Chapter 6

Eventual Consistency & Bitcoin

How would you implement an ATM? Does the following implementation work satisfactorily?

Algorithm 6.1 Naïve ATM

```
1: Issue withdrawal request to bank
2: Wait for response from bank
3: if balance of customer sufficient then
4:     Dispense cash
5: else
6:     Display error
7: end if
```

Remarks:

- A connection problem between the bank and the ATM may block Algorithm 6.1 in Line 2.

- A *network partition* is a failure where a network splits into at least two parts that cannot communicate with each other. Intuitively any non-trivial distributed system cannot proceed during a partition *and* maintain consistency. In the following we introduce the tradeoff between consistency, availability and partition tolerance.

- There are numerous causes for partitions to occur, e.g., physical disconnections, software errors, or incompatible protocol versions. From the point of view of a node in the system, a partition is similar to a period of sustained message loss.

6.1 Consistency, Availability and Partitions

Definition 6.2 (Consistency)**.** *All nodes in the system agree on the current state of the system.*

Definition 6.3 (Availability). *The system is operational and instantly processing incoming requests.*

Definition 6.4 (Partition Tolerance). *Partition tolerance is the ability of a distributed system to continue operating correctly (consistently and available) even in the presence of a network partition.*

Theorem 6.5 (CAP Theorem). *It is impossible for a distributed system to simultaneously provide Consistency, Availability and Partition Tolerance. A distributed system can satisfy any two of these but not all three.*

Proof. Assume two nodes, sharing some state. The nodes are in different partitions, i.e., they cannot communicate. Assume a request wants to update the state and contacts a node. The node may either: 1) update its local state, resulting in inconsistent states, or 2) not update its local state, i.e., the system is no longer available for updates. □

Algorithm 6.6 Partition tolerant and available ATM

```
if bank reachable then
  Synchronize local view of balances
  if balance of customer insufficient then
    Display error and abort
  end if
end if
Dispense cash
Log withdrawal for synchronization
```

Remarks:

- Algorithm 6.6 is partition tolerant and available since it continues to process requests even when the bank is not reachable.
- The ATM's local view of the balances may diverge from the balances as seen by the bank, therefore consistency is no longer guaranteed.
- The algorithm will synchronize any changes it made to the local balances back to the bank once connectivity is re-established. This is known as eventual consistency.

Definition 6.7 (Eventual Consistency). *If no new updates to the shared state are issued, then eventually the system is in a quiescent state, i.e., no more messages need to be exchanged between nodes, and the shared state is consistent.*

Remarks:

- Eventual consistency is a form of *weak consistency.*
- Eventual consistency guarantees that the state is eventually agreed upon, but the nodes may disagree temporarily.
- During a partition, different updates may by semantically conflicting with each other. A *conflict resolution* mechanism is required to resolve the conflicts and allow the nodes to eventually agree on a common state.

- One example of eventual consistency is the Bitcoin cryptocurrency system.

6.2 Bitcoin

Definition 6.8 (Bitcoin Network)**.** *The Bitcoin network is a randomly connected overlay network of a few thousand* ***nodes****, controlled by a variety of users or corporations. All nodes perform the same operations, i.e., it is a homogenous network and without central control.*

Remarks:

- The lack of structure is intentional: it ensures that an attacker cannot strategically position itself in the network and manipulate the information exchange. Information is exchanged via a simple broadcasting protocol.
- Not everyone needs to run a fully validating node, and end-users will likely use a lightweight client that only temporarily connects to the network.

Definition 6.9 (Address)**.** *An address is a textual representation of the hash of an elliptic curve public key. A private/public key pair is used to uniquely identify the owner of funds an address.*

Remarks:

- A user may generate an arbitrary number of addresses which serve as the user's identities in the system. It is hard to link addresses to the user that controls them, hence Bitcoin is often referred to as being *pseudonymous.*
- The Bitcoin network collaboratively tracks the balance in bitcoins of each address.
- The textual representation of an address is composed of a network identifier byte, the hash of the public key and a checksum. It is commonly stored in base 58 encoding, a custom encoding similar to base 64 with some ambiguous symbols removed, e.g., lowercase letter "l" since it is similar to the number "1".
- The hashing algorithm produces addresses of size 20 bytes. This means that there are 2^{160} distinct addresses. It might be tempting to brute force a target address, however at one billion trials per second one still requires approximately 2^{45} years in expectation to find a matching private/public key pair. Due to the birthday paradox the odds improve if instead of brute forcing a single address we attempt to brute force any address. While the odds of a successful trial increase with the number of addresses, lookups become more costly.

Definition 6.10 (Output)**.** *An output is a tuple consisting of an amount of bitcoins and a spending condition. Most commonly the spending condition requires a valid signature associated with the private key of an address.*

Remarks:

- Spending conditions are scripts that offer a variety of options. Examples include conditions that require the result of a simple computation to be provided, or the solution to a cryptographic puzzle.
- Outputs exist in two states: unspent and spent. Any output can be spent at most once. The address balance is the sum of bitcoin amounts in unspent outputs that are associated with the address.
- The set of unspent transaction outputs (UTXO) and the rules about how they can be spent is the shared state of Bitcoin. Every node in the Bitcoin network holds a complete replica of that state. Local replicas may temporarily diverge, but consistency is eventually re-established.

Definition 6.11 (Input)**.** *An input is a tuple consisting of a reference to a previously created output and arguments to the spending condition, proving that the transaction creator has the permission to spend the referenced output.*

Definition 6.12 (Transaction)**.** *A transaction is a datastructure that describes the transfer of bitcoins from spenders to recipients. The transaction consists of a number of inputs and new outputs. The inputs result in the referenced outputs being marked as spent (removed from the UTXO), and the new outputs being added to the UTXO.*

Remarks:

- Inputs reference the output that is being spent by a (h, i)-tuple, where h is the hash of the transaction that created the output, and i specifies the index of the output in that transaction.
- Transactions are broadcast in the Bitcoin network and processed by every node that receives them.

Algorithm 6.13 Node Receives Transaction

```
Receive transaction $t$
for each input $(h, i)$ in $t$ do
  if output $(h, i)$ is spent in local UTXO or signature invalid then
    Drop $t$ and stop
  end if
end for
if sum of values of inputs $<$ sum of values of outputs then
  Drop $t$ and stop
end if
for each input $(h, i)$ in $t$ do
  Mark $(h, i)$ as spent in local UTXO
end for
Append $t$ to local history
Forward $t$ to neighbors in the Bitcoin network
```

Remarks:

- Note that the effect of a transaction on the state is deterministic. In other words if all nodes receive the same set of transactions in the same order (Definition 1.8), then the state across nodes is consistent.
- Notice that so far we only described a local acceptance policy. Nothing prevents nodes to locally accept different transactions that spend the same output.
- Transactions are in one of two states: unconfirmed or confirmed. Incoming transactions from the broadcast are unconfirmed and added to a pool of transactions called the *memory pool*.

Definition 6.14 (Doublespend)**.** *A doublespend is a situation in which multiple transactions attempt to spend the same output. Only one transaction can be valid since outputs can only be spent once. Nodes accepting different transactions in a doublespend do not agree on the validity of those transactions, and the global state is inconsistent.*

Remarks:

- Doublespends may occur naturally, e.g., if outputs are co-owned by multiple users. However, often doublespends are intentional – we call these doublespend-attacks: In a transaction, an attacker pretends to transfer an output to a victim, only to doublespend the same output in another transaction back to itself.
- Doulespends result in an inconsistent state since the validity of transactions depends on the order in which they arrive. If two conflicting transactions are seen by a node, the node considers the first to be valid, see Algorithm 6.13. The second transaction is invalid since it spends an output that is already spent. The order in which transactions are seen may not be the same for all nodes, hence the inconsistent state.
- If doublespends are not resolved, the system state diverges. Therefore a conflict resolution mechanism is needed to decide which of the conflicting transactions is to be confirmed (accepted by everybody), to achieve eventual consistency.

Definition 6.15 (Proof-of-Work)**.** *Proof-of-Work (PoW) is a mechanism that allows a party to prove to another party that a certain amount of computational resources have been utilized for a period of time. A function $\mathcal{F}_d(c, x) \rightarrow \{true, false\}$, where difficulty d is a positive number, while challenge c and nonce x are usually bit-strings, is called a Proof-of-Work function if it has following properties:*

1. *$\mathcal{F}_d(c, x)$ is fast to compute if d, c, and x are given.*
2. *For fixed parameters d and c, finding x such that $\mathcal{F}_d(c, x) = true$ is computationally difficult but feasible. The difficulty d is used to adjust the time to find such an x.*

Algorithm 6.16 The Bitcoin Proof-of-Work function

```
Nonce $x = 0$, challenge $c$, difficulty $d$
$d_{min} = 2^{224}$
$target = d_{min}/d$
repeat
    $x = x + 1$
until SHA256(SHA256($c|x$)) $< target$
return $x$
```

Remarks:

- In Bitcoin the PoW function $\mathcal{F}_d$ concatenates the challenge and nonce, and hashes them twice using SHA256. The output of SHA256 is a cryptographic hash with a numeric value in $\{0, \ldots, 2^{256} - 1\}$ which is compared to the target value. The target value is the ratio between minimum difficulty d_{min} and the current difficulty d.

- SHA256 is a cryptographic hash function with pseudorandom output. No better algorithm is known to find an input x such that the function returns true than simply iterating over possible inputs. This is by design to make it difficult to find such an input, but simple to verify the validity once it has been found.

- If all nodes were computing the PoW for the same challenge, then the fastest node would always win. However, as we will see in Definition 6.18, each node attempts to find a valid PoW for a challenge specific to the node.

Definition 6.17 (Block)**.** *A block is a datastructure used to communicate incremental changes to the local state of a node. A block consists of a list of transactions, a reference to a previous block and a nonce. A block lists some transactions the block creator has accepted to its memory-pool since the previous block. The creation of a block is a rare global event thanks to the use of a PoW mechanism.*

Remarks:

- With their reference to a previous block, the blocks build a tree, rooted in the so called *genesis block*.

- The PoW mechanism allows the block creator ("*miner*") to vary the nonce until a valid PoW is found. The difficulty is adjusted so that globally a block is found about every 10 minutes in expectation.

- The primary goal of the PoW mechanism is to adjust the rate at which blocks are found in the network, giving the network time to synchronize on the latest block.

- Finding a block allows the finder to impose changes in its local memory pool to all other nodes. Upon receiving a block, all nodes roll back any local changes since the previous block and apply the new block's changes.

- Transactions contained in a block are said to be *confirmed* by that block.

Definition 6.18 (Reward Transaction)**.** *The first transaction in a block is called the reward transaction. The block's miner is rewarded for confirming transactions by allowing it to mint new coins. The reward transaction has a dummy input, and the sum of outputs is determined by a fixed amount and the sum of the fees of transactions confirmed in the block.*

Remarks:

- A reward transaction is the sole exception to the rule that the sum of inputs must be at least the sum of outputs.
- The number of bitcoins that are minted by the reward transaction and assigned to the miner is determined by a subsidy schedule that is part of the protocol. Initially the subsidy was 50 bitcoins for every block, and it is being halved every 210,000 blocks, or 4 years in expectation. Due to the halving of the block reward, the total amount of bitcoins in circulation never exceeds 21 million bitcoins.
- It is expected that the cost of performing the PoW to find a block, in terms of energy and infrastructure, is close to the value of the reward the miner receives from the reward transaction in the block.

Definition 6.19 (Blockchain)**.** *The longest path from the genesis block, i.e., root of the tree, to a leaf is called the blockchain. The blockchain acts as a consistent transaction history on which all nodes eventually agree.*

Remarks:

- The path length from the genesis block to block b is the height h_b.
- Only the longest path from the genesis block to a leaf is a valid transaction history, since branches may contradict each other because of doublespends.
- Since only transactions in the longest path are agreed upon, miners have an incentive to append their blocks to the longest chain, thus agreeing on the current state.
- The mining incentives quickly increased the difficulty of the PoW mechanism: initially miners used CPUs to mine blocks, but CPUs were quickly replaced by GPUs, FPGAs and even application specific integrated circuits (ASICs) as bitcoins appreciated. This results in an equilibrium today in which only the most cost efficient miners, in terms of hardware supply and electricy, make a profit in expectation.
- If multiple blocks are mined more or less concurrently, the system is said to have *forked*. Forks happen naturally because mining is a distributed random process and two new blocks may be found at roughly the same time.

Algorithm 6.20 Node Receives Block

```
Receive block $b$
Current head is block $b_{max}$ at height $h_{max}$
Connect block $b$ in the tree as child of its parent $p$ at height $h_b = h_p + 1$
if $h_b > h_{max}$ then
    $h_{max} = h_b$
    $b_{max} = b$
    Compute UTXO for the path leading to $b_{max}$
    Cleanup memory pool
end if
```

Remarks:

- Algorithm 6.20 describes how a node updates its local state upon receiving a block. Notice that, like Algorithm 6.13, this describes the local policy and may also result in node states diverging, i.e., by accepting different blocks at the same height as current block.

- Unlike extending the current path, switching paths may result in confirmed transactions no longer being confirmed, because the blocks in the new path do not include them. Switching paths is referred to as a *reorg*.

- Cleaning up of the memory pool involves 1) removing transactions that were confirmed in a block in the current path, 2) removing transactions that conflict with confirmed transactions, and 3) adding transactions that were confirmed in the previous path, but are no longer confirmed in the current path.

- In order to avoid having to recompute the entire UTXO at every new block being added to the blockchain, all current implementations use datastructures that store undo information about the operations applied by a block. This allows efficient switching of paths and updates of the head by moving along the path.

Theorem 6.21. *Forks are eventually resolved and all nodes eventually agree on which is the longest blockchain. The system therefore guarantees eventual consistency.*

Proof. In order for the fork to continue to exist, pairs of blocks need to be found in close succession, extending distinct branches, otherwise the nodes on the shorter branch would switch to the longer one. The probability of branches being extended almost simultaneously decreases exponentially with the length of the fork, hence there will eventually be a time when only one branch is being extended, becoming the longest branch. □

6.3 Smart Contracts

Definition 6.22 (Smart Contract)**.** *A smart contract is an agreement between two or more parties, encoded in such a way that the correct execution is guaranteed by the blockchain.*

Remarks:

- Contracts allow business logic to be encoded in Bitcoin transactions which mutually guarantee that an agreed upon action is performed. The blockchain acts as conflict mediator, should a party fail to honor an agreement.
- The main enabler for smart contracts is the use of scripts in inputs and outputs to setup the spending conditions and arguments. Scripts, together with some additional features such as timelocks, allow encoding complex conditions, specifying who may spend the funds associated with an output and when.

Definition 6.23 (Timelock)**.** *Bitcoin provides a mechanism to make transactions invalid until some time in the future:* ***timelocks****. A transaction may specify a locktime: the earliest time, expressed in either a Unix timestamp or a blockchain height, at which it may be included in a block and therefore be confirmed.*

Remarks:

- Transactions with a timelock are not released into the network until the timelock expires. It is the responsibility of the node receiving the payment to store the transaction locally until the timelock expires and then release it to the network.
- Nodes in the Bitcoin network discard transactions with future timelocks. Any block including the transaction, that appears at a lower height or before the specified time, is deemed invalid.
- Timelocks can be used to replace or supersede transactions: a timelocked transaction t_1 can replaced by another transaction t_0, spending some of the same outputs, if the replacing transaction t_0 has an earlier timelock and can be broadcast in the network before the replaced transaction t_1 becomes valid.

Definition 6.24 (Singlesig and Multisig Outputs)**.** *When an output can be claimed by providing a single signature it is called a* ***singlesig output****. In contrast the script of* ***multisig outputs*** *specifies a set of m public keys and requires k-of-m (with $k \leq m$) valid signatures from distinct matching public keys from that set in order to be valid.*

Remarks:

- Most smart contracts begin with the creation of a 2-of-2 multisig output, requiring a signature from both parties. Once the transaction creating the multisig output is confirmed in the blockchain, both parties are guaranteed that the funds of that output cannot be spent unilaterally.

Algorithm 6.25 Parties A and B create a 2-of-2 multisig output o

```
$B$ sends a list $I_B$ of inputs with $c_B$ coins to $A$
$A$ selects its own inputs $I_A$ with $c_A$ coins
$A$ creates transaction $t_s\{[I_A, I_B], [o = c_A + c_B \rightarrow (A, B)]\}$
$A$ creates timelocked transaction $t_r\{[o], [c_A \rightarrow A, c_B \rightarrow B]\}$ and signs it
$A$ sends $t_s$ and $t_r$ to $B$
$B$ signs both $t_s$ and $t_r$ and sends them to $A$
$A$ signs $t_s$ and broadcasts it to the Bitcoin network
```

Remarks:

- t_s is called a *setup transaction* and is used to lock in funds into a shared account. If t_s is signed and broadcast immediately then one of the parties not collaborating to spend the multisig output would result in the funds becoming unspendable. To avoid a situation where the funds cannot be spent the protocol also creates a timelocked *refund transaction* t_r which guarantees that, should the funds not be spent before the timelock expires, the funds are returned to the respective party. At no point in time one of the parties holds a fully signed setup transaction without the other party holding a fully signed refund transaction guaranteeing that funds are eventually returned.

- Both transactions require the signature of both parties. In the case of the setup transaction because it has two inputs from A and B respectively which require individual signatures. In the case of the refund transaction the single input spending the multisig output requires both signatures being a 2-of-2 multisig output.

Algorithm 6.26 Simple Micropayment Channel from S to R with capacity c

```
$c_S = c$, $c_R = 0$
$S$ and $R$ use Algorithm 6.25 to set up output $o$ with value $c$ from $S$
Create settlement transaction $t_f\{[o], [c_S \rightarrow S, c_R \rightarrow R]\}$
while channel open and $c_r < c$ do
    In exchange for good with value $\delta$
    $c_R = c_R + \delta$
    $c_S = c_S - \delta$
    Update $t_f$ with outputs $[c_R \rightarrow R, c_S \rightarrow S]$
    $S$ signs and sends $t_f$ to $R$
end while
$R$ signs latest $t_f$ and broadcasts it
```

Remarks:

- Algorithm 6.26 implements a Simple Micropayment Channel, a smart contract that is used for rapidly adjusting micropayments from a sender to a receiver. Only two transactions are ever broadcast and inserted into the blockchain: the setup transaction t_s and the last settlement transaction t_f. There may have been any number of updates to the settlement transaction, transferring ever more of the shared output to the recipient.

- The number of bitcoins c used to fund the channel is also the maximum total that may be transferred over the simple micropayment channel.

- At any time the recipient R is guaranteed to eventually receive the bitcoins, since she holds a fully signed settlement transaction, while the sender only has partially signed ones.

- The simple micropayment channel is intrinsically unidirectional, since the recipient may choose any of the settlement transactions in the protocol, she will use the one with maximum payout for her. If we were to transfer bitcoins back we would be reducing the amount paid out to the recipient, hence she would choose not to broadcast that transaction.

6.4 Weak Consistency

Eventual consistency is only one form of weak consistency. A number of different tradeoffs between partition tolerance and consistency exist in literature.

Definition 6.27 (Monotonic Read Consistency)**.** *If a node u has seen a particular value of an object, any subsequent accesses of u will never return any older values.*

Remarks:

- Users are annoyed if they receive a notification about a comment on an online social network, but then are unable to reply because the web interface does not show the same notification yet. In this case the notification acts as the first read operation, while looking up the comment on the web interface is the second read operation.

Definition 6.28 (Monotonic Write Consistency)**.** *A write operation by a node on a data item is completed before any successive write operation by the same node (i.e. system guarantees to serialize writes by the same node).*

Remarks:

- The ATM must replay all operations in order, otherwise it might happen that an earlier operation overwrites the result of a later operation, resulting in an inconsistent final state.

Definition 6.29 (Read-Your-Write Consistency)**.** *After a node u has updated a data item, any later reads from node u will never see an older value.*

Definition 6.30 (Causal Relation)**.** *The following pairs of operations are said to be causally related:*

- *Two writes by the same node to different variables.*

- *A read followed by a write of the same node.*

- *A read that returns the value of a write from any node.*
- *Two operations that are transitively related according to the above conditions.*

Definition 6.31 (Causal Consistency)**.** *A system provides causal consistency if operations that potentially are causally related are seen by every node of the system in the same order. Concurrent writes are not causally related, and may be seen in different orders by different nodes.*

Chapter Notes

The CAP theorem was first introduced by Fox and Brewer [FB99], although it is commonly attributed to a talk by Eric Brewer [Bre00]. It was later proven by Gilbert and Lynch [GL02] for the asynchronous model. Gilbert and Lynch also showed how to relax the consistency requirement in a partially synchronous system to achieve availability and partition tolerance.

Bitcoin was introduced in late 2008 by Satoshi Nakamoto [Nak08]. Nakamoto is thought to be a pseudonym used by either a single person or a group of persons; it is still unknown who invented Bitcoin, giving rise to speculation and conspiracy theories. Among the plausible theories are noted cryptographers Nick Szabo [Big13] and Hal Finney [Gre14]. The first Bitcoin client was published shortly after the paper and the first block was mined on January 3, 2009. The genesis block contained the headline of the release date's The Times issue "*The Times 03/Jan/2009 Chancellor on brink of second bailout for banks*", which serves as proof that the genesis block has been indeed mined on that date, and that no one had mined before that date. The quote in the genesis block is also thought to be an ideological hint: Bitcoin was created in a climate of financial crysis, induced by rampant manipulation by the banking sector, and Bitcoin quickly grew in popularity in anarchic and libertarian circles. The original client is nowadays maintained by a group of independent core developers and remains the most used client in the Bitcoin network.

Central to Bitcoin is the resolution of conflicts due to doublespends, which is solved by waiting for transactions to be included in the blockchain. This however introduces large delays for the confirmation of payments which are undesirable in some scenarios in which an immediate confirmation is required. Karame et al. [KAC12] show that accepting unconfirmed transactions leads to a non-negligible probability of being defrauded as a result of a doublespending attack. This is facilitated by *information eclipsing* [DW13], i.e., that nodes do not forward conflicting transactions, hence the victim does not see both transactions of the doublespend. Bamert et al. [BDE$^+$13] showed that the odds of detecting a doublespending attack in real-time can be improved by connecting to a large sample of nodes and tracing the propagation of transactions in the network.

Bitcoin does not scale very well due to its reliance on confirmations in the blockchain. A copy of the entire transaction history is stored on every node in order to bootstrap joining nodes, which have to reconstruct the transaction history from the genesis block. Simple micropayment channels were introduced by Hearn and Spilman [HS12] and may be used to bundle multiple transfers between two parties but they are limited to transferring the funds locked into the channel once. Recently Duplex Micropayment Channels [DW15] and the Lightning Network [PD15] have been proposed to build bidirectional micropayment channels in which the funds can be transferred back and forth an arbitrary number of times, greatly increasing the flexibility of Bitcoin transfers and enabling a number of features, such as micropayments and routing payments between any two endpoints.

This chapter was written in collaboration with Christian Decker.

Bibliography

[BDE+13] Tobias Bamert, Christian Decker, Lennart Elsen, Samuel Welten, and Roger Wattenhofer. Have a snack, pay with bitcoin. In *IEEE Internation Conference on Peer-to-Peer Computing (P2P), Trento, Italy*, 2013.

[Big13] John Biggs. Who is the real satoshi nakamoto? one researcher may have found the answer. http://on.tcrn.ch/l/R0vA, 2013.

[Bre00] Eric A. Brewer. Towards robust distributed systems. In *Symposium on Principles of Distributed Computing (PODC)*. ACM, 2000.

[DW13] Christian Decker and Roger Wattenhofer. Information propagation in the bitcoin network. In *IEEE International Conference on Peer-to-Peer Computing (P2P), Trento, Italy*, September 2013.

[DW15] Christian Decker and Roger Wattenhofer. A Fast and Scalable Payment Network with Bitcoin Duplex Micropayment Channels. In *Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, 2015.

[FB99] Armando Fox and Eric Brewer. Harvest, yield, and scalable tolerant systems. In *Hot Topics in Operating Systems*. IEEE, 1999.

[GL02] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 2002.

[Gre14] Andy Greenberg. Nakamoto's neighbor: My hunt for bitcoin's creator led to a paralyzed crypto genius. http://onforb.es/1rvyecq, 2014.

[HS12] Mike Hearn and Jeremy Spilman. Contract: Rapidly adjusting micro-payments. https://en.bitcoin.it/wiki/Contract, 2012. Last accessed on November 11, 2015.

[KAC12] G.O. Karame, E. Androulaki, and S. Capkun. Two Bitcoins at the Price of One? Double-Spending Attacks on Fast Payments in Bitcoin. In *Conference on Computer and Communication Security*, 2012.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf, 2008.

[PD15] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network. 2015.