TEE-Based Distributed Watchtowers for Fraud Protection in the Lightning Network

Marc Leinweber[1], Matthias Grundmann[1], Leonard Schönborn[2], and Hannes Hartenstein[1]

[1] Karlsruhe Institute of Technology, Karlsruhe, Germany
firstname.lastname@kit.edu
[2] Karlsruhe Institute of Technology, Karlsruhe, Germany
leonard.schoenborn@student.kit.edu

Abstract. The Lightning Network is a payment channel network built on top of the cryptocurrency Bitcoin. It allows Bitcoin to scale by performing transactions off-chain to reduce load on the blockchain. Malicious payment channel participants can try to commit fraud by closing channels with outdated balances. The Lightning Network allows resolving this dispute on the blockchain. However, this mechanism forces the channels' participants to watch the blockchain in regular intervals. It has been proposed to offload this monitoring duty to a third party, called a watchtower. However, existing approaches for watchtowers do not scale as they have storage requirements linear in the number of updates in a channel. In this work, we propose TEE Guard, a new architecture for watchtowers that leverages the features of Trusted Execution Environments to build watchtowers that require only constant memory and are thus able to scale. We show that TEE Guard is deployable because it can run with the existing Bitcoin and Lightning Network protocols. We also show that it is economically viable for a third party to provide watchtower services. As a watchtower needs to be trusted to be watching the blockchain, we also introduce a mechanism that allows customers to verify that a watchtower has been running continuously.

1 Introduction

On top of the Bitcoin protocol [19], the Lightning Network [21] has emerged as a second layer solution to reduce the load of transactions on the Bitcoin blockchain by implementing a payment channel network. The Lightning Network consists of interconnected bilateral payment channels. Each payment channel encodes the distribution of a fixed amount of bitcoins between its two participants. At any time, either party can close the channel by publishing a commitment transaction to the Bitcoin blockchain that pays each party the amount of coins they own in the channel. However, in payment channels, it is possible to commit a *fraud* by publishing an outdated commitment transaction to the blockchain. A fraud is profitable when the outdated commitment transaction assigns more coins to the cheating party than the most current channel state. To mitigate the fraud, the cheated party has the opportunity to react and invalidate the outdated transaction during a predefined time span.

Channels are usually configured to allow users 24 hours (144 blocks) to react to such transactions with a revocation transaction that reverts the fraudulent commitment transaction. This means that users have to be online and search the blockchain at least once every 24 hours to defend against fraud attempts. To avoid the obligatory activity, to allow for longer phases of inactivity and to enable the development of lightweight nodes, watchtower (or custodian) services have already been proposed. Dryja [9] proposed in 2016 a monitor that searches new blocks on the blockchain for commitment transactions concerning the channels of its users and publishes, in the case of a fraud, revocation transactions on the user's behalf. The monitor needs to store a revocation transaction for every state of the channel which obviates the need for sharing of secrets but introduces linear memory consumption and is therefore limited in its scalability. Dryja's approach was followed by several other monitoring proposals [20,16,4], whose strengths and limitations we discuss in Section 2.

To run such watchtower services economically, their ability to *scale* to many users is of vital importance. To keep the watchtower's provider's cost economical, the computation time and storage requirements need to be low per monitored channel and, for channels being open for a long time, the storage requirements need to be *constant with respect to the number of updates in the channel*. In this work, we present TEE Guard, a watchtower solution for the Lightning Network based on Trusted Execution Environments (TEEs) that is scalable due to its low and constant storage requirements. With TEE Guard, users of the Lightning Network can create a watchtower for their channels that is hosted by a so-called platform provider. To improve reliability, such a user can become a customer of multiple independent platforms and interconnect the watchtowers running at these different platforms. For initialisation of the watchtower service, the customer transmits, in a secure channel and unseen by the platform provider, a public key hash where revocation transactions will be paid to and references to the Lightning channels to monitor. With each channel update, the customer has to transmit the required secrets to invalidate the now outdated commitment to their watchtowers. As the watchtower provider is not trusted by its customers and might even collude with a customer's channel partner, all *secrets need to be kept private* by the watchtower service. By storing them inside the TEE, we ensure that they cannot be read by the provider. The key derivation system of the Lightning Network makes it possible to store the secrets for all past updates of a channel using constant space. The watchtower provider may not only try to learn secrets from its customers but also try to cheat them by stopping the watchtower service to save operational costs. It would be too late if a customer would notice that a watchtower stopped their service only after the watchtower did not react to an outdated commitment transaction. To allow a watchtower's customers to *verify the watchtower's availability*, TEE Guard enables the customers to interconnect multiple watchtowers forming a watchtower network and the watchtowers will monitor each other and create logs about the others' availability. These logs are signed and, in conjunction with the remote attestation feature of TEEs, the customer can thus verify that the provider has been running the watchtower continuously.

In sum, TEEs allow us to design an architecture, on top of the currently deployed Bitcoin and Lightning protocols, that is (1) scalable, (2) economically viable, and (3) in its redundancy tunable and verifiable by the user. The remainder of this work is structured as follows. The background on payment channels in general and the Lightning Network in particular as well as a definition and an overview on TEEs is given in Section 2. Furthermore, related work is discussed. The TEE Guard architecture is presented in detail in Section 3, quantitatively evaluated in Section 4 and its security is assessed in Section 5. We conclude the paper and give an outlook on future work with Section 6.

2 Background and Related Work

2.1 Lightning Network

The Lightning Network consists of bilateral payment channels that encode the distribution of bitcoins between their two participants. To open a new channel, two parties agree on an initial distribution, exchange `payment_basepoint`s, and the funder of the bitcoins, who is one of the two participants, creates, signs, and publishes a transaction that spends the bitcoins to a 2-of-2 multi-signature output. The current distribution is managed in a commitment transaction which, in turn, is not published until the channel is closed. Hence, with each change of the channel's balance, a new commitment transaction is negotiated. Each commitment transaction uses keys derived from a new `per_commitment_secret` and includes a counter (called `commitment_number`) that is incremented for each update in the channel. After a new commitment transaction has been created, both parties exchange the `per_commitment_secret`s for the outdated transaction. In case one of the channel's participants publishes an outdated commitment transaction, the other party can spend the transaction's outputs using the corresponding `per_commitment_secret` and its own `revocation_basepoint_secret`. To allow for the revocation transaction to be published, the commitment transaction's outputs can only be spent by the other party after a fixed time span called `to_self_delay` (measured in number of blocks). This mechanism forces the channel's participants to check the Bitcoin blockchain in regular intervals for invalid commitments and to store all old `per_commitment_secret`s. The secrets can, however, be stored efficiently at constant size using a key derivation mechanism [1].

In the Lightning Network, payment channels are interconnected and the resulting network can be used for transactions over multiple hops using Hashed Time-Locked Contracts (HTLCs). The HTLC construction ensures that either all participants are able to execute the payment (using their knowledge of a secret x) or none of them are. A conditional output is added to each commitment transaction along the payment's path that includes $y = h(x)$ (with h being a collision-resistant hash function) in the condition. The output can be redeemed by the receiver by publishing x or becomes invalid after a timeout. If a commitment transaction with an HTLC output is published on chain, it can either be spent by the receiver with an HTLC success transaction providing x or after the specified timeout with an HTLC timeout transaction by the sender.

2.2 Trusted Execution Environments (TEEs)

By using trustworthy features in hardware and software that isolate processes from other user and system processes, an environment (called enclave) is established that allows the trusted execution of code. Such TEEs are implemented, for example, by Intel in the form of *Security Guard Extensions (SGX)* [17,2] or by AMD's *Memory Encryption Technology* [12]. Besides these proprietary approaches, Keystone [13] is an open-source framework for implementing TEEs based on the open hardware architecture RISC-V. These implementations have different design goals and features. Current proposals for systems using TEEs are typically based on SGX. SGX protects the confidentiality of the application's data and the integrity of code and data against malicious applications and a malicious operating system. The only trusted component is the Intel CPU whereby the required trust is reduced to the hardware designer and manufacturer.

For TEE Guard, we require a TEE to provide *isolated execution*, *protected memory*, and *remote attestation*. With these features, code is protected from manipulation and runtime data is guaranteed to be of integrity, confidential and fresh. The remote attestation enables the remote verification of the platform's identity (CPU secrets) and the code's identity (its instruction order). The TEE can derive encryption keys from its own identity and the code's identity to store data on disk.

2.3 Blockchain and TEEs

The work presented in this paper connects the research fields of blockchain and TEEs. Previous work in this intersection has been done with different focuses. TEEs have been used to implement alternatives to proof-of-work as used in Bitcoin, e.g. proof-of-luck [18] or proof-of-elapsed-time [11]. They have also been used to implement a trusted exchange [5], second layer architectures (e.g. Teechain [14] and Banklaves [10]), and an extension to Bitcoin to run arbitrary smart contracts [7]. In this work, we present TEE Guard, which is, to the best of our knowledge, a new approach to use TEEs in the world of cryptocurrencies by leveraging their features to implement a watchtower service for payment channel networks.

2.4 Watchtowers

Watchtowers have already been proposed very early in the development of the Lightning Network by Dryja [9]. In his approach, a customer creates a revocation transaction for every old commitment transaction, encrypts it with the old commitment transaction's ID, and sends it to the watchtower. When an old commitment transaction is published, the watchtower uses the old commitment transaction's ID to decrypt and then publish the revocation transaction. The approach preserves the channel's privacy and the watchtower does not learn any secrets that could be abused, but requires the watchtower to store $O(n)$ transactions where n is the number of updates in the channel. To incentivise the watchtower to perform this service, the customer can add an additional output to the revocation transaction that pays the watchtower provider a fee. However, this leaves the watchtower provider unpaid in case no fraud happens in a channel and in case of a fraud only the first watchtower to publish the revocation transaction earns the fee.

Osuntokun [20] proposed an updated construction of channels for the Lightning Network that requires the introduction of a new opcode to the Bitcoin protocol, but allows for constant storage for channel participants and watchtowers. However, the opcode has not been implemented since and, thus, the approach is not deployable yet.

Later, Pisa [16] was proposed, which is an approach for watchtowers (called custodians) which is constructed to work with state channels on top of Ethereum instead of payment channels as used in the Lightning Network. As the state channel construction uses a smart contract that can compare states by their commitment numbers, the custodian only needs constant sized memory. However, in the Lightning Network, as currently deployed, the state is replaced by revocation [21], which requires a specific revocation transaction for each new state. Storing all revocation transactions requires storage linear to the number of payments in a channel. As our approach uses TEEs, we can outsource the secrets needed to generate the revocation transactions which reduces the required storage of the watchtower to a constant size. Pisa requires the custodian to provide a security deposit which is being destroyed in case the custodian fails to settle a dispute. To prevent the custodian from colluding with a channel partner, the security deposit has to be high enough that the custodian would not profit from such a collusion. This necessitates high security deposits by the custodian and the required high investment makes it somewhat unrealistic that someone would assume the role of such a custodian. In our approach, we do not need such a security deposit. In return, we require that multiple custodians (or watchtowers) are used and we assume that they do not collude. We assume that it is more realistic to find non-colluding service providers for watchtowers (at least one correctly working watchtower is enough to protect the channel) than finding custodians that are willing to place a high amount of capital as security deposits.

Decker et al. proposed eltoo [8] which is a payment channel construction for Bitcoin with a different approach than the currently deployed Lightning Network. Instead of replacing old states by revocation, it adds a counter to each transaction and uses a construction that allows old transactions to be spent by transactions with higher counter values. This requires both parties to only store the transactions with the latest state and is thus also a solution to the linear storage problem. However, the concept uses so called *floating transactions* which require an opcode added to the Bitcoin protocol which has not been included yet. On a payment channel network using eltoo, watchtowers would require only constant sized storage per channel. Yet, watchtowers would be necessary to watch the blockchain for outdated commitment transactions. Our approach could still be a valuable addition on such a network because TEE Guard introduces redundancy and the mutual monitoring of watchtowers allows verification of the watchtower providers' availability.

In [4], Avarikioti et al. present DCWC, a protocol for a network of watchtowers which forward revocation transactions to other watchtowers and publish them during dispute. They also present a modified protocol DCWC*, which is implementable on the Lightning Network in contrast to DCWC. Watchtowers are incentivised to participate in the protocol by receiving a part of a channel's funds in case of dispute. The issue here is that watchtowers are not paid in case no dispute arises and therefore they might not be sufficiently incentivised. Our approach incentivises watchtowers by users paying the watchtower providers a fixed rate per month. DCWC* requires the watchtowers to store $O(n)$ messages, where n is the number of channel updates. Using TEEs to store the revocation secrets, we limit the watchtower's storage to a constant size, which makes it possible to watch a high number of channels for an unlimited time using a single machine.

Recently, with Brick [3], a new state channel construction has been proposed. Brick replaces the on-chain dispute process by a mediator committee of third parties. Therefore, watchtowers are no longer necessary in Brick. However, the channel partners have to agree on the committee members and trust that at least a third of them will be honest.

3 TEE Guard Architecture

TEE Guard implements a watchtower solution for the Lightning Network in a TEE. The TEE assures untampered and confidential execution of the watchtower code and enables remote attestation of the TEE's platform identity and the executed code identity. Watchtowers are distributed to increase reliability and implement mutual monitoring.

A Lightning Network user can become a *customer* of one or more independent TEE Guard *platform providers*. Each platform provides watchtower *instances* for an arbitrary number of customers, who send updates to the watchtower after each update in their channels. The watchtowers monitor the blockchain for outdated commitment transactions and react to such transactions with revocation transactions. By interconnecting their watchtower instances, customers can construct a distributed system that fits their needs. The instances will then monitor each other's state to increase the reliability and to detect potentially malicious or unreliable providers. On the detection of outdated information, correct states will be exchanged. The activity monitoring results are logged and provided to the customer. A provider can be, for example, a company, organisation or a private person.

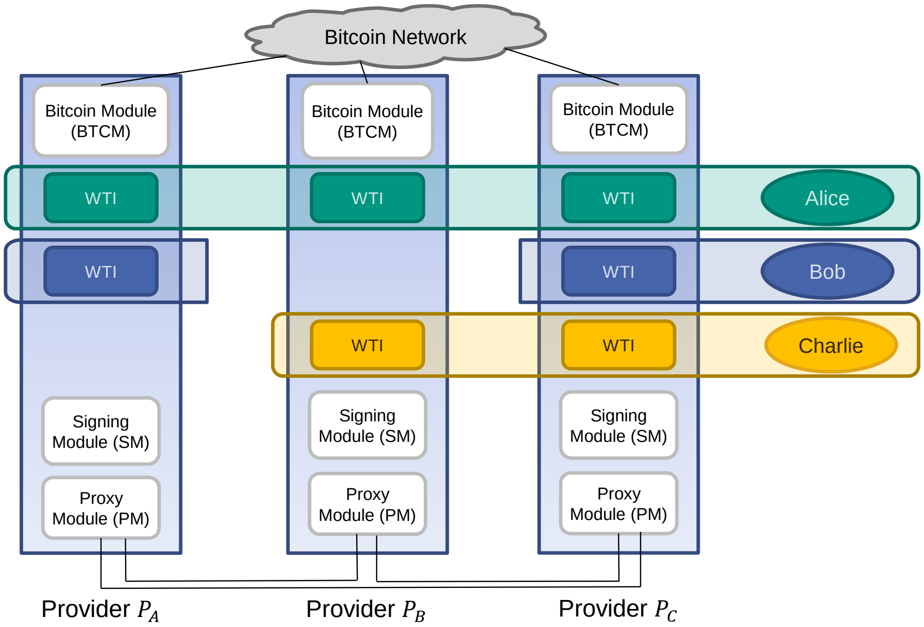

Fig. 1. Abstract overview of the TEE Guard interconnected watchtowers showing the TEE Guard software running at three watchtower providers with their different modules. Each provider hosts different watchtower instances (WTI) that are owned by a specific customer. In this scenario three users (Alice, Bob, and Charlie) operate a network of interconnected watchtowers hosted at two or three providers respectively.

Figure 1 illustrates an example of the distributed system with three provider platforms P_A, P_B and P_C and three customers Alice, Bob and Charlie. The figure only shows the TEE Guard components and abstracts away the enclave's surrounding runtime environment. Each provider platform is connected to the Bitcoin peer-to-peer network and receives the blockchain. Because each platform has a customer that is also customer of one of the two others, all platforms are interconnected. The operating principle and the rationale behind it is discussed in detail in the remainder of this section.

3.1 Provider Platform

A provider platform is implemented in one enclave and divided into modules.

Watchtower Module (WTM) The WTM implements the watchtower functionality: monitoring of channel-related transactions, creation of revocation transactions and monitoring of connected watchtowers. An instance of the WTM (called WTI) is created for each customer.

Bitcoin Module (BTCM) The BTCM encapsulates the communication with the Bitcoin blockchain. It manages a local copy of the last blocks of the blockchain and provides them to the WTIs. The number of blocks to store depends on the monitored channel with the longest time span between publication and validity of its commitment transaction (given by the channel's `to_self_delay`). When a WTI needs to publish a revocation transaction, this is done by the BTCM, too.

Proxy Module (PM) The PM implements the communication between provider systems. The motivation behind the PM is to hide the traffic patterns of individual WTIs. Thus, providers cannot identify which WTI is communicating with another provider and they cannot deduce the distribution of the customers among the providers.

Signing Module (SM) To prove the presence or absence of the provider's service over time, the WTIs write encrypted and integrity-protected logs of the monitoring process to secondary storage. The logs are encrypted to hide the connection information of a customer from the provider and to prevent manipulation of the information. A customer can request the logs to verify the providers' activity. To unambiguously associate the logs with a provider, the SM signs the logs before they are sent to the customer. To do so, the SM creates an asymmetric key pair on the first start of the provider platform.

3.2 Setup

The customer chooses a number $n \in \mathbb{N}$ of providers, sets up the n instances ("Instance Setup") and connects them ("Network Setup").

Provider Platform Setup To allow verifiable and reproducible builds, which is essential for meaningful remote attestation, TEE Guard has to be open-source software. All a provider has to do is to download, build, and run the software. The provider is flexible in the choice of the executed software and is free to alter the code. The remote attestation feature ensures that the customer cannot be betrayed by running code they do not agree with.

Instance Setup The provider creates a new WTI and gives access to it to the customer. During a remote attestation, the customer verifies the provider platform's identity and establishes a secure channel to the WTI. Once the customer has completed the verification, the WTI cannot be accessed by the provider anymore. This is enforced by the implementation which is protected by the TEE. The customer then has to provide the `revocation_payout_address`, a Bitcoin address for revocation payouts, that also serves as a customer identifier. Additionally, a list of trusted TEE Guard code identities for the network setup can be transmitted. During the operation of the WTI, the customer can add and remove channels to monitor.

Network Setup As mentioned before, TEE Guard is designed to be a distributed system. The WTIs of a customer can be connected to increase reliability and enable mutual monitoring. To connect two WTIs A and B on platforms P_A and P_B, the customer tells A to establish a connection to another provider platform's proxy module. In order to (re)establish a new or broken (down) connection, one of the platforms has to be publicly reachable. Firstly, P_A's proxy module checks if it already established a connection to P_B. If not, both proxy modules perform a remote attestation and store the identity of the remote platform. A compares P_B's identity with its list of trusted TEE Guard code identities. If the list does not contain P_B's identity, A cancels the connection process. The proxy module of P_B checks if the platform hosts a WTI B with the same `revocation_payout_address`. Next, B checks its list of trusted identities against P_A's identity and, if P_A's identity is not trusted, B cancels the connection process. If both WTIs A and B agree on establishing a connection, they tell their platform's proxy modules to route upcoming messages between them. Finally, the proxy module of A adds P_B to A's list of connected platforms and vice versa.

3.3 Operation

A channel participant can attempt a fraud by issuing an outdated commitment transaction tx_F with $\texttt{commitment\_number}(tx_\mathrm{F}) < \texttt{commitment\_number}(tx_\mathrm{C})$ of the most current commitment transaction tx_C. If no revocation transaction is issued within the time span defined by the channel's `to_self_delay` parameter, the fraud can claim the stolen funds.

Thus, to detect a fraud, the watchtower needs to scan the last `to_self_delay` blocks of the blockchain because they may contain a revocable outdated commitment transaction. To identify and revoke outdated transactions, the most current `commitment_number` of a channel, the information to find channel-related commitment transactions on the blockchain, and the secrets to sign a valid revocation transaction are needed (see Table 1).

In sum, the customer has to contact the watchtowers on each channel transaction and the watchtower has to get active with each new block added to the blockchain.

Channel Updates Most of the necessary parameters are fixed for the channel's life time and need to be transmitted during the setup of the channel monitoring. For each channel update the customer has to contact its WTIs and provide the `per_commitment_secret` for the now outdated transaction and the new `commitment_number`. According to BOLT#3 [1] the `per_commitment_secrets` can be stored efficiently by storing at most 49 (value, index) pairs at any given point in time, which can be used to derive the keys for outdated commitment transactions.

Blockchain Updates The BTCM of each provider platform is connected to the Bitcoin network and parses incoming messages. On the reception of a new block, the BTCM verifies the Merkle hash tree, compares the transaction inputs to the

Table 1. Information required for watchtower operation that has to be stored for a channel by each watchtower

Data	Purpose
`funding_txid`	identification of channel-related commitment transactions
both participants' `payment_basepoints`	decryption of `commitment_numbers` in commitment transactions
most current `commitment_number`	identification of outdated commitment transactions
`revocation_basepoint_secret`, values to derive `per_commitment_secrets`	revocation key derivation

list of monitored channels of the platform's WTIs and stores the block together with a signature of it on the untrusted secondary storage of the provider's system. On future accesses, there is no need to verify the block again but only the signature. If a transaction input in the block is on the list of monitored channels[3], the corresponding WTI is informed. The WTI reconstructs the `commitment_number` using the `payment_basepoints` and compares it to the stored most current `commitment_number`.

In case the commitment transaction is outdated, the issuer of the transaction has to be identified. If the WTI succeeds in calculating the revocation key corresponding to the outdated commitment transaction, the transaction was not issued by the customer but by the customer's channel partner and the transaction is identified as fraudulent. The WTI constructs and publishes a revocation transaction that spends the fraud's output to the `revocation_payout_address`. If the outdated commitment transaction contains HTLC outputs, these outputs are used as inputs for the revocation transaction, too. However, the revocation transaction issued by the WTI may be preceded by an HTLC success or HTLC timeout transaction. If so, the revocation transaction would be a double spend of this transaction because both spend the same outputs. Thus, the watchtower has to monitor the blockchain until the revocation transaction becomes valid and, in case an HTLC transaction was issued by the fraud, the revocation transaction has to be reissued using the HTLC transaction's output instead of the commitment transaction's output. Channels can be removed from the list of monitored channels if a block old enough contains the revocation or a closing transaction.

With each new block, the WTI contacts the connected WTIs on other provider platforms with a `New Block` message. This message contains the list of monitored channels, the most currently known commitment number and the current blockchain height. If the receiving WTI identifies outdated channel information (missing channels or outdated commitment numbers), it answers with a `New Block Response` message. The incoming `New Block` messages are logged.

[3] More specifically, this means that the transaction id of the input equals a `funding_txid` contained in the list of monitored channels.

Rollback Protection In case the freshness of the state information cannot be ensured, the temporal correctness of code execution is not guaranteed. As long as a TEE system is running, it can assure the freshness of the main memory data. This does not hold for reboots or secondary storage. Thus, state has to be identified with respect to time.

A WTI compares the correctness of its current channel information with each blockchain update. The list of connected provider platforms, however, is not transmitted. When a provider platform is rebooted, it stores its current state to the secondary storage (encrypted and signed). During initialisation, this stored state is restored to main memory. If a provider would manipulate the information on the secondary storage by replaying a previously stored image, they could remove, however not fine-grained, information on connected provider platforms. Hardware monotonic counters in current TEE implementations have been shown to be of reduced usability [15]. We use a distributed `state_counter` to circumvent hardware monotonic counters. Each time a channel is added or removed or a provider platform is connected or disconnected, the WTI where the action was taken updates the counter and broadcasts the change to the connected WTIs. When the provider platform is restarted, the WTIs ask the known connected WTIs for their current state and update if necessary. The WTIs log received and sent state information.

Verification of Availability To verify that a provider fulfills the required availability, a user can collect and analyse the providers' logs. If the logs of watchtower A show that a message for a new block from watchtower B has been received after receiving a newer block from the blockchain, this indicates that A or B was not online for some time. Comparing the logs of all watchtowers allows distinguishing whether A or B was offline.

4 Quantitative Evaluation

In this section, we analyse TEE Guard assuming an implementation with Intel SGX. We analyse the reliability of our approach using a simulation and its scalability by analysing the computing and storage requirements. Using the results of the analysis, we estimate the cost for a watchtower provider and find that it is economically viable to provide this service for a cost below one USD cent per channel per month, provided that enough customers per provider platform use the service.

4.1 Reliability

The reliability solely rests on the availability of the provider platforms. If a system is ready to work and not hindered in its execution, it will scan the blockchain and emit revocation transactions if necessary. Availability is disrupted by outages of any kind or malicious providers. Both kinds of disruption can be proven by the logging utility, which will be discussed in Section 5.2.

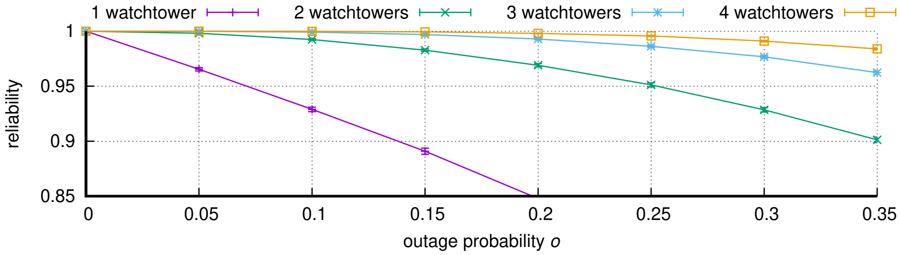

Fig. 2. Reliability study with $u = 5$ users, mean channel lifetime $c = 30$ d, mean transaction interval $t = 36$ h and simulated time $s = 20$ y set fix. The interval delimiters show the 99.9 % Student's t-distribution confidence interval.

To understand the implications of outages not only for TEE Guard but distributed watchtower scenarios in general, we conducted a simulation study using a discrete-event simulator written in Java. A standard failure model cannot be applied and analytically analysed, because not only outages of watchtowers are relevant but also their state. The simulation model is simplified in contrast to Section 3.1 and consists of users, watchtowers (WTIs), channels, and a simplified blockchain. Channels and watchtowers are assigned to a user. Provider platforms are not modelled; each watchtower runs independently of the other watchtowers. Modelling the provider platforms would cause correlations of WTI outages because a provider platform outage causes the outage of all WTIs running on it. The blocks of the blockchain are only containing references to channels that are being closed in this block. The opening of new channels is not modelled explicitly. The parameters are the number of users u, the number of watchtowers per user w, the outage probability of a watchtower o, the mean channel life time c, the mean transaction interval for a channel transaction t and the simulated time s. The parameters c and t are mean values for exponential distributions; a new block is emitted every 10 minutes on average, following an exponential distribution. To get a worst case estimation, outage periods of significant length are needed. The outage length is set fix to 48 hours. Thus, the watchtowers fail independently every 48 hours for a period of 48 hours with probability o. On startup, the users and their watchtowers are created and for each user one channel is opened. The channel transaction events as well as the channel close events are scheduled. All channels are closed with an outdated commitment number chosen uniformly at random and a new channel is immediately created for the user. After 144 blocks it is evaluated, whether the fraud was detected. The watchtowers exchange state information with every block update as discussed in Section 3.3.

We conducted the simulation with $u = 5$ users, mean channel lifetime $c = 30$ d, mean transaction interval $t = 36$ h and simulated time $s = 20$ y set fix. The number of users might seem low, but an increased number of users does not affect the reliability, because each user has their own watchtowers which fail independently of others. Increasing the number of users u or the simulated time s only reduces the confidence intervals at the cost of a higher simulation time.

A reduced channel transaction interval causes better reliability results because it increases the probability that at least one watchtower learns the most current commitment number before the channel partner tries to close the channel. We have chosen channels to live a month on average and to have a new transaction every 36 hours on average to model the behaviour of a super market and its customer. The parameters w and o were varied in a range of 1 to 10 in steps of 1 resp. 0.00 to 1.00 in steps of 0.05. Each parameter combination was simulated 120 times with different seeds. Figure 2 shows the reliability for 1 to 4 watchtowers with an outage probability range from 0.00 to 0.35. The simulation results show that, using three watchtower instances, the mean reliability is over 96% for an unrealistically high outage probability of 35 %. The following analytical evaluation is based on these results and three watchtowers per user are assumed.

4.2 Scalability

Memory For the provider platforms, we have to distinguish between the enclave memory that resides in main memory when the enclave is running and persistent memory that is used to store data. In Intel SGX, the enclave memory is limited to about 97 MB, but the persistent storage can be as high as the disk space available in the hosting system. In Table 2, we show which data needs to be stored in an enclave's memory. Summing it up leads to the following memory requirements, where n_{U} is the number of customers, n_{C} the number of monitored channels, n_{E} the number of connected provider platforms per WTI, and n_{P} the total number of connected provider platforms:

$$\begin{aligned} M(n_{\mathrm{U}}, n_{\mathrm{C}}, n_{\mathrm{E}}, n_{\mathrm{P}}) = {} & n_{\mathrm{U}} \cdot 125 \text{ byte} + n_{\mathrm{C}} \cdot 1996 \text{ byte} + n_{\mathrm{P}} \cdot 81 \text{ byte} \\ & + n_{\mathrm{E}} \cdot n_{\mathrm{U}} \cdot 4 \text{ byte} + 36 \text{ byte} \end{aligned}$$

The provider platform also needs runtime memory for the verification of new blocks. As they can be verified consecutively, only one block needs to be stored at a time. We assume 2.5 MB memory usage for block verification which is more than enough for the currently biggest block of 2.3 MB. We estimate the memory requirement of the enclave's code to be limited by 4 kB. Assuming $n_{\mathrm{P}} = 100$ different connected provider platforms, $n_{\mathrm{E}} = 3$ connected provider platforms per customer, and 5 channels per customer, the enclave's memory suffices for monitoring the channels of more than 9,500 users.

Disk space The received blocks need to be stored on disk. However, it suffices to store the last 144 + 6 blocks (the highest value of `to_self_delay` + the blocks needed for a block to be considered confirmed). Using the values from above ($n_{\mathrm{C}} = 45,000$, $n_{\mathrm{E}} = 3$), the logs take a maximum of 3.5 MB per block. For six months this would accumulate to 90 GB. However, this can be reduced massively by using a non-trivial storage format. To save storage cost, the watchtower providers can agree with their customers on a time after which they are allowed to remove old logging entries.

Table 2. Memory requirements of a provider platform

Data	Type and Size
key material for communication between WTI and customer	ECDH public/private key, 65 byte per customer, symmetric key 32 byte per customer
`revocation_payout_address`es	32 byte hash per customer
blockchain height	4 byte integer
most current block hash	32 byte hash
state counter	per customer: 4 byte integer
monitored channels	per channel: 32 byte `funding_txid`, 6 byte `commitment_number`, both participants' `payment_basepoint`s, $2 \cdot 32$ byte 32 byte `revocation_basepoint_secret`, $49 \cdot 38$ byte to derive `per_commitment_secret`s
connected provider platforms	per provider platform: 16 byte IP address, ECDH public key 33 byte, symmetric key 32 byte
connected provider platforms per WTI	list of 4 byte indizes

Bandwidth When connecting provider platforms, messages are exchanged during remote attestation. As this is only done once for establishing a connection, we can ignore it for estimating the required bandwidth for the continuous operation of the system. We need to download each new block that is added to the blockchain, which results in an incoming traffic of about 10 Mbit/10 min = 17 kBit/s. Assuming that, in the unrealistic worst case, a revocation transaction has to be created for every transaction in a block, we need the same outgoing bandwidth for publishing the revocation transactions. As new blocks are mined every 10 minutes on average in Bitcoin, the messages sent between watchtowers after receiving a new block are so rare that they can be neglected for estimating the bandwidth. For every update to a channel, a user has to send 74 bytes to the watchtowers. Therefore, we can estimate the required bandwidth for channel updates to $\beta \cdot n_{\mathrm{c}} \cdot 74$ byte, where β is the update rate per channel and n_{c} is the number of channels a user has. For a high channel update rate of $\beta = 10/s$ for five channels per user, this results in a bandwidth requirement of 28.9 kBit/s. With 9,000 users on a system updating their five channels with such a high rate, this results in an incoming bandwidth of 254 MBit/s, which is still realistic for a professionally hosted system.

Computation time Channel updates do not require much computation and searching the transactions inside a block could take a few minutes without a problem, because new blocks arrive only every 10 minutes on average. Therefore, the computation time is not a bottleneck.

4.3 Deployability

Estimating that 100 GB of disk space will be more than enough for the persistent storage, we used the Amazon AWS Calculator[4] to estimate the operational cost of running a watchtower system. We require an SGX-enabled CPU, 256 GB outgoing traffic per month and 100 GB of disk space. This results in a monthly cost of about 80 USD. Assuming a system working to capacity with 9,000 users and 5 channels per user, the system is viable if each user pays at least 0.18 cents per channel per month. Assuming only 1000 users and 5 channels per user, each user would have to pay at least 1.6 cents per channel per month to make the system viable. As self hosted systems might be cheaper than using AWS, this shows that providers are incentivised to host watchtower systems and offer them to their customers for reasonable prices. Payment channels can be used for the monthly payments to keep transaction fees low.

5 Security Assessment

5.1 Confidentiality

Based on the assumptions of TEEs, all secrets that are stored in the watchtower are protected from observation by any third party. The secrets are stored securely in the enclave's memory resp. encrypted on hard disk. To send secrets to the provider platform, customers use an encrypted authenticated channel which is established during remote attestation. This ensures that secrets cannot be leaked when uploading them to the provider platform and that they are stored only at attested and trusted platforms.

5.2 Verification of Availability

To fulfill the requirements considering verification of availability, it is necessary to recognise the outage of enclaves and prove it to others. TEE Guard creates logs of the availability of connected enclaves. To establish trust in these logs, the messages are signed with the providers signing key. The logs are stored in an encrypted and integrity-protected form. To disguise a time of unavailability in the past, a provider would have to manipulate the logs stored at their competitors' instances. The only way to manipulate the log in order to make a competitor's instance look unresponsive is by suppressing messages sent from and to that instance. However, this also leads to the provider's own instance being seen as unresponsive from the competitor's instance, which is not in the provider's interest. If the provider would delete the logs before the user retrieves them, the manipulation would be detected by retrieving the logs from the connected platforms. The availability of other enclaves is verified using logs of all received `New Block` messages which only allows determining the availability once per block interval. Since a WTI searches the transactions inside a block for fraud before sending the `New Block` message, this suffices to verify the availability required for a watchtower.

[4] https://calculator.aws/#/configureEc2, June 2019

5.3 Attacks on TEEs

It has been shown that practical implementations of TEEs are not free of bugs [6] and experience tells us that any implementation might have flaws and thus potential bugs have to be considered. Therefore, we lastly analyse how the security of TEE Guard is affected when an attacker breaks the security assumptions of TEEs. If an attacker extracts the hardware secrets from the enclave, they might be able to impersonate an enclave and fool customers into sending them the data instead of a legitimate watchtower. In this case, the attacker could use the revocation keys to create revocation transactions that do not send the revoked funds to the customer but an address owned by the attacker. However, this also requires an outdated commitment transaction to be published on the blockchain. To run this attack, an active fraud attempt is needed. If the customer's channel partner does not attempt a fraud, the attack is only possible when the provider colludes with the customer's channel partner. The maximum profit of such an attack run by a watchtower provider, who broke the TEE and colludes with the channel partner of the customer, would be the difference between the customer's balance in the last commitment transaction and the commitment transaction with the lowest balance for the customer. In the worst case, this could be all funds of the channel. It is not possible for an attacker to create new commitment transactions because the `basepoint_secret` is not sent to the watchtower and thus cannot be stolen even by an attacker breaking the TEE.

6 Conclusion

In this work, we presented TEE Guard, a proposal for a watchtower solution for the Lightning Network. The use of TEEs led to a concept that scales, because TEEs allow secrets to be securely outsourced to the watchtower and, thus, revocation transactions to be created directly by the watchtower instead of the customer. The decentralised nature of TEE Guard ensures a high reliability of the watchtower service. We have shown with a simulation that, even at unrealistically high outage rates per platform provider, the whole system already has a reliability of 96 % with only three watchtowers. As customers do not trust the platform providers to keep their watchtowers running, we presented the logging mechanism that allows the providers to prove to their customers that they have continuously been running the service. Even a very strong attacker, who is able to break the TEE, does only get access to the revocation secrets, which cannot be abused as long as there is no dispute in the channel or the watchtower provider colludes with the customer's channel partner. We quantitatively analysed the concept showing that the system is deployable and running a provider platform is financially profitable for the provider even with very low prices for the customers.

For future work, we plan to analyse the requirements for watchtowers for other payment channel networks and the security trade-offs implied by adapting TEE Guard to these networks. We also want to generalise the concept by transferring the ideas of TEE Guard to state channels.

References

1. BOLT 3: Bitcoin Transaction and Script Formats (2018), https://github.com/lightningnetwork/lightning-rfc/blob/914ebab9080ccccb0ff176/03-transactions.md
2. Anati, I., Gueron, S., Johnson, S., Scarlata, V.: Innovative Technology for CPU Based Attestation and Sealing. In: Proc. of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy. HASP '13, ACM, New York, NY, USA (2013)
3. Avarikioti, G., Kogias, E.K., Wattenhofer, R.: Brick: Asynchronous State Channels. arXiv preprint arXiv:1905.11360 (May 2019)
4. Avarikioti, G., Laufenberg, F., Sliwinski, J., Wang, Y., Wattenhofer, R.: Towards Secure and Efficient Payment Channels. arXiv preprint arXiv:1811.12740 (2018)
5. Bentov, I., Ji, Y., Zhang, F., Li, Y., Zhao, X., Breidenbach, L., Daian, P., Juels, A.: Tesseract: Real-Time Cryptocurrency Exchange using Trusted Hardware. IACR Cryptology ePrint Archive **2017**, 1153 (2017)
6. Bulck, J.V., Minkin, M., Weisse, O., Genkin, D., Kasikci, B., Piessens, F., Silberstein, M., Wenisch, T.F., Yarom, Y., Strackx, R.: Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. In: 27th USENIX Security Symposium (USENIX Security 18). USENIX Association, Baltimore, MD (2018)
7. Das, P., Eckey, L., Frassetto, T., Gens, D., Hostáková, K., Jauernig, P., Faust, S., Sadeghi, A.R.: FastKitten: Practical Smart Contracts on Bitcoin. In: 28th USENIX Security Symposium (USENIX Security 19). pp. 801–818. USENIX Association, Santa Clara, CA, https://www.usenix.org/conference/usenixsecurity19/presentation/das
8. Decker, C., Russell, R., Osuntokun, O.: eltoo: A Simple Layer2 Protocol for Bitcoin. White paper: https://blockstream. com/eltoo. pdf (2018)
9. Dryja, T.: Unlinkable Outsourced Channel Monitoring (10 2016), talk at Scaling Bitcoin, Milano, 2016
10. Grundmann, M., Leinweber, M., Hartenstein, H.: Banklaves: Concept for a Trustworthy Decentralized Payment Service for Bitcoin. In: 2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC). pp. 268–276 (May 2019). https://doi.org/10.1109/BLOC.2019.8751394, https://publikationen.bibliothek.kit.edu/1000092459
11. Intel: PoET 1.0 Specification (2015), https://sawtooth.hyperledger.org/docs/core/releases/latest/architecture/poet.html
12. Kaplan, D., Powell, J., Woller, T.: AMD Memory Encryption (2016), http://developer.amd.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v7-Public.pdf
13. Lee, D., Kohlbrenner, D., Shinde, S., Song, D., Asanović, K.: Keystone: A Framework for Architecting TEEs. arXiv preprint arXiv:1907.10119 (2019)
14. Lind, J., Eyal, I., Kelbert, F., Naor, O., Pietzuch, P.R., Sirer, E.G.: Teechain: Scalable Blockchain Payments using Trusted Execution Environments (2017), http://arxiv.org/abs/1707.05454
15. Matetic, S., Ahmed, M., Kostiainen, K., Dhar, A., Sommer, D., Gervais, A., Juels, A., Capkun, S.: ROTE: Rollback Protection for Trusted Execution pp. 1289–1306 (Aug 2017), https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/matetic
16. McCorry, P., Bakshi, S., Bentov, I., Miller, A., Meiklejohn, S.: Pisa: Arbitration Outsourcing for State Channels. IACR Cryptology ePrint Archive **2018**, 582 (2018)

17. McKeen, F., Alexandrovich, I., Berenzon, A., Rozas, C.V., Shafi, H., Shanbhogue, V., Savagaonkar, U.R.: Innovative Instructions and Software Model for Isolated Execution. In: Proc. of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy. HASP '13, ACM, New York, NY, USA (2013)
18. Milutinovic, M., He, W., Wu, H., Kanwal, M.: Proof of Luck: An Efficient Blockchain Consensus Protocol. In: Proc. of the 1st Workshop on System Software for Trusted Execution. pp. 2:1–2:6. SysTEX '16, ACM, New York, NY, USA (2016), http://doi.acm.org/10.1145/3007788.3007790
19. Nakamoto, S.: Bitcoin: A Peer-to-Peer Electronic Cash System (2008), https://bitcoin.org/bitcoin.pdf
20. Osuntokun, O.: Hardening Lightning (01 2018), talk at Blockchain Protocol Analysis and Security Engineering, 2018
21. Poon, J., Dryja, T.: The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments (2016), https://lightning.network/lightning-network-paper.pdf