Catena: Efficient Non-equivocation via Bitcoin

Alin Tomescu
MIT CSAIL

Srinivas Devadas
MIT CSAIL

***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 implement Catena in Java using the *bitcoinj* library and use 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 secure many systems today, such as public-key directories, Tor directory servers and software transparency schemes.**

I. Introduction

Security often depends on *non-equivocation* [1], [2]. For example, when a Certificate Authority (CA) equivocates by signing contradicting certificates for the same identity, it can impersonate websites and compromise users' privacy. In fact, this has happened many times in the past [3]–[9]. To prevent equivocation, Certificate Transparency (CT) [10] has been introduced as a way of publicly logging all CA-issued certificates. However, a CT log server can still equivocate about the log of issued certificates and, together with a colluding CA, can launch impersonation attacks. While gossiping [11] about the log can help detect equivocation, detection can be slow or not happen at all, as gossip messages can be delayed indefinitely. Another example is the Tor [12] anonymity network, where malicious directory servers can equivocate about the set of Tor relays and deanonymize users by tricking them to use malicious relays [13]. Thus, we believe non-equivocation is an important security requirement in many systems today, such as public-key distribution, blockchain-based transparency [14], [15] and software transparency (see §II-A).

Unfortunately, without online trusted parties, achieving non-equivocation is impossible [16]. To deal with this impossibility result, systems resort to enforcing a weaker property called *fork consistency* [16]. Fork-consistent systems essentially make equivocation "permanent" and thus easier to prove later when clients are able to communicate or "gossip" out-of-band. However, as illustrated above, for systems such as public-key directories and Tor directory servers, undetected equivocation attacks can seriously impact users' security. Thus, we believe a more proactive approach [17] to security is desirable for such systems.

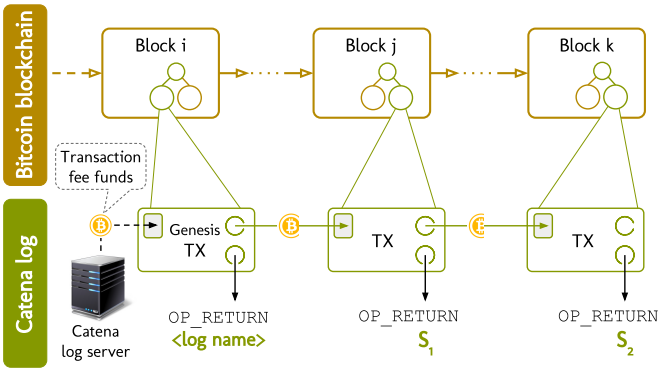

Fig. 1. A Catena log is a chain of Bitcoin transactions. Each Catena transaction has two outputs: (1) a continuation output, which is spent by the next Catena transaction, thus creating a chain and (2) an `OP_RETURN` output, which commits some application-specific statement. The server pays Bitcoin transaction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

To prevent equivocation proactively, recent work [14], [15] uses the Bitcoin blockchain [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 [10], or witnesses as in CoSi [17]. 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 existing Bitcoin witnessing schemes has been that auditors have to download the entire Bitcoin blockchain, which, in November 2016, was almost 90 GB [19] in size and growing by 52 GB every year.

This paper presents *Catena*, an efficient Bitcoin-based witnessing scheme that 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 block-

chain 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) [18] (see §II-B6). This drastically decreases auditing bandwidth 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 decreases to less than 1 KB of data every 10 minutes.

A. Efficient Non-equivocation via Bitcoin

Previous Bitcoin witnessing schemes [14], [15] cannot *efficiently* 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. ($tx_1[0]$ refers to the first output of transaction tx_1; see §II-B4 for Bitcoin background.)

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 §IV-A1). 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 §IV, we explain how this implicitly prevents equivocation in a Catena log. Our design is simple and 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. However, we emphasize that Catena's novelty is not in leveraging `OP_RETURN` (previous work already does that; see §VIII), but in chaining together `OP_RETURN` transactions that contain log statements, which makes it possible to check for non-equivocation efficiently. Furthermore, Catena does not place unnecessary stress on the Bitcoin P2P network. First, clients query the Catena log server directly to discover statements instead of using disk-intensive Bloom filtering on the Bitcoin P2P network (see §II-B6). Second, to avoid depleting the small connection pool of Bitcoin's P2P network, clients query a *header relay network* (HRN) to obtain the latest Bitcoin block headers (see §IV-B). Put simply, the HRN can be thought of as an "extension" of Bitcoin's P2P network for handling additional block header requests coming from Catena clients. We discuss potential attacks on the HRN in §V-E.

3) Applications: Due to Bitcoin's 10-minute block rate, the Catena log server can only issue a statement every 10 minutes while clients have to wait at least 60 minutes before accepting an issued statement. Still, even with these delays, we believe Catena can help secure applications such as key transparency schemes, Tor directory servers and software transparency schemes. We discuss these applications in more detail in §II-A and discuss Catena's application-agnostic nature in §VII-3.

4) Evaluation: To demonstrate the feasibility of Catena, we implement a small-scale prototype in 3000 lines of Java using the bitcoinj Simplified Payment Verification (SPV) library [23] (see §VI). Our current prototype does not include a Header Relay Network (HRN) so it will not scale to too many Catena clients without stressing Bitcoin's P2P network. We leave this to future work. 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 US cents per statement. Since existing systems like Keybase [14] already pay close to 7 US cents per transaction, we believe this cost is practical. Finally, we use our prototype to add Bitcoin witnessing to CONIKS [2], a recent key transparency scheme (see §VI-D), to 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 §II. We describe our system's actors, threat model and goals in §III. We present Catena's design in §IV and we discuss attacks and countermeasures in §V. We discuss our prototype implementation, its overheads and our extension of CONIKS in §VI. We discuss remaining issues and future work in §VII. We go over related work in §VIII and we conclude in §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. At the same time, we want to improve previous blockchain-based transparency schemes [14], [24] whose shortcomings we describe in §II-A2. Finally, we want a non-equivocation scheme that does not require many trustworthy parties to come into existence and that can be deployed today.

1) Key Transparency: Catena can prevent equivocation attacks in current key transparency work [2], [10], [25]–[28] 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 [29]. Users are presented with digests of the directory as it evolves over time and can verify someone's public key against a digest of the directory, preventing equivocation with respect to that digest. The remaining problem for key transparency schemes is to prevent equivocation about the digests themselves. For this, current schemes rely on federated trust [2], any-trust assumptions [27], non-collusion between actors [27], [28] or on users gossiping between themselves [2], [11], [25], [26] or with trusted validators [28].

With Catena, we propose using the Bitcoin blockchain as a hard-to-coerce, trustworthy witness that can vouch for directory digests. 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 look up new STHs in the Catena log and be certain that the log server has not equivocated 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 digests 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 public-key directory [14]. Currently, Keybase publishes digests of their public-key directory in Bitcoin by creating transactions signed by a predetermined public key [30]. Keybase clients recognize these transactions and read directory digests from them (see §VIII for details). The problem with this approach is that thin clients cannot securely use Bloom filtering (see §II-B6) to avoid downloading irrelevant transactions, as an adversary could selectively hide Keybase transactions and equivocate about the directory (we explain this attack in §IV-C). 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 improve Blockstack's thin client security [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 (see §VIII for details). Blockstack clients could also choose to trust someone else's consensus hash and verify public key lookups against it efficiently using Simplified Name Verification (SNV) [15]. However, 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 [31], is that it requires a secret key to manage the Catena log and would thus "centralize" the system. To address this, an alternative design would be to introduce *auditors* who verify and publish Blockstack consensus hashes in a jointly-signed Catena log. While this approach centralizes trust for thin clients, such as mobile phones, it does so in a more accountable and transparent manner. Specifically, the auditors can't equivocate about consensus hashes but can still publish *internally inconsistent* [20] consensus hashes (see §VII-3). However, such misbehavior would be evident in the Bitcoin blockchain when audited by a full Blockstack client.

3) Software Transparency: Catena can prevent equivocation in *software transparency* schemes [32] and thus thwart man-in-the-middle attacks that try to inject malicious software binaries on victims' machines [32]. In fact, Bitcoin developers were concerned in the past about these kinds of attacks on Bitcoin binaries [33]. 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 [17] already highlights the necessity of software transparency in the face of insecure software update schemes [34], [35], key loss or compromise [36] and black markets for code-signing certificates [37].

4) Tor Directory Servers: Catena can be used to prevent Tor directory servers [12] from equivocating 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 [13] 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 [38]. Since Tor does not try to conceal who is connected to the network [12], we are not concerned

about Catena's header relay network learning who is using Tor. Finally, because Tor consensus is updated every hour, we believe it should be suitable for embedding in a Catena log.

5) 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 secret key and does not necessarily trust the other $n-1$ servers. In this scheme, each server submits an operation to the log by creating a Catena transaction that is spendable by all n servers (see §III-A1). To disincentivize the other servers from stealing the coins, the log is funded with small amounts of bitcoins and is frequently "re-funded" (see §IV-E). This scheme allows all servers to reach consensus on the log and relies on Bitcoin miners to decide which server's operation gets included in the log. To prevent adversarial servers from monopolizing the log with their operations by paying higher transaction fees, the servers can agree on an upper bound on fees.

B. Bitcoin Background

Bitcoin [18], [39]–[41] 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* [42] 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 [43]. 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 Bitcoin by allowing everyone to agree on the correct log of transactions while preventing Sybil attacks [44]. 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.

1) P2P Network: Bitcoin uses a peer-to-peer (P2P) network of volunteer nodes to store the blockchain [43], listen for new transactions or new blocks, and propagate this information throughout the network. Users, such as merchants and their customers, download the blockchain by becoming part of the P2P network and can then receive or issue Bitcoin transactions. Miners are also part of the P2P network where they listen for new blocks and broadcast their own blocks.

2) Blockchain: Bitcoin's "blockchain" is implemented as a hash-chain of *blocks* (see Figure 2) and keeps track of all transactions in the system, allowing anyone to verify that no double spends have occurred. A Bitcoin block is made up of a set of transactions (up to 1 MB) and a small *block header* (80 bytes) that contains a hash pointer to the previous block. The transactions in the block are hashed in a Merkle tree [45] whose root hash is stored inside the block header. The Merkle tree allows Bitcoin *thin clients* (see §II-B6) to obtain efficient *membership proofs* that a transaction is part of a block.

3) Decentralized Consensus: To solve the consensus problem in the decentralized or *permissionless* setting, where

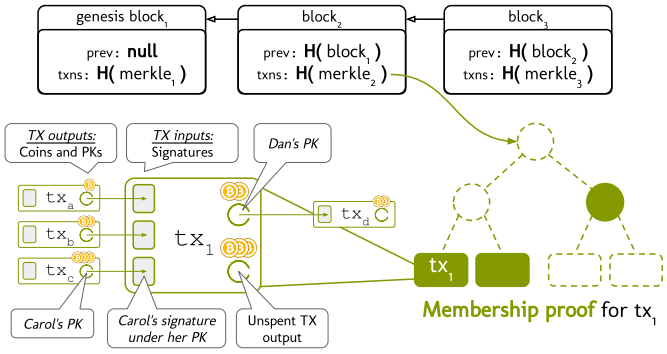

Fig. 2. The Bitcoin blockchain is a hash chain of blocks. Each block has a Merkle tree of transactions. Efficient membership proofs of transactions can be constructed with respect to the Merkle root. Here, tx_1 transfers coins from Alice, Bob and Carol to Dan and somebody else (miners receive a fee of 1 coin). Alice authorizes the transfer of her coins by signing tx_1, which has an input pointing to her coins locked in the 1st output of tx_a. Bob and Carol do the same. Similarly, Dan later spends his coins locked in tx_1's 1st output by signing a new transaction tx_d with an input pointing to tx_1's 1st output.

participants can enter and leave the protocol as they please, Bitcoin introduces a novel Byzantine consensus protocol called *proof-of-work consensus* [42], [46]–[48]. Though it does so at a high computational cost, this protocol defeats Sybil attacks [44] and achieves consensus on the blockchain if 51% of the computational power amongst participants remains honest.

Participants called *miners* race to solve computationally-difficult proof-of-work puzzles derived from the previous Bitcoin block. If a miner finds a solution, the miner can publish the next block by announcing it along with the solution (in reality, the solution is part of the next block) over the P2P network. Furthermore, this miner will receive a *block reward* in bitcoins, an incentive for miners to participate in the consensus protocol. The puzzle difficulty is adjusted every 2016 blocks based on the inferred computational power of the miners, or *network hashrate*, so that a new block is found or "mined" on average every 10 minutes.

When two miners find a solution at the same time, the Bitcoin blockchain is said to *accidentally fork* into two chains. In this case, Bitcoin peers use the *heaviest chain rule* and select the heavier fork as the *main chain* that dictates consensus. The *weight* of a fork is simply the amount of computational work expended to create that fork. Assuming no difficulty changes, the heaviest fork is the longest fork. However, across difficulty changes, it could be that a fork with fewer blocks is heavier than a longer fork (though this never happens in practice).

During an accidental fork, both forks have the same length and weight (assuming the fork does not cross a difficulty recomputation point), so Bitcoin peers adopt the fork they saw first as their main chain. As more blocks are mined, one of the forks becomes heavier than the other and is accepted as the main chain by the whole network [42]. In this case, the other abandoned fork and its blocks are said to 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.

4) Transactions: Bitcoin transactions facilitate the transfer of coins between users (see Figure 2). A Bitcoin transaction has an arbitrary number of *transaction inputs*, which authorize the transfer of coins, and *transaction outputs* (TXOs), which specify who receives those coins and in what amounts. Naturally, the number of coins locked in the outputs cannot exceed the number of coins specified in the inputs (with the exception of so-called "coinbase" transactions, which mint new coins and have no inputs). 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 only make use of 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 TXO there exists at most one transaction input that refers to or spends that TXO. This invariant is known as the *TXO invariant* and Catena leverages it to prevent forks. Finally, a transaction's fee is the difference between the coins spent in its inputs and the coins transferred by its outputs. The fee is awarded to the miner who mines a block containing that transaction. In theory, the fee can be zero, but in practice recent contention for space in the blockchain requires users to pay transaction fees.

5) Storing Data in Transactions: Bitcoin allows users to store up to 80 bytes of data in transactions through provably-unspendable `OP_RETURN` transaction outputs. Importantly, any coins specified in the output are forever unspendable or "burned". For simplicity, Catena uses `OP_RETURN` outputs to store application-specific statements in the Bitcoin blockchain (see §IV). However, there are other ways to store data in Bitcoin transactions: in the value of transferred coins [49], in transaction inputs [50], in transaction sequence numbers [49], or in an output's public key (either via vanity public keys [49], fake public keys [30], multisig public keys [38] or "pay-to-contract" public keys [51]).

6) Thin Nodes vs. Full Nodes: Bitcoin's P2P network is made up of *full nodes*, which download the entire blockchain and validate all the transactions (see §II-B1) and *thin nodes*, which only download small 80 byte block headers and cannot fully validate transactions. Since full nodes are more expensive to run (higher bandwidth, computation and space), smaller devices such as smartphones can run thin nodes instead, also known as *Simplified Payment Verification* (SPV) nodes.

Thin nodes verify Bitcoin transactions more efficiently under a slightly stronger assumption about the Bitcoin network. A thin node considers a transaction valid if it sees a correct Merkle proof of membership for that transaction in a block. Furthermore, the more blocks are mined after a transaction's block (also known as *confirmations*), the more confident a thin node can be that the transaction 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. As a result, thin nodes assume Bitcoin miners follow their incentives and create correct blocks or otherwise thin nodes could accept invalid transactions. This assumption can be reasonable since miners would lose their block reward if they create invalid blocks (see §II-B3).

Finally, the only way for thin nodes to avoid downloading unnecessary data is to use a Bitcoin feature called Bloom filtering [52]. This feature allows thin nodes to only receive transactions of interest by asking remote peers to filter out irrelevant transactions using a Bloom filter [53]. Bloom filtering is cheap for the requesting thin client but quite expensive for the servicing full node, which has to load all requested blocks from disk, pass them through the filter and send filtered blocks to the thin client.

III. Model and Goals

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

A. Actors

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. We call these transactions *Catena transactions* and defer their discussion to §IV-A2. In this paper, we will mostly talk about a single log server managing the log, but by using Bitcoin multisignatures [38], 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 (discussed next). Catena clients want to prevent log server equivocation: 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 (see §III-D1).

3) Header Relay Network: Due to the low connection capacity of the Bitcoin P2P network (see §IV-B), Catena clients use a separate header relay network (HRN) to obtain Bitcoin block headers (see §IV-B). Otherwise, Catena would put unnecessary stress on Bitcoin's P2P network and would not scale well.

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.

$S.\mathsf{CreateLog}(d) \rightarrow (\mathsf{sk}, \mathsf{pk})$. Creates an empty log. All future log statements can be verified against the log's public key pk. Embeds some arbitrary data d in the log (e.g., the log's name).

$S.\mathsf{AppendStmt}(\mathsf{sk}, s_i)$. Appends the statement s_i to the log, signing it using sk.

$C.\mathsf{VerifyStmt}(\mathsf{pk}, i, s_i) \rightarrow \{\mathsf{True}, \mathsf{False}\}$. Verifies that the statement s_i is contained in the log with public key pk at position i. Returns true if successful or false otherwise. Before being called on s_i, VerifyStmt must first be called on $s_1, s_2, \ldots, s_{i-1}$, in that order.

To recap, a server creates a new log by calling CreateLog and appends statements to this log using AppendStmt. Clients verify each new statement s_i by calling VerifyStmt in order for $i = 1, 2, 3, \cdots$.

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 §IV-A1). We note that both Catena and previous work [2], [10], [17], [25], [27], [28] all rely on some sort of initial public-key distribution. However, unlike previous work, Catena can prevent equivocation once a client has the "public key" or genesis transaction. It's important to understand that, similarly 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, we argue it might be easier for end-users to verify the genesis transaction if they know the log's creation date. Specifically, users can download just the blocks around that date and check that no other genesis transaction for the log exists in those blocks.

2) Proof-of-Work Consensus: Similar to previous work [14], [15], [24], [54]–[59], 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. Finally, we have to assume Bitcoin's P2P network is reliable and miners hear about each other's blocks quickly, or else proof-of-work consensus could be easily subverted [42], [60]. We discuss attacks on Bitcoin's consensus in more detail in §V.

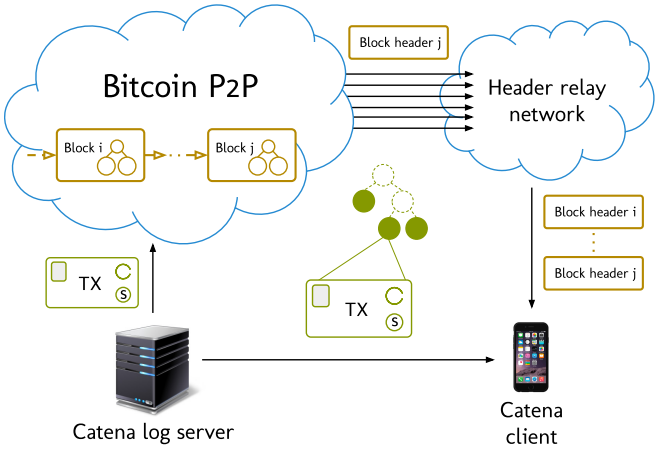

Fig. 3. The log server broadcasts Catena transactions with statements embedded in them to the Bitcoin P2P network. Catena clients query the header relay network for block headers and the log server for statements with proofs they were witnessed in the Bitcoin blockchain. The header relay network maintains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

3) SPV Assumption: Catena clients use thin nodes (see §II-B6) to efficiently verify the log for non-equivocation. It's important to note that thin nodes are less secure than full nodes against adversarial mining attacks (see §V-C). Also, thin nodes have to assume miners verify their own blocks and the blocks of other miners before mining, otherwise thin nodes risk accepting invalid transactions. Fortunately, Bitcoin miners have a strong incentive to verify blocks, as they would lose the block reward if they extend an invalid blockchain. However, recent work [61] shows that when block verification is expensive miners have an incentive to skip it. We discuss such an event that occurred in 2015 in §V-B.

4) Header Relay Network: We trust Catena's header relay network to serve Catena clients with the latest Bitcoin block headers. Similar to a compromised Bitcoin P2P network, a compromised HRN can eclipse [60] Catena clients and help adversaries win mining races faster and thus equivocate (see §V-C). However, such adversaries would need a significant fraction of mining power to win races fast enough without Catena clients noticing they are being eclipsed. We discuss such attacks in §V-E.

D. Goals

Our goals are to prevent equivocation and to do so in an efficiently-verifiable manner, enabling each user to audit individually and thus minimizing trust in 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 [2], [11], [25], [26].

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 see all issued statements, including incorrect ones. This allows clients to detect attacks at the application layer, as we discuss later in §VII-3.

2) Publicly Verifiable: Given a log's genesis transaction $\mathsf{tx}_{\mathsf{genesis}}$ (i.e., its public key), 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 $\mathsf{tx}_{\mathsf{genesis}}$ that 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 [2], [14], [15] is inefficient, requiring auditors to download the entire blockchain to prevent equivocation (see §II-A2). This 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: clients only download 80-byte block headers for each Bitcoin block and 600-byte Merkle membership proofs for each statement (see §IV-C).

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 equivocated 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 that spends the genesis transaction and commits that first statement via an `OP_RETURN` transaction output (see §II-B5). Finally, the server can append new statements to the log by creating a new transaction that spends the previously-created transaction and commits a new statement.

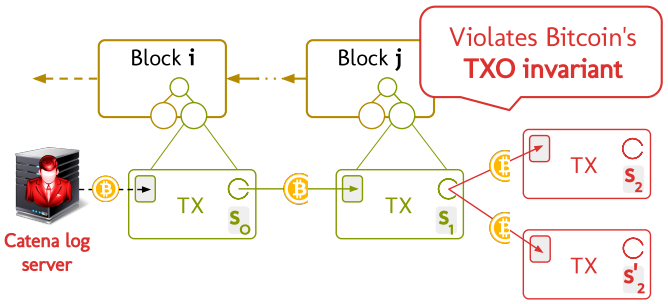

Fig. 4. Equivocating in a Catena log is as hard as double spending in Bitcoin, which requires forking the blockchain (see §II-B). This is because Catena's design requires a new Catena transaction to spend the previous one, which linearizes the history of statements embedded in those transactions.

Catena clients first obtain the log's genesis transaction, which can be shipped with the higher-level application that Catena secures (see §III-C1). Then, clients obtain and verify all Bitcoin block headers from the header relay network (discussed in §IV-B). 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" (see Figure 4) and Bitcoin prevents double spends, clients are assured the server has not equivocated (see §III-D1).

Catena's overhead is small. For each 32-byte statement, the server sends over a 235-byte Catena transaction and a Merkle path of up to 350 bytes 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.

A. Transaction Format

1) 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. As discussed in §III-C1, a "public key" such as the genesis transaction is a necessary element of any system which aims to prevent equivocation.

2) Catena transactions: A Catena log is just a chain of specially-crafted Bitcoin transactions called *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, and extra inputs for "re-funding" the log (see §IV-E). 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 (see 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 spends that output* (see TXO invariant in §II-B4). 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 can 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 §II-B4 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 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 transactions.

As mentioned before, the log server has to pay fees to Bitcoin miners to get its transactions included in the blockchain. We describe how this works in §IV-E and we analyze the server's cost per Catena statement in §VI-C1.

B. Header Relay Network

We want to avoid stressing 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 [43], [62]. Thus, Bitcoin's P2P network currently supports at most 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 Catena clients to discover and connect to a diverse set of Bitcoin peers. As a result, if each Catena client were to maintain 8 outgoing connections to Bitcoin's P2P network, then Catena would not scale beyond tens of thousands of clients without putting significant stress on the Bitcoin network.

To provide scalability, 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. An HRN node operates as a full node in the Bitcoin P2P network, contributing to its health while providing an interface to Catena clients for obtaining block headers fast. However, note that HRN nodes do not mine nor attempt to reach consensus on block headers: they just gossip and verify blocks like the rest of the Bitcoin P2P network. Catena clients trust the HRN to serve them with the latest Bitcoin block headers. Importantly, clients ask multiple HRN nodes for block headers to ensure they are not being eclipsed by a single malicious HRN node. We discuss attacks on the HRN in §V-E.

A header relay network can be bootstrapped in various ways. The simplest way is to have a set of volunteer HRN nodes that act as an extension of the Bitcoin P2P network. Another way is to rely on current blockchain explorers [63]–[66] since they are well connected to the Bitcoin network and already provide public APIs for fetching block headers. A diverse HRN could be implemented by publishing block headers across various websites, such as Twitter, Facebook or GitHub, in a publicly-verifiable manner similar to how Keybase [14] users publish identity proofs. A peer-to-peer HRN can be bootstrapped amongst Catena clients themselves. Catena clients can occasionally fetch block headers from the Bitcoin P2P network and then distribute them amongst themselves. To avoid stressing Bitcoin P2P nodes, each client would query the Bitcoin P2P network with probability inversely proportional to the size of HRN (estimated using known techniques [67]). Finally, Sybil attacks [44] in all these types of HRNs can be addressed by requiring HRN nodes to "burn" bitcoins in a publicly-verifiable manner (see §II-B5) and tie their identity to those burned coins.

C. Auditing a Catena Log

To audit a log, clients download the Catena transaction chain and verify that transactions are signed and spend each other 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. This way, Catena clients avoid Bloom filtering on Bitcoin's P2P network, which causes significant disk activity for full nodes (see §II-B6). 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).

To verify a new Catena transaction tx_i, a client checks that:

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

It's important to understand that without clients verifying transaction chaining (i.e., step 3 and 4 above), a malicious log server can equivocate about statements in the log. For example, consider two Catena clients c_1 and c_2 which correctly obtain the genesis transaction $\mathsf{tx}_{\mathsf{genesis}}$ of the log but do not verify transactions are chained. 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 shows client c_1 the transaction for s_1 but hides the one for s_1'. Similarly, it shows client c_2 the transaction for s_1' and hides the one for s_1. As a result, the log server can easily equivocate to clients who don't

verify transaction chaining as they cannot ensure the server is not hiding an inconsistent statement. To conclude, in Catena, clients prevent this attack by checking that every statement they accept is part of a transaction that spends the previous transaction's continuation output, chaining all the way back to the genesis transaction.

D. Blockchain Reorganizations

Like Bitcoin, Catena also needs to deal with small day-to-day accidental forks or *blockchain reorganizations* (see §II-B3). 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 [42]. 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 allows clients to set their own application-specific number of required confirmations before accepting a statement (a minimum of 6 is recommended). 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.

E. 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 [68]. Then, the server can issue the log's genesis transaction and pay for its fee. The server locks 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 are 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. We analyze the costs of running a Catena log in terms of transaction fees in §VI-C1.

To "re-fund" the chain, Catena transactions can have additional inputs that lock 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.

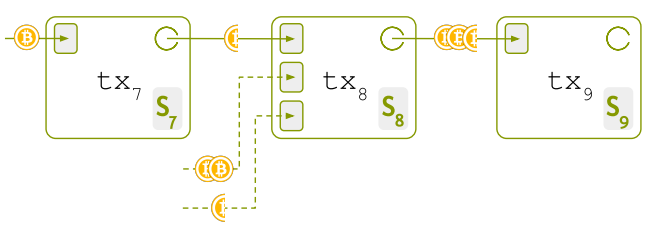

Fig. 5. A Catena chain can be "re-funded" by allowing the next transaction in the chain to have additional inputs that lock extra coins in that transaction's continuation output. In this example, Catena transactions pay .5 BTC as a fee so to ensure tx_8 does not run out of coins we "re-fund" it using extra inputs.

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 also describe attacks launched by a compromised log server or a compromised header relay network.

A. Log Server Attacks

An attacker might compromise the Catena log server and steal its statement key. In this case, the attacker can issue his own statements, but he cannot fork the log to equivocate about statements. That is, all clients will see all attacker-issued statements and can check their correctness at the application layer (see §VII-3). 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 [69].

Once the attacker has the statement key, he can 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. A compromised log server could also 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 above, the log server cannot equivocate about statements as that would require double spending a transaction in Bitcoin.

B. 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 [70], [71] and one of them, in July 2015, was caused by at least one irrational miner [72], 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 [71], 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 [73].

We stress that accidental forks have been rare and are thus outside of our threat model. Furthermore, clients find out about forks via the header relay network and refuse accepting statements until forks are resolved, which gives them an extra line of defense. Thus, an adversary who wants to exploit an accidental fork has to compromise the header relay network to hide one of the forks (or compromise the Bitcoin P2P network instead). As a last line of defense, Catena clients can wait for additional confirmations to protect themselves against accidental forks at the cost of additional latency.

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 [74]. "SPV" normally stands for Simplified Payment Verification as discussed in §II-B6, but here it is used to indicate that miners are not verifying the block they are mining on. SPV mining is used by some rational miners as a way to lower their rate of orphaned blocks by starting to mine earlier [61]. 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 [75]. Then, they mine 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 mining rewards [72].

SPV mining remains a concern for the 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 and block verification speed as well as new fast block relay networks, such as Falcon [76] and FIBRE [77].

C. 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 6-block long side chain that is at least one block longer than the main chain [78]. The side chain's first block 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 ceases to mine, issues tx' to the main chain and lets 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 [78].

Similar to previous work [14], [15], [24], [54]–[59], we exclude adversaries who can mine a 6-block 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.

D. Bitcoin P2P Network Attacks

An attacker can "eclipse" nodes on the P2P network and withhold newly mined blocks from them via Sybil attacks [44] and so-called eclipse attacks [60]. For example, an attacker who eclipses a Catena client can increase their chances of succeeding at an adversarial mining attack. Even worse, an attacker who eclipses Bitcoin miners can double spend *without adversarially mining* by simply preventing miners from seeing each other's blocks. Fortunately, eclipse attacks on miners have not been observed yet in the wild and it is not clear that they could remain undetected for long. First, countermeasures against eclipse attacks have been already implemented in Bitcoin's P2P network code. Second, mining pool operators would quickly notice the fork by an increase in their fraction of mined blocks. Third, eclipse attacks could be detected faster in the future if miners also broadcast "status reports:" block headers that are below the difficulty target but are sufficiently difficult to give information about how much mining power is behind a fork [79]. Finally, block relay networks such as the Bitcoin Fast Relay Network (FRN) [80], Falcon [76] or FIBRE [77] are being deployed or are already deployed between Bitcoin miners, making eclipsing miners much harder.

A more powerful attacker could simply Sybil-attack Bitcoin's P2P network, which would constitute a break of Bitcoin itself and, if practical, would be a concern for both Bitcoin and Catena. We plan on investigating to what extent a Sybil attack can partition the Bitcoin P2P network in future work.

E. Header Relay Network Attacks

A sufficiently powerful attacker who can adversarially mine and who controls the header relay network (HRN) could equivocate to a Catena client. First, the attacker controlling

the HRN eclipses the client from the Bitcoin network, hiding all newly-mined blocks from the client. Second, the attacker adversarially mines a sufficiently long side chain that confirms some fake statement s_i. Depending on the attacker's mining power, this could take days or weeks, which means the victim would become suspicious, as they are not seeing any mining activity. When done, the attacker shows this side chain to the victim who will accept s_i. Finally, the attacker stops eclipsing the victim and shows them the main chain that confirms an inconsistent statement s_i' and unconfirms s_i. Importantly, even without an HRN in our design, this attack would be possible via an adversarial Bitcoin P2P network [60].

Fortunately, this attack can be easily detected by Catena clients if they use their local time to compute the rate at which blocks are mined and compare it to the normal Bitcoin rate while accounting for variance in the time between blocks. However, such heuristics for detecting attacks are beyond the scope of our work, so we defer them to future work. Finally, note that a more powerful attacker could leverage control of the HRN network to make generalized "Vector76" attacks more likely to succeed (see §V-C). Specifically, as he gets closer to successfully mining the 6-block long side chain, the attacker could eclipse the victim, which effectively buys him some extra time to win the race against the Bitcoin network. As before, such a powerful attacker could also pull off this attack using Bitcoin's P2P network should our design not require an HRN. The same countermeasures as discussed above and in §V-C could be used to prevent this attack.

VI. Prototype and Evaluation

We implemented a Catena prototype in Java using the bitcoinj [23] library in 3000 lines of code, as measured with the `sloccount` tool. Our code is available on GitHub:

`https://github.com/non-equivocation/catena-java`

Our prototype implements a Catena log server and a Catena client, both operating as thin nodes on the Bitcoin network, but does not implement a header relay network (HRN). Instead, in our first implementation, Catena clients use only the Bitcoin P2P network to fetch both block headers and fetch 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. Next, we discuss our implementation and its internal API, which can be used to implement Catena's API from §III-B.

A. Catena Log Server

The log server manages the statement key used to sign new Catena statements (see §IV-A2) and a set of *funding keys* used to "re-fund" a Catena log (see §IV-E). The server provides an appendStatement(s) API for issuing a statement s, which abstracts the Bitcoin layer away from applications. Though currently not implemented, the server should "re-fund" the chain automatically assuming there are sufficient funds controlled by the funding keys.

B. Catena Log Client

The client connects to the Bitcoin P2P network and sets a Bloom filter [53] on all its connections to filter out irrelevant transactions. This way, Catena clients only receive server-issued Catena transactions (see §II-B6) and save orders of magnitude in bandwidth. Recall that Catena chains together transactions and clients verify this, thereby preventing malicious P2P nodes from equivocating about statements (see §IV-C). While additional bandwidth will be consumed by small blockchain reorganizations (see §IV-D), this amount should be negligible.

Catena clients expose an onStatementAppended(s) API that notifies the higher level application of newly issued statements that 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 (see §VII-3). If the Catena log is caught equivocating, the Catena client notifies the application via an onEquivocation(s, s') API that 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 onReorganize() API that 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 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 [82] (1 satoshi = 10^{-8} BTC). For a 235-byte Catena transaction that commits a statement consisting of a 256-bit SHA-256 hash, the fee would be 16,450 satoshis or 12 US cents per statement (on November 2016, 1 BTC = $706.54). 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 issues statements less often (every 6 hours), paying a smaller fee of 10,000 satoshis or 7 US cents per transaction [83].

2) Overheads: Catena's CPU overhead is insignificant. A Catena log server can issue at most one statement per Bitcoin block, so it only has to perform one signature every 10

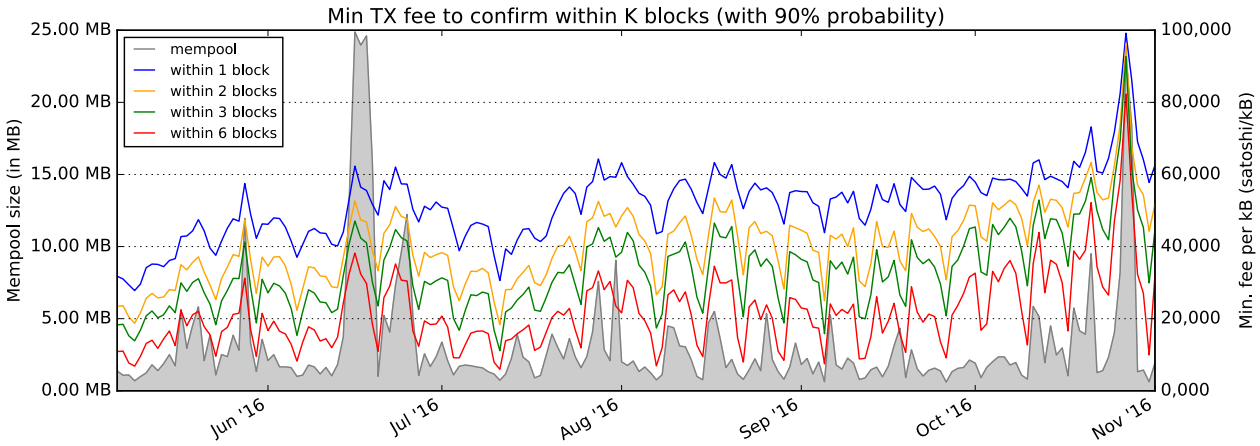

Fig. 6. This graphs shows the minimum transaction fee (in satoshi/kB) that guarantees a transaction will be included in the blockchain within k blocks (for $k = 1, 2, 3, 6$) with 90% probability (modeled using Feesim [81]). Fees tend to increase with contention for space in the Bitcoin "mempool" of unconfirmed transactions and transactions with higher fees get included in the blockchain faster. 1 satoshi = 0.00000001 BTC = 10^{-8} BTC and 1 kB = 10^3 bytes.

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 clients need 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 equivocation about withdrawn statements during blockchain reorganizations (see §IV-D), Catena clients remember the past 100 statements issued by the server (no more than 80 bytes each due to `OP_RETURN` limits; see §II-B5). 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 §II-B3). Once this is done, Catena clients need 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, possibly requested from multiple HRN nodes. In contrast, the server needs bandwidth linear in the number of Catena clients, since it serves every statement to each client.

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 §IV-B, there simply aren't enough connections available in the Bitcoin network to support a large number of Catena clients. In addition, our current implementation relies on disk-intensive Bloom filtering (see §II-B6). 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 Equivocation in CONIKS

To demonstrate Catena's applicability to key transparency schemes, we modified CONIKS [2] to publish directory digests 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. (We changed Java source files, project files and configuration files.)

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 once per minute to once per ten 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 ob-

taining the "public keys" for the Catena logs of providers. Specifically, our "public key" is the log's genesis transaction (see §IV-A1). We commit the old public key of the provider in the auxiliary data of the genesis transaction (see §III-B). CONIKS clients need this public key to verify CONIKS server replies to their queries.

VII. Discussion and Future Work

1) Building Catena on Top of Bitcoin: We chose to design Catena on top of Bitcoin because of Bitcoin's resilient proof-of-work consensus [42] and its real-world deployment. This makes Catena-enabled applications easy to deploy (no need to wait for trustworthy parties to come into existence) and expensive to attack (adversaries have to double spend in Bitcoin). Still, it's important to note that Bitcoin's security as a "black box" consensus protocol remains an open problem. For example, it can be difficult to dismiss externally-motivated adversaries who are well-incentivized to maliciously mine and double spend. Finally, we note that Catena could also be built on top of other blockchains such as Ethereum [84], but we believe Bitcoin's security currently outmatches the security of all other blockchains.

In particular, we avoided Ethereum for a few reasons. First, we believe Bitcoin is a more mature ecosystem to base applications on, given the many blockchain-based apps built on top of it [14], [15], [85]–[87]. Second, Ethereum plans on transitioning to a proof-of-stake consensus algorithm [88] called Casper [89] that could change the trust assumptions behind thin nodes. For instance, an additional assumption in Casper is that clients who are offline for too long can authenticate a list of "bonded" validators out-of-band [89]. Third, recent work shows that rational Ethereum miners have an incentive to skip verifying "expensive" blocks that other miners constructed maliciously [61]. In Bitcoin, such attacks are less practical since block verification does not involve executing arbitrarily complex smart contracts.

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 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* [20] of statements, which needs to be checked at the application layer. Instead, Catena only guarantees *historical consistency* [20] of statements, 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) [10] see the same history of signed tree heads (STHs). However, clients 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 [10] can equivocate, giving Alice a signed tree head (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.

VIII. Related Work

Tamper-evident logging [20] allows auditors to ensure a log's 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 address equivocation attacks, assuming 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.

Proofs of proofs of work (PPOW) [90] enable thin clients to verify the "weight" of a blockchain (see §II-B3) by downloading only $O(\log n)$ rather than all n block headers. Should Bitcoin adopt this interesting technique, Catena clients could leverage it to download fewer block headers from the HRN. First, we can leverage PPOWs to decrease the bootstrapping bandwidth of Catena clients from $O(n)$ to $O(\log n)$ when initially syncing with a blockchain of n blocks. Second, we can leverage PPOWs to skip downloading some of the block headers between two statements s_i and s_{i+1}, should there be a large number of blocks between consecutive statements. However, we cannot leverage PPOWs to skip downloading Catena transactions and their corresponding block headers. This is because Catena clients need to verify that all Catena transactions are chained correctly (see §IV-C).

Peter Todd's uniquebits [91] allows users to publish signed hashes of arbitrary data in the Bitcoin blockchain for later auditing. At a high level, the scheme commits the signed data d and PGP fingerprint information about the signer by leveraging both "Pay-to-Script Hash" (P2SH) transactions [50]

and "fake" public keys (see §II-B5). Unfortunately, uniquebits does not address the equivocation problem: the scheme cannot efficiently prevent a signer from publishing two pieces of data d and d' and equivocating to thin clients about which piece was signed. Similarly, CommitCoin [59] uses the Bitcoin blockchain to "timestamp" commitments and prove they were made at a certain time, but cannot efficiently prevent equivocation.

Concomitant with the publication of our online version of Catena [92], Peter Todd independently proposed a Bitcoin-based log that prevents equivocation using *single-use seals* [93]. Single-use seals generalize the concept of spending a transaction output in Bitcoin while committing some arbitrary data in the process. The transaction output waiting to be spent is the *identifier* of the single-use seal, while the transaction that spends that output and commits some arbitrary data (see §II-B5) is known as the *closing transaction*. Importantly, when a seal is "closed" by spending its output via a closing transaction, another seal can be specified as the arbitrary data in that transaction. This is similar to how, in Catena, a new transaction spends the previous transaction's continuation output (equivalent to "closing a seal") and creates a new continuation output (equivalent to specifying a new seal).

As opposed to Catena, single-use seals provide a certain degree of censorship-resistance because the identifier of the next seal can be hidden from miners by committing to it in the closing transaction. In contrast, with Catena, the next continuation output (see §IV-A2) is always public and known by miners, so they can censor transactions that try to spend it. However, Catena is more efficient than single-use seals, since each statement requires a single transaction to be posted on the blockchain, while single-use seals require two transactions (one transaction for the seal's identifier plus another closing transaction). Furthermore, although thin client verification of a log would be possible with single-use seals, the details of this are never considered in depth [93], a contribution that Catena makes. Finally, to the best of our knowledge, single-use seals have not been implemented yet, making Catena the first implementation of an efficient Bitcoin-based log.

Keybase [14] and Blockstack [15], [24] use the Bitcoin blockchain to prevent equivocation but do so inefficiently, requiring clients to run a full Bitcoin node. Keybase periodically publishes the Merkle root of its public-key directory by committing it in transactions signed under a fixed public key known by Keybase clients [83]. Specifically, Keybase stores the Merkle root in the transaction's output as a "fake" public key (see §II-B5). Unfortunately, this approach does not allow clients to efficiently and correctly obtain all Keybase-issued transactions using Bloom filtering (see §II-B6). This is because Bitcoin P2P nodes can selectively hide transactions from clients and show different transactions to different users, thus equivocating about the Keybase directory (see §IV-C). In Catena, a new transaction always spends the previous one, creating a unique chain due to the difficulty of double spending. As a result, all Catena clients see the same history of transactions, implicitly agreeing on the history of statements.

In Blockstack [15], users submit their own operations (e.g., "register public key" or "update public key") to the Bitcoin blockchain by creating transactions that include these operations in an `OP_RETURN` output. Blockstack nodes download the full Bitcoin blockchain and filter these Blockstack-specific transactions (recognized via a magic byte in the `OP_RETURN` data), accumulating the Blockstack operations into a *consensus hash* [15]. Importantly, thin clients can query Blockstack nodes for public keys (e.g., "look up Alice's public key") and authenticate their responses against a consensus hash. Unfortunately, because thin clients cannot download the entire blockchain, they would have to obtain consensus hashes from a trusted entity which, if compromised, could equivocate about these hashes. In this sense, Catena could make equivocation about consensus hashes as hard as forking Bitcoin, thereby increasing Blockstack's thin client security (discussed in §II-A2).

Coin coloring schemes [85]–[87] leverage the Bitcoin blockchain to enable the secure issuance and transfer of assets different than bitcoins, such as smart property, stocks or bonds [94]. The key idea is that a set of coins locked in an output can be assigned a static color which can be correctly maintained as those coins change hands. To prevent double spending of colored coins, coloring schemes also leverage Bitcoin's security against double spends along with some additional verification by "color-aware" wallets. The overhead of these schemes is similar to Catena's, since "color-aware" (thin) wallets only need to keep track of transactions that affect an asset (e.g., reassigned ownership).

Coin coloring schemes and Catena both rely on the difficulty of double spending but do so to solve slightly different problems. Specifically, while coin coloring schemes prevent equivocation about the never-changing color of a coin, Catena prevents equivocation about an ever-growing log of statements. While some coloring schemes support committing arbitrary data in their transactions [95] and could be adapted to prevent equivocation, to the best of our knowledge, this has not yet been done. Thus, we believe Catena to be the first system that solves the equivocation problem efficiently using Bitcoin.

Non-equivocation contracts [96] disincentivize equivocation by penalizing it with monetary loss. Specifically, if an authority equivocates, its Bitcoin secret key, which locks some funds, is implicitly revealed via a mechanism similar to double-authentication-preventing signatures (DAPS) [97]. As a result, anybody who detects equivocation can spend those funds. The advantage of non-equivocation contracts is that statements are not included in the Bitcoin blockchain so they can be issued faster than in Catena. This enables interesting applications, such as asynchronous payment channels [96], which are not possible with Catena. However, as opposed to Catena, non-equivocation contracts only disincentivize equivocation and do not necessarily prevent it. For example, an outsider who steals the authority's secret key and wants to harm the authority is actually incentivized to equivocate and can easily do so. With Catena, even with a stolen secret key, an outsider cannot equivocate without forking the Bitcoin blockchain.

CoSi [17] prevents equivocation by requiring a threshold number of "witnesses" to also verify and sign an authority's

statements. Depending on the application, these witnesses could also check the internal consistency of statements (see §VII-3). In contrast, Catena prevents equivocation by assuming its authority, Bitcoin, is trustworthy, without relying too much on the header relay network (HRN) to keep it honest (see §V-E). Both CoSi and Catena make assumptions about connectivity of participants. CoSi requires a relatively well-connected set of witnesses for its tree broadcast scheme while Catena requires the Bitcoin P2P network not to be easily partitioned. One drawback of CoSi is that it requires an admission control process for witnesses to prevent Sybil attacks [44]. 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 (see §IV-B).

Like CoSi, Catena can offer both "proactive" and "retroactive" security [17]. In particular, Catena can be used retroactively by clients to validate previously accepted statements, 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 care about 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 [2], [11], [25], [26], gossip between users and trusted validators [28], federated trust [2], any-trust assumptions [27] or non-collusion between actors [27], [28]. 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 [17] 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 [98] uses the Ethereum blockchain [84] to prevent equivocation in CONIKS [2], 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. Like Catena, EthIKS also efficiently leverages proof-of-work consensus within a cryptocurrency to prevent equivocation in the log. However, different from Catena, EthIKS guarantees internal consistency (see §VII-3) of the CONIKS log, though this comes at the expense of additional Ethereum transaction fees paid by the EthIKS log server [98]. In contrast, Catena clients have to check each log statement for internal consistency themselves, incurring an overhead linear in the number of statements in the log. For certain applications where internal consistency checks are not expensive (e.g., monitoring your own binding in a public-key directory) and minimizing server costs is a priority, Catena could be better suited. However, when server costs are not a concern (e.g., costs can be shifted to users), Ethereum-based approaches like EthIKS could be better suited.

IX. Conclusion

We design and implement 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 by having a new transaction spend the previous one, 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 [13], 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], [24] which currently require auditors to download 90 GB of blockchain data [19]. We develop a prototype of Catena in Java and apply it to CONIKS, a key transparency scheme, demonstrating Catena's feasibility. Next, we plan on extending our prototype to scale for popular applications.

Our main reason for designing Catena is to prevent equivocation in compromised online services. 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 secure messaging apps such as Signal [99] or public-key directories such as Keybase [14], giving end users stronger guarantees about non-equivocation.

Acknowledgment

We would like to thank our shepherd, Jay Lorch, and the anonymous reviewers for their insightful comments that helped improve this paper. We would also like to thank Conner Fromknecht, Albert Kwon, Ilia Lebedev, Neha Narula, Ling Ren and the MIT Digital Currency Initiative for the many productive conversations that shaped and improved this work.

References

[1] A. J. Feldman, A. Blankstein, M. J. Freedman, and E. W. Felten, "Social Networking with Frientegrity: Privacy and Integrity with an Untrusted Provider," in *21st USENIX Security Symposium (USENIX Security '12)*. Berkeley, CA, USA: USENIX Association, 2012, pp. 647–662.

[2] 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)*. Berkeley, CA, USA: USENIX Association, Aug. 2015, pp. 383–398.

[3] A. Niemann and J. Brendel, "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.

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

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

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

[7] 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.

[8] 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.

[9] C. Soghoian and S. Stamm, "Certified Lies: Detecting and Defeating Government Interception Attacks Against SSL (Short Paper)," in *Financial Cryptography and Data Security (FC 2011)*. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 250–259.

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

[11] 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.

[12] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The Second-generation Onion Router," in *13th USENIX Security Symposium (USENIX Security '04)*. Berkeley, CA, USA: USENIX Association, 2004.

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

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

[15] M. Ali, J. Nelson, R. Shea, and M. J. Freedman, "Blockstack: A Global Naming and Storage System Secured by Blockchains," in *2016 USENIX Annual Technical Conference (USENIX ATC 16)*. USENIX Association, Jun. 2016, pp. 181–194.

[16] 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 (OSDI '04)*. Berkeley, CA, USA: USENIX Association, 2004.

[17] 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," in *2016 IEEE Symposium on Security and Privacy*, May 2016, pp. 526–545.

[18] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," https://bitcoin.org/bitcoin.pdf, 2008, Accessed: 2017-03-08.

[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 *18th USENIX Security Symposium (USENIX Security '09)*. Berkeley, CA, USA: USENIX Association, 2009, pp. 317–334.

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

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

[23] M. Hearn, "bitcoinj," https://bitcoinj.github.io/, Accessed: 2016-10-10.

[24] 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.

[25] M. D. Ryan, "Enhanced certificate transparency and end-to-end encrypted mail," in *21st Annual Network and Distributed System Security Symposium (NDSS 2014)*, Feb. 2014.

[26] J. Yu, V. Cheval, and M. Ryan, "DTKI: A New Formalized PKI with Verifiable Trusted Parties," *The Computer Journal*, vol. 59, no. 11, pp. 1695–1713, 2016.

[27] 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 (CCS '14)*. New York, NY, USA: ACM, 2014, pp. 382–393.

[28] 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 (WWW '13)*. New York, NY, USA: ACM, 2013, pp. 679–690.

[29] S. A. Crosby and D. S. Wallach, "Authenticated Dictionaries: Real-World Costs and Trade-Offs," *ACM Trans. Inf. Syst. Secur.*, vol. 14, no. 2, Sep. 2011.

[30] 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.

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

[32] B. Ford, "Apple, FBI, and Software Transparency," https://freedom-to-tinker.com/2016/03/10/apple-fbi-and-software-transparency/, Mar. 2016, Accessed: 2017-03-08.

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

[34] 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 (HOTSEC '06)*. Berkeley, CA, USA: USENIX Association, 2006.

[35] 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 SIGSAC Conference on Computer and Communications Security (CCS '08)*. New York, NY, USA: ACM, 2008, pp. 565–574.

[36] 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.

[37] 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.

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

[39] 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.

[40] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder, *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton University Press, 2016.

[41] F. Tschorsch and B. Scheuermann, "Bitcoin and Beyond: A Technical Survey on Decentralized Digital Currencies," in *IEEE Communications Surveys Tutorials*, vol. 13, no. 3, 2016, pp. 2084–2123.

[42] 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.

[43] J. A. Donet Donet, C. Pérez-Solà, and J. Herrera-Joancomartí, "The Bitcoin P2P Network," in *Financial Cryptography and Data Security (FC 2014)*, R. Böhme, M. Brenner, T. Moore, and M. Smith, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 87–102.

[44] J. R. Douceur, "The Sybil Attack," in *Revised Papers from the First International Workshop on Peer-to-Peer Systems (IPTPS '01)*. London, UK, UK: Springer-Verlag, 2002, pp. 251–260.

[45] R. C. Merkle, "Method of providing digital signatures," U.S. Patent 4 309 569, Jan. 5, 1982.

[46] 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*, Apr. 2015, pp. 281–310.

[47] A. Kiayias and G. Panagiotakos, "Speed-Security Tradeoffs in Blockchain Protocols," Cryptology ePrint Archive, Report 2015/1019, 2015, http://eprint.iacr.org/2015/1019.

[48] A. Miller and J. J. LaViola Jr, "Anonymous Byzantine Consensus from Moderately-Hard Puzzles: A Model for Bitcoin," University of Central Florida, Tech. Rep., 2014.

[49] A. Shomer and O. Leiba, "Data Storage Methods," https://github.com/Colored-Coins/Colored-Coins-Protocol-Specification/wiki/Data%20Storage%20Methods, Accessed: 2017-02-16.

[50] R. de Vries, "CIP Proposal: P2SH data encoding," https://counterpartytalk.org/t/cip-proposal-p2sh-data-encoding/2169/1, Accessed: 2017-02-16.

[51] I. Gerhardt and T. Hanke, "Homomorphic Payment Addresses and the Pay-to-Contract Protocol," *CoRR*, vol. abs/1212.3257, 2012. [Online]. Available: http://arxiv.org/abs/1212.3257

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

[53] B. H. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970.

[54] 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.

[55] C. Garman, M. Green, and I. Miers, "Decentralized Anonymous Credentials," in *21st Annual Network and Distributed System Security Symposium (NDSS 2014)*, Feb. 2014.

[56] 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 (CCS '14)*, Nov. 2014, pp. 1304–1316.

[57] I. Bentov and R. Kumaresan, "How to Use Bitcoin to Design Fair Protocols," in *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference*, Aug. 2014, pp. 421–439.

[58] 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 (WEIS 2014)*, Jun. 2014.

[59] J. Clark and A. Essex, "CommitCoin: Carbon Dating Commitments with Bitcoin," in *Financial Cryptography and Data Security (FC 2012)*, A. D. Keromytis, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, Feb. 2012, pp. 390–398.

[60] 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)*. Berkeley, CA, USA: USENIX Association, Aug. 2015, pp. 129–144.

[61] 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 (CCS '15)*. New York, NY, USA: ACM, 2015, pp. 706–719.

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

[63] blockchain.info, "Blockchain Data API," https://blockchain.info/api/blockchain_api, Accessed: 2016-11-08.

[64] blockexplorer.com, "API Reference," https://blockexplorer.com/api-ref, Accessed: 2016-11-08.

[65] blockr.io, "blockr.io Blockchain API Code," https://blockr.io/documentation/api, Accessed: 2016-11-08.

[66] blocktrail.com, "BlockTrail API Documentation," https://www.blocktrail.com/api/docs#api_block, Accessed: 2016-11-08.

[67] N. Evans, B. Polot, and C. Grothoff, "Efficient and Secure Decentralized Network Size Estimation," in *11th International IFIP TC 6 Networking Conference (NETWORKING 2012)*, R. Bestak, L. Kencl, L. E. Li, J. Widmer, and H. Yin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, May 2012, pp. 304–317.

[68] howtobuybitcoins.info, "How To Buy Bitcoins," https://howtobuybitcoins.info/, Accessed: 2016-10-20.

[69] R. Gennaro, S. Goldfeder, and A. Narayanan, "Threshold-optimal DSA/ECDSA signatures and an application to Bitcoin wallet security," Cryptology ePrint Archive, Report 2016/013, 2016, http://eprint.iacr.org/2016/013.

[70] en.bitcoin.it, "Value overflow incident," https://en.bitcoin.it/wiki/Value_overflow_incident#cite_note-7, Accessed: 2016-10-18.

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

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

[73] 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.

[74] 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.

[75] ——, "Stratum mining protocol," https://en.bitcoin.it/wiki/Stratum_mining_protocol, Accessed: 2016-10-18.

[76] S. Basu, I. Eyal, and E. G. Sirer, "The Falcon Network," http://www.falcon-net.org/, Accessed: 2016-10-17.

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

[78] Y. Sompolinsky and A. Zohar, "Bitcoin's Security Model Revisited," *CoRR*, vol. abs/1605.09193, 2016. [Online]. Available: http://arxiv.org/abs/1605.09193

[79] A. P. Ozisik, G. Andresen, G. D. Bissias, A. Houmansadr, and B. N. Levine, "A Secure, Efficient, and Transparent Network Architecture for Bitcoin," UMass Amherst, Tech. Rep. UM-CS-2016-006, 2016. [Online]. Available: https://web.cs.umass.edu/publication/details.php?id=2417

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

[81] bitcoinfees.github.io, "Bitcoin Fee Estimation," https://bitcoinfees.github.io/, Accessed: 2016-10-31.

[82] bitcoinfees.21.co, "Bitcoin Fees for Transactions," https://bitcoinfees.21.co/, Accessed: 2016-11-4.

[83] blockchain.info, "Keybase-issued Bitcoin transactions," https://blockchain.info/address/1HUCBSJeHnkhzrVKVjaVmWg2QtZS1mdfaz, Accessed: 2016-05-15.

[84] G. Wood, "Ethereum: A Secure Decentralised Generalised Transaction Ledger," http://gavwood.com/paper.pdf, Accessed: 2016-05-15.

[85] OpenAssets, "Colored coins and the Open Assets protocol," https://github.com/OpenAssets, Accessed: 2016-11-09.

[86] Colu, "Colu - local digital wallet," https://www.colu.com, Accessed: 2017-02-15.

[87] CoinSpark, "Bitcoin upgraded with messaging and assets," http://coinspark.org, Accessed: 2017-02-15.

[88] I. Bentov, A. Gabizon, and A. Mizrahi, "Cryptocurrencies Without Proof of Work," in *Financial Cryptography and Data Security (FC 2016)*, J. Clark, S. Meiklejohn, P. Y. Ryan, D. Wallach, M. Brenner, and K. Rohloff, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, Feb. 2016, pp. 142–157.

[89] V. Zamfir, "Introducing Casper 'the Friendly Ghost'," https://blog.ethereum.org/2015/08/01/introducing-casper-friendly-ghost/, Accessed: 2017-02-17.

[90] A. Kiayias, N. Lamprou, and A.-P. Stouka, "Proofs of Proofs of Work with Sublinear Complexity," in *Financial Cryptography and Data Security (FC 2016)*, J. Clark, S. Meiklejohn, P. Y. Ryan, D. Wallach, M. Brenner, and K. Rohloff, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 61–78.

[91] P. Todd, "uniquebits," https://github.com/petertodd/uniquebits, Accessed: 2017-02-15.

[92] A. Tomescu and S. Devadas, "Catena: Preventing Lies with Bitcoin," Cryptology ePrint Archive, Report 2016/1062, 2016, http://eprint.iacr.org/2016/1062.

[93] P. Todd, "Preventing Consensus Fraud with Commitments and Single-Use-Seals," https://petertodd.org/2016/commitments-and-single-use-seals, Accessed: 2017-03-07.

[94] M. Rosenfeld, "Overview of Colored Coins," https://bitcoil.co.il/BitcoinX.pdf, 2012, Accessed: 2017-02-15.

[95] A. Shomer, O. Leiba, and E. Alsheich, "Metadata," https://github.com/Colored-Coins/Colored-Coins-Protocol-Specification/wiki/Metadata, Accessed: 2017-03-08.

[96] T. Ruffing, A. Kate, and D. Schröder, "Liar, Liar, Coins on Fire!: Penalizing Equivocation By Loss of Bitcoins," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15)*. New York, NY, USA: ACM, 2015, pp. 219–230.

[97] B. Poettering and D. Stebila, "Double-authentication-preventing signatures," *International Journal of Information Security*, pp. 1–22, 2015.

[98] J. Bonneau, "EthIKS: Using Ethereum to Audit a CONIKS Key Transparency Log," in *Financial Cryptography and Data Security (FC 2016)*, J. Clark, S. Meiklejohn, P. Y. Ryan, D. Wallach, M. Brenner, and K. Rohloff, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, Feb. 2016, pp. 95–105.

[99] Open Whisper Systems, "Signal," https://whispersystems.org/, Accessed: 2016-11-09.