pdf?a=1, 2016, Accessed: 2016-03-29. [Online]. Available: https://d28rh4a8wq0iu5.cloudfront.net/bitcointech/readings/princeton_bitcoin_book.pdf?a=1

[44] F. Tschorsch and B. Scheuermann, "Bitcoin and beyond: A technical survey on decentralized digital currencies," in *IEEE Communications Surveys Tutorials*, vol. PP, no. 99, 2016, pp. 1–1. [Online]. Available: https://eprint.iacr.org/2015/464.pdf

[45] J. A. D. Donet, C. Pérez-Sola, and J. Herrera-Joancomartí, "The bitcoin p2p network," in *Financial Cryptography and Data Security*. Springer, 2014, pp. 87–102. [Online]. Available: http://fc14.ifca.ai/bitcoin/papers/bitcoin14_submission_3.pdf

[46] J. R. Douceur, "The Sybil Attack," in *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, ser. IPTPS '01. London, UK, UK: Springer-Verlag, 2002, pp. 251–260. [Online]. Available: http://dl.acm.org/citation.cfm?id=646334.687813

[47] R. Merkle, "Method of providing digital signatures," Jan. 5 1982, US Patent 4,309,569. [Online]. Available: https://www.google.com/patents/US4309569

[48] J. A. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, 2015, pp. 281–310. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-46803-6_10

[49] A. Kiayias and G. Panagiotakos, "Speed-security tradeoffs in blockchain protocols," *IACR Cryptology ePrint Archive*, vol. 2015, p. 1019, 2015. [Online]. Available: http://eprint.iacr.org/2015/1019

[50] A. Miller and J. J. LaViola Jr, "Anonymous byzantine consensus from moderately-hard puzzles: A model for bitcoin," p. 5, 2014.

[51] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970. [Online]. Available: http://doi.acm.org/10.1145/362686.362692

[52] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek, "Secure multiparty computations on bitcoin," in *2014 IEEE Symposium on Security and Privacy*, May 2014, pp. 443–458.

[53] C. Garman, M. Green, and I. Miers, "Decentralized anonymous credentials," in *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*, 2014. [Online]. Available: http://www.internetsociety.org/doc/decentralized-anonymous-credentials

[54] J. van den Hooff, M. F. Kaashoek, and N. Zeldovich, "Versum: Verifiable computations over large public logs," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, 2014, pp. 1304–1316. [Online]. Available: http://doi.acm.org/10.1145/2660267.2660327

[55] I. Bentov and R. Kumaresan, "How to use bitcoin to design fair protocols," in *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, 2014, pp. 421–439. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-44381-1_24

[56] J. Clark, J. Bonneau, E. W. Felten, J. A. Kroll, A. Miller, and A. Narayanan, "On decentralizing prediction markets and order books," in *Workshop on the Economics of Information Security, State College, Pennsylvania*, 2014.

[57] J. Clark and A. Essex, *CommitCoin: Carbon Dating Commitments with Bitcoin*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 390–398. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-32946-3_28

[58] bitcoin.org, "11/12 March 2013 Chain Fork Information," https://bitcoin.org/en/alert/2013-03-11-chain-fork, Accessed: 2016-10-17.

[59] —, "Some Miners Generating Invalid Blocks," https://bitcoin.org/en/alert/2015-07-04-spv-mining, Accessed: 2016-10-17.

[60] L. Luu, J. Teutsch, R. Kulkarni, and P. Saxena, "Demystifying incentives in the consensus computer," in *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. New York, NY, USA: ACM, 2015, pp. 706–719. [Online]. Available: http://doi.acm.org/10.1145/2810103.2813659

[61] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on bitcoin's peer-to-peer network," in *24th USENIX Security Symposium (USENIX Security 15)*. Washington, D.C.: USENIX Association, Aug. 2015, pp. 129–144. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/heilman

[62] M. Corallo, "The Bitcoin Relay Network," http://bitcoinrelaynetwork.org/, Accessed: 2016-10-17.

[63] E. G. S. Soumya Basu, Ittay Eyal, "The Falcon Network," http://www.falcon-net.org/, Accessed: 2016-10-17.

[64] M. Corallo, "FIBRE," http://bitcoinfibre.org/, Accessed: 2016-10-17.

[65] "Bitnodes," https://bitnodes.21.co/, Accessed: 2016-11-5.

[66] J. A. Donet Donet, C. Pérez-Solà, and J. Herrera-Joancomartí, *The Bitcoin P2P Network*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 87–102. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-44774-1_7
[67] "blockchain.info API," https://blockchain.info/api/blockchain_api, Accessed: 2016-11-08.
[68] "blockexplorer.com," https://blockexplorer.com/api-ref, Accessed: 2016-11-08.
[69] "blockr.io," https://blockr.io/documentation/api, Accessed: 2016-11-08.
[70] "blocktrail.com," https://www.blocktrail.com/api/docs#api_block, Accessed: 2016-11-08.
[71] N. Evans, B. Polot, and C. Grothoff, *Efficient and Secure Decentralized Network Size Estimation*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 304–317. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-30045-5_23
[72] howtobuybitcoins.info, "How To Buy Bitcoins," https://howtobuybitcoins.info/, Accessed: 2016-10-20.
[73] en.bitcoin.it, "Value overflow incident," https://en.bitcoin.it/wiki/Value_overflow_incident#cite_note-7, Accessed: 2016-10-18.
[74] macbook air, "A successful DOUBLE SPEND US$10000 against OKPAY this morning." https://bitcointalk.org/index.php?topic=152348.0;all, Accessed: 2016-10-14.
[75] en.bitcoin.it, "July 2015 chain forks," https://en.bitcoin.it/w/index.php?title=July_2015_chain_forks&redirect=no, Accessed: 2016-10-21.
[76] ——, "Stratum mining protocol," https://en.bitcoin.it/wiki/Stratum_mining_protocol, Accessed: 2016-10-18.
[77] Y. Sompolinsky and A. Zohar, "Bitcoin's security model revisited," *CoRR*, vol. abs/1605.09193, 2016. [Online]. Available: http://arxiv.org/abs/1605.09193
[78] https://bitcoinfees.github.io/, Accessed: 2016-10-31.
[79] "Bitcoin Fees for Transactions," https://bitcoinfees.21.co/, Accessed: 2016-11-4.
[80] blockchain.info, "Keybase-issued Bitcoin transactions," https://blockchain.info/address/1HUCBSJeHnkhzrVKVjaVmWg2QtZS1mdfaz, Accessed: 2016-05-15.
[81] Gavin Wood, "Ethereum: A Secure Decentralised Generalised Transaction Ledger," http://gavwood.com/paper.pdf, Accessed: 2016-05-15.
[82] "Light client protocol," https://github.com/ethereum/wiki/wiki/Light-client-protocol, Accessed: 2016-11-09.
[83] I. Bentov, A. Gabizon, and A. Mizrahi, *Cryptocurrencies Without Proof of Work*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 142–157. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-53357-4_10
[84] "OpenAssets," https://github.com/OpenAssets, Accessed: 2016-11-09.
[85] "Colu," https://www.colu.com/solutions, Accessed: 2016-11-09.
[86] "OP_RETURN Stats," http://opreturn.org/, Accessed: 2016-11-09.
[87] J. Bonneau, "EthIKS: Using Ethereum to audit a CONIKS key transparency log," BITCOIN'16, 2016, http://www.jbonneau.com/doc/B16b-BITCOIN-ethiks.pdf.
[88] Open Whisper Systems, "Signal," https://whispersystems.org/, Accessed: 2016-11-09.