Twilight: A Differentially Private Payment Channel Network

Maya Dotan*, Saar Tochner*, Aviv Zohar, and Yossi Gilad

The Hebrew University of Jerusalem

Abstract

Payment channel networks (PCNs) provide a faster and cheaper alternative to transactions recorded on the blockchain. Clients can trustlessly establish payment channels with relays by locking coins and then send signed payments that shift coin balances over the network's channels. Although payments are never published, anyone can track a client's payment by monitoring changes in coin balances over the network's channels [22, 30]. We present Twilight, the first PCN that provides a rigorous differential privacy guarantee to its users. Relays in Twilight run a noisy payment processing mechanism that hides the payments they carry. This mechanism increases the relay's cost, so Twilight combats selfish relays that wish to avoid it using a trusted execution environment (TEE) that ensures they follow its protocol. The TEE does not store the channel's state, which minimizes the trusted computing base. Crucially, Twilight ensures that even if a relay breaks the TEE's security, it cannot break the integrity of the PCN. We analyze Twilight in terms of privacy and cost and study the trade-off between them. We implement Twilight using Intel's SGX framework and evaluate its performance using relays deployed on two continents. We show that a route consisting of 4 relays handles 820 payments/sec.

1 Introduction

Blockchain systems such as Bitcoin create a public ordered log of transactions without relying on a trusted party. Instead, these systems distribute trust among potentially many participants that connect in a peer-to-peer network. They allow for fair trading protocols [4] and can reduce friction in financial markets [16]. Users expect meaningful privacy guarantees from financial systems, but most blockchains only provide so-called "pseudo-anonymity", where actions are associated with pseudonym addresses rather than the users' identities. Several works show that often anyone can link users to their addresses by analyzing the public information posted on the blockchain [35, 37]. To counter such analysis, researchers proposed systems like ZCash [3], which rely on zero-knowledge proofs to provide excellent privacy. However, these systems bear significant performance costs due to their reliance on a system-wide consensus regarding the transactions' log.

A promising direction for achieving high-throughput and low-latency transactions with a meaningful privacy guarantee is connecting users through a payment channel network (PCN). In a PCN, users create payment channels with relays by locking coins in joint on-blockchain accounts and use the blockchain again only to commit the coin distribution when a channel closes. The relays are interconnected through payment channels as well. Alice pays an indirectly-connected Bob by finding a route of channels through the network's relays to him and then creating a payment that would move coins to the first relay, conditioned on a secret that only Bob knows. Each relay creates a similar payment for the next hop until Alice's payment reaches Bob, who reveals the secret. Intuitively, since users neither broadcast payments they make nor create payment channels with the users they transact, PCNs seem promising for simultaneously addressing the blockchain's sclability bottleneck and privacy challenge.

In practice, however, PCNs offer very little privacy [17, 23, 25, 30], much less than users would expect from traditional payment systems (e.g., governed by banks). Any user can test whether the liquidity (number of coins available to transact) on any channel is below a threshold that they choose – merely by asking to relay a payment on that channel for the threshold amount. As a result, anyone can learn about changes in each channel's liquidity, which leaks all the information about the network's payments and creates a privacy risk in practice [22].

We present Twilight, a PCN that hides a user's payment history from other users. Twilight provides a strong privacy guarantee supported by a differential privacy analysis. It models payments over a channel as queries, and the responses convey whether or not they can go through the channel (i.e., the channel has sufficient liquidity). Twilight provides payment-privacy by noising these responses. Instead of carrying a payment whenever sufficient liquidity exists, the relay rejects

*Both authors contributed equally

payments if they would not leave enough liquidity to cover a randomized "noise" payment. Through this noisy payment processing mechanism, Twilight ensures that only a small amount of statistical information about the payments it carries may leak to other users. The key to designing Twilight is structuring the noise such that the privacy guarantee holds even if the attacker uses many clients to send queries at a high rate and observes the responses over a long time. Since queries are cheap to execute, this property is crucial to providing a meaningful privacy guarantee. At the same time, it is important to avoid throttling the rate of queries, which hurts honest users and lowers the system's throughput even when there is no attack. To achieve this, Twilight leverages techniques from the differential privacy literature to reuse noise values when hiding the same set of payments [10,34]. This allows Twilight to group payments into subsets and mask them with noise, ensuring that new information about old payments does not leak to the attacker with every query.

Twilight's differential privacy approach comes at a cost: it might block some payments due to the noise that a vanilla PCN would carry. Relays in Twilight mitigate this issue by locking extra coins. Of course, a relay can always tear down the channel and get these coins back but locking coins still incurs an operational expense. We analyze this trade-off between privacy and cost, and evaluate a model where users cover the relay's expense with payment fees. We quantify the financial impairment from locking coins and estimate the payment volume a relay would need to process to cover this cost by charging fees for its privacy service. For example, a relay handling a payment volume of 79-coins/day and charges a 1% fee covers the cost of operating a noisy channel that hides 1-coin payments (for privacy level $\varepsilon = 0.15, \delta = 10^{-7}$, in differential privacy terms [11]).

Selfish relays may claim to their users that they do noisy payment processing but attempt to avoid it in practice to save costs. Twilight addresses this problem by running the noising logic in a trusted execution environment (TEE), which allows clients to verify the payment processing logic. We architect the code operating inside the TEE to avoid keeping the channels' state. In particular, it does not keep track of the channel's liquidity or payments already processed. Instead, the relay provides this state every time it calls the TEE to process a payment. Designing Twilight in this manner minimizes the trusted computing base inside the TEE but requires handling two key challenges. First, a relay might inform the TEE that it has very high liquidity, enough to cover any noise it might choose. Second, a relay might attempt to eliminate the effect of noising payments by repeatedly calling the TEE with the same payment until the TEE adds sufficiently-low noise to approve it. Twilight handles these challenges by noising payments on the relay's incoming channel and encrypting payments between TEEs until they reach the recipient. A relay cheating its TEE about the incoming channel's liquidity only risks losing coins by accepting payments that the previous hop cannot cover. Each TEE outputs its noisy payment processing result encrypted for the next TEE on the route, so all of a TEE's outputs are indistinguishable, and the relay cannot choose the response it likes. Only when a payment reaches the recipient, all relays en-route can decrypt the TEE outputs and update the channel liquidity. Importantly, Twilight does not rely on TEEs to secure funds. Even if a relay exploits a vulnerability in the TEE or Twilight's code that it executes to learn the TEE's secrets, all payments Twilight carries are valid, and the relay cannot steal coins from others.

We implement Twilight and test its performance using machines in America and Europe with the SGX TEE [9] in Azure. A route of 4 relays supports 820 payments/sec, and at this peak rate, the payment latency is 550ms above the network latency (which is about 510ms in our experiments).

In summary, our contributions are the following:

- A rigorous definition for privacy in PCNs based on differential privacy, and the design of Twilight, a PCN that meets this goal.
- The combination of noising payments inside a TEE to combat selfish relays.
- An analysis of Twilight's integrity, privacy and cost.
- An implementation and performance evaluation.

2 A Primer on PCNs

We overview the key concepts and mechanisms that comprise PCNs as background for describing Twilight.

Payment channels. Two participants establish a bidirectional payment channel by "locking" coins in a joint on-chain account that requires both parties to approve every spend. The amount that each participant locks is the channel's initial liquidity in the direction of their peer. The locked coins ensure that payments carried over the channel can always be redeemed. Alice pays Bob by signing a transaction that adjusts the split of their account balance (giving more coins to Bob than he deposited). Bob can counter-sign this transaction at any time and redeem his funds by posting it on the blockchain, which would also close the payment channel. Instead, Bob holds the transaction from Alice. He keeps the channel open to allow them to continue updating the balance split. A short appeal period begins when either user posts a transaction that closes the channel. During this time, posting a transaction with information that proves that the channel was closed with an obsolete state corrects the account split. In this manner, the appeal period protects users if their counterpart closed the channel using an outdated transaction.

Routing payments. A PCN comprises clients that issue payments and relays that carry these payments between clients for a fee. Relays and clients connect through payment channels, allowing Alice to pay Bob even if they are not directly connected. When two relays establish a channel, they start

announcing the channel through a peer-to-peer network. Alice's client finds a route to Bob and onion-encrypts it with the relays' public keys. Namely, it encrypts Bob's identity using the last relay's key, concatenates to the ciphertext that relay's identity, and encrypts again with the previous relay's key. It continues in this fashion for every hop. When a relay receives a payment, it subtracts a fee from the amount and decrypts the top layer to find the next hop.

Hash-time-locked contracts (HTLCs). Hash-time-locked contracts are a mechanism for ensuring that every relay along a payment's route that sends coins to the next hop can recover the funds from the previous hop. Before Alice can pay Bob, his client chooses a random secret s and gives Alice's client $h(s)$, the cryptographic hash of s. Alice's payment moves coins to the first relay conditioned on the relay providing s before a deadline; this condition is called an HTLC. Each relay creates a similar payment conditioned on the preimage of $h(s)$, which it sends to the next hop in the route. As soon as Bob receives the payment, he can potentially post s on the blockchain and redeem the payment; this would also reveal s to all the other relays, allowing them to redeem their payments as well. Typically, however, Bob avoids the on-chain transaction: his client reveals s to the last relay and asks the relay to sign a transaction that updates the split in the channel between them – this time, without conditioning on s. That relay can then do the same, show s to the previous hop, and update the previous channel's split. This way, s propagates back through the route, updating all channel balances and eventually reflecting that Alice paid Bob.

3 Overview

Figure 1 shows a PCN connecting three users. The users connect to the PCN via private channels (illustrated by the dashed links in Figure 1). Their clients reject payments from unauthorized origins [36] which mitigates exposing information about the liquidity on these channels to other users. In this figure, Alice sends a payment to Bob via two inter-relay channels. In contrast to the user's private channels, relay-to-relay channels are public and any client can route payments over them. This allows an attacker to probe for changes in channel liquidity by asking the relay to route payments between his clients and observe whether the relay rejects his payments for exceeding the available liquidity. By probing for changes in liquidity across inter-relay channels, an attacker can track payments between users in the network [15, 17, 30]. Twilight is a PCN that hides its users' payments from attackers probing inter-relay channels by introducing noise to a relay's payment processing logic. It ensures users that relays add this noise by leveraging a trusted execution environment (TEE). We next formalize the threat model and Twilight's goals against it.

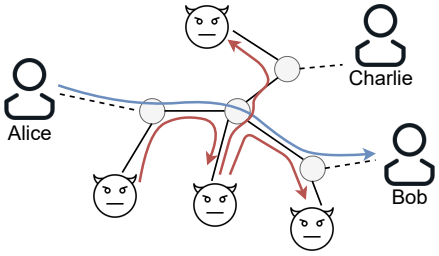


Figure 1: Users connect to relays through private channels (dashed black line), while inter-relay channels are public (solid black lines). An attacker sends payment requests between his clients (in red) to monitor changes in liquidity and infer about payments between users (in blue).

3.1 Threat model

Before delving into defining Twilight's threat model and goals, we discuss different types of potential attackers.

Off-path clients. Today's PCNs are vulnerable to attacks by *clients*: anyone on the Internet can probe payment channels and learn about others' payments [22, 30]. This constitutes a lower standard of privacy compared to traditional financial systems. Such attacks are easy to launch and difficult to block since the attacker may send probes via multiple clients and disguise probes as legitimate payment requests.

On-path selfish relays. Users should *know* their payment history is hidden from other clients. In particular, PCNs are designed as distributed systems that include relays operated by multiple autonomous organizations. A privacy solution that involves costs that intermediate relays can shirk (at the expense of users' privacy) should ensure that relays follow the protocol despite these costs.

On-path adversarial relays. Much like other PCNs, Twilight uses onion routing for payments (§2), which protects against network adversaries and malicious relays that only have one local visibility or presence in the route, so they can only view the previous or next hop but not the entire route. However, colluding relays located near the payer and payee may still track the payment by correlating its time, payout value, or locked contract's secret. Such attacks are harder to launch than those involving only off-path clients: as shown in [42], the number of relays on the route is small, and since the payer chooses which route to use, she can avoid relays that she considers less trustworthy (e.g., operated by less reputed organizations). We architect Twilight to provide similar usability to vanilla PCNs but with better privacy; it, therefore, does not handle such correlation attacks, since that would require delaying payments and somehow ensuring that Bob's balance changes regardless of Alice's balance changes (to break these correlations). Recent works allow decorrelating the locked contract's secret [30, 31]; we discuss it in related work. We believe that techniques from the private communi-

cation literature, such as Poisson mixing [32], where every relay delays payments using a randomized mechanism can address the timing challenge, but hiding correlated changes in balances remains a challenge. We leave this attacker model out of Twilight's scope.

Secrecy of closing balances. Payment channel teardown involves posting a blockchain transaction stating its closing balance split (§2). Since anyone watching the blockchain can observe the channel-close transaction, the privacy that Twilight can provide depends on the underlying blockchain. Twilight gives the most privacy when the PCN deploys over a blockchain that supports private transactions (e.g., using zero-knowledge proofs [24]). However, the closing transaction reveals only the aggregate sum of payments over the channel through its lifetime. In practice, payment channels are open for a long time [1], so we expect the closing balance to reveal very little about a user's payments. We design Twilight to be largely-independent of the blockchain choice (except for requiring smart contract support), and it can deploy over blockchains with or without support for private transactions.

3.1.1 Twilight's attacker

Twilight hides a user's payment history in the face of attackers who can run any number of the PCN's clients even if the user routes her payments through selfish relays. The attacker can route payments over any inter-relay channel to probe the available liquidity. These probes are extremely cheap: the attacker can abort the payments after seeing the response and avoid the associated fee.

3.2 Goals

Integrity. Twilight should enable private off-chain payments as a viable and trustless substitute to on-chain payments. Therefore, it must guarantee that the sequence of all payments that a channel carries can be committed to the blockchain. This property must hold even if all relays break the TEE security guarantee (i.e., they attest different code than what they execute and learn all the TEE's secrets).

Privacy. There are several ways to capture privacy in PCNs. For one, Twilight could target hiding just the payment amount. However, in many cases exposing that Alice pays Bob is sufficient to reveal sensitive information. For example, donating to a political party, no matter what amount, tells Alice's political views. Twilight could also ensure that Alice has at least one "cover story," making paying Bob appear similar to paying at least one other user. However, this approach might provide users with a cover story that does not make sense in practice (e.g., the alternative payee might be in another continent), and users might not be aware of the cover story the system then provides. Instead, Twilight sets an ambitious goal: ensuring that the attacker's view through the clients he operates is likely to be the same regardless of whether Alice makes her payment. In particular, this approach makes Alice paying Bob look similar to Alice paying *any other* user. (Since Alice paying Bob looks similar to not making a payment, which then looks similar to Alice paying any other user.)

More formally, consider the vector O of the attacker's observations (from routing payments between malicious clients) and two scenarios: one scenario where a user, call her Alice, pays another user, call him Bob, and the other scenario where Alice never makes this payment. Twilight's goal is to provide (ε, δ)-differential privacy [11] with respect to the two scenarios above. It guarantees that the following inequalities hold for a small $\varepsilon \geq 0$, except for a small error probability $\delta \geq 0$:

$$e^{-\varepsilon}\Pr[O|\mathsf{X}] \leq \Pr[O|\text{Alice} \to \text{Bob}] \leq e^{\varepsilon}\Pr[O|\mathsf{X}] \qquad (1)$$

The arrow denotes Alice paying Bob, and the X-mark denotes the case where she does not make this payment. The probability is over the coin-tosses in the nosing mechanism. Differential privacy quantifies the statistical information that leaks to the attacker. It provides a strong formal guarantee for the level of privacy users should expect. This privacy guarantee holds even for multi-hop payments (payments routed through several payment channels between Alice and Bob); we quantify impact of a payment's path length on its privacy in terms of ε and δ.

	Description
$\mu, \sigma > 0$	mean and standard deviation for Gaussian noise
T	number of time-slots (leaves in the tree)
$\mathcal{N}_t$	the minimal set of tree nodes covering $[0, t-1]$
N	$\lvert\mathcal{N}_t\rvert$ upper bound, $N = \lceil \log_b T \rceil + b - 2$
$b \geq 2$	children per node in tree

Table 1: Symbols of the noisy payment processing mechanism

4 Noisy Payment Processing

Twilight protects users against attackers probing inter-relay links (Figure 1). It provides a strong privacy guarantee, regardless of the attacker's probing rate, by adapting ideas for continuous release of information from the differential privacy literature [10, 11, 27] to the PCN context. The system splits time into short intervals, arranged in a tree, and models the attacker's probes as queries arriving in those intervals. Twilight cannot distinguish between the attacker's probes and real payments so it must respond to both. It protects users' privacy through noisy payment processing. Twilight noises the decision whether to carry or drop a payment (the relay's response) in a structured way, depending on the interval the query arrived in [34]. Table 1 summarizes the notations of the noisy payment processing mechanism described below.

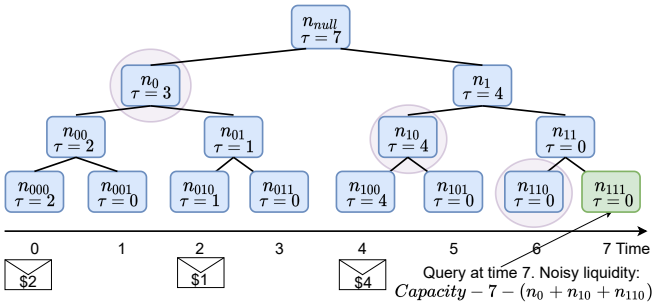


Figure 2: A channel's noise tree with $T = 2^3$ slots. For node i in the tree, n_i is the noise and τ_i the sum of payments in its subtree. The response mechanism for query at time 7 uses the nodes $\{0, 10, 110\} \in N_7$ (circled in red).

4.1 Response mechanism

Twilight divides the lifetime of a channel into T tiny time-slots. In every slot, at most one request to carry a payment may arrive. For example, dividing time into nanosecond slots seems reasonable since payments are over 100B-long (they include account addresses, a signature, etc.). Even with 100Gbps links, each request takes at least 7 time-slots to receive. Allocating 2^{64} nanosecond slots supports a channel's operation for over 500 years.

Time slots are numbered and arranged in the leaves of a tree; each node in the tree has a label. Consider a binary tree, for now. The root has the empty label, and every node's left/right child appends '0'/'1' to the parent's label. Figure 2 illustrates this tree structure. The relay assigns every node i in the tree an independently drawn Gaussian noise, $n_i \in_R Gauss(\mu, \sigma^2)$. In addition, node i stores τ_i, the sum of all payment amounts over the channel (positive or negative, depending on the direction) in the time-slots beneath it.

When a payment requesting to move m coins to the channel's other endpoint arrives at the relay at time slot t, the relay finds $\mathcal{N}_t$, the minimal set of nodes in the tree that precisely covers (i.e., includes the ancestors of) the time slots in the interval $[0, t-1]$. Appendix A provides the algorithm for finding the minimal covering and proves its correctness. The relay agrees to carry the payment if Equation 2 holds:

$$channel.Capacity - \sum_{i \in \mathcal{N}_t} (\tau_i + n_i) \geq m \qquad (2)$$

Figure 2 illustrates the response mechanism for a request at time t = 7. In this example, $\mathcal{N}_7 = \{0, 10, 110\}$ is the minimal covering set of nodes for the time slots preceding the query (slots 0–6). Since $\mathcal{N}_t$ is the minimal set of tree-nodes that covers the interval $[0, t-1]$, $\sum_{i \in \mathcal{N}_t} \tau_i$ is the sum of all payment amounts before time t, so $channel.Capacity - \sum_{i \in \mathcal{N}_t} \tau_i$ is the current liquidity. The response in Equation 2 obfuscates it by subtracting noise, i.e., the aggregate of Gaussians $\sum_{i \in \mathcal{N}_t} n_i$.

The tree structure from the differential privacy literature [10] ensures that for any user payment at time $t' < t$, there is only one ancestor of the leaf node t' in the covering $\mathcal{N}_t$. Only that ancestor might leak information about user's payment (by contributing a different value to the response mechanism if the user never makes this payment). Since there are only $\log T$ ancestors for any slot, there are only a few tree nodes that might leak information about a user's payment over time (even if the attacker probes often).

Intuitively, the more nodes in $\mathcal{N}_t$, the more noise the payment processing mechanism subtracts from the channel's capacity in Equation 2, and the less accurate the response becomes. Qardaji et al. [34] optimize privacy and accuracy using a tree with a higher branching factor. When every node in the tree branches to b children (Table 1), the size of the covering set $|\mathcal{N}_t| \leq \lceil \log_b T \rceil + b - 2$ (one node from each level in the tree, except the last level where there may be up to $b-1$ leaves). We denote this upper bound by N (Table 1). With $T = 2^{64}$ time slots, using $b = 8$ results in the minimal number of nodes in the covering, $N = 28$ nodes (rather than 64).

If the covering set has less than the maximal N nodes, the relay "pads" it with dummy nodes, i.e., nodes where $\tau = 0$ and fresh noise is drawn. So, every time a relay responds it uses the aggregate of N Gaussian random variables as noise. The reason for padding is that the aggregate is much more likely to be non-negative than the noise of a single node. We later use this property to prove Twilight's integrity (§6).

4.2 Stateless noising

Storing the tree illustrated in Figure 2 is impractical for a large number of time-slots (e.g., 2^{64}). Every node i in the tree contains two elements: the sum of payment amounts in the time-slots of its subtree (τ_i) and an independently drawn noise (n_i). The response mechanism in Equation 2 only uses the sum of all payment amounts before the request's time slot. A relay, therefore, only keeps track of this sum. It also avoids storing the noise values and instead recomputes them when they are needed. Specifically, the relay keeps one global secret s, and for a payment-channel *chanID*, sets the secret value of tree node i to be deterministically drawn from the noise distribution seeded by the hash value $h(s, chanID, i)$.

5 Countering Selfish Relays using TEEs

Users rely on Twilight's relays for privacy, but how can a payer trust that the relays she chooses for her payment's route indeed perform noisy payment processing (§4)? Moreover, she has to be convinced that the relays *continue* to perform noisy payment processing since the attacker's future queries may leak information about her payment. Providing users with this guarantee is important since relays may try to circumvent the noising mechanism to avoid rejecting payments they would otherwise carry and losing the associated fees.

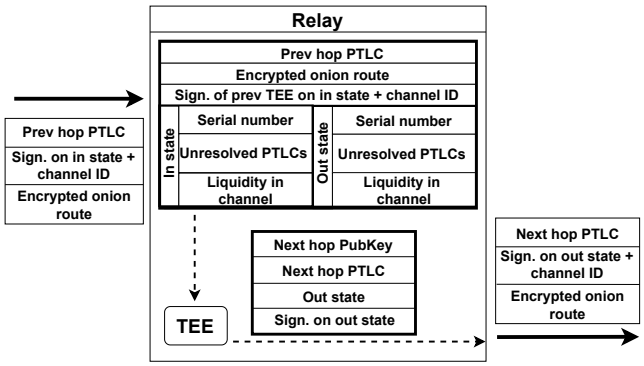


Figure 3: Protocol messages and relay-to-TEE interface. We abbreviate incoming/outgoing channel state by in/out state.

Twilight addresses this concern using a trusted execution environment (TEE) on its relays. TEEs ship with commodity hardware (e.g., SGX is now standard in Intel's i9 10th generation processors [18]); their remote attestation allows clients to verify the payment processing logic, which relays run inside an enclave [9]. Each channel is associated with a state that includes the current balance split between peers, all current pending payments, and a message serial number (for dispute resolution). Our design does not require the TEE to keep track of this state, which minimizes the trusted computing base and simplifies Twilight's implementation but introduces challenges as we describe next. Crucially, Twilight ensures that even if a rogue relay breaks its TEE and learns its secrets, the PCN maintains integrity: all payments over the channel are valid, and the relay cannot steal coins from others.

Challenges. Twilight must force a relay to involve the TEE in processing payments to ensure it performs noisy payment processing throughout the channel's lifetime. Figure 3 illustrates how a relay interacts with its TEE. The enclave running inside the TEE does not store the channel's state but receives it from the relay with every payment processing request to decide whether to approve a payment. This leads to two key challenges: First, the relay may provide the enclave with incorrect or outdated states. Specifically, ones in which a high amount of liquidity is available. Such manipulation can ensure that the noisy liquidity check in Equation 2 always passes. The relay may do this selectively, whenever it has sufficient liquidity to forward the payment and wishes to pass the noisy check with greater certainty. Second, the TEE cannot remember which payments it processed, so the relay may invoke the TEE multiple times to process the same payment until it adds sufficiently low noise to accept it.

Architecture. Twilight solves these challenges by hiding payment amounts and the TEE's responses from the relay, encrypting them for the TEE at the next hop. The enclave running inside the TEE performs noisy payment processing on the incoming channel when the relay receives a new payment request. Namely, the enclave (noisily) checks that the previous hop has enough liquidity on the channel to cover the current payment. To do so, the enclave gets from the relay the incoming channel's state, which consists of the channel's liquidity and a list of "unresolved" payments (for which the secret that completes the coin transfer has yet to be revealed, see §2), and processes the payment as in §4. The reasoning behind noising payments on the incoming channel (rather than the outgoing channel) is to mitigate a relay's incentive to cheat. Suppose the relay informs its enclave of false high liquidity on the incoming channel; this nullifies the check that the previous hop has sufficient liquidity to cover the payment, so the relay risks moving coins to the next hop without any return. There is no privacy risk if the relay reports false liquidity: since the TEE outputs are hidden from the relay (encrypted for the next hop), it cannot choose a response it likes. Therefore, it cannot adjust the noise distribution to the payment; instead, it has to submit the noisy response to the next hop. Privacy holds even if the relay colludes with the relay in the next hop, who also cannot access the information, which is encrypted for its TEE. We next explain how the TEEs establish public keys that facilitate this encryption and how relays interact with their TEEs to process payments. Appendix B provides a fully detailed version of the protocol.

5.1 Establishing public keys and channel IDs

The TEE creates a secret key, and the relay binds the corresponding public key to its payment channel. When two parties establish a new payment channel, they deposit coins to an account managed through a smart contract. Each party gives one public key to govern this account; a relay uses the public key corresponding to its TEE's secret key. In Twilight, clients contact the relays and check: (1) that the key was created in the TEE, and using remote attestation, (2) that the logic inside the TEE never exports the keys it creates and performs noisy payment processing. By registering the TEE's public key in the smart contract, the relay ensures its users that the TEE must be processing their payments on this channel. The smart contract's account address serves as the channel ID; signed messages about the channel include this ID, so they cannot be confused with other channels. The smart contract allows flexibility in designing dispute resolution when a channel closes. Specifically, Twilight uses the smart contract functionality to allow an offended party to reveal an unsettled encrypted payment (and correct the closing balance, if needed), as we describe in §5.3. Appendix B.1 provides the smart contract's pseudo-code, which we implement in Solidity (§8).

5.2 Processing payments inside TEEs

In Twilight, relays receive and forward encrypted payments. As a result, relays do not know the outcome of noisy payment processing, so payment messages always reach the payee,

even if a TEE on some relay along the route rejects the payments. Twilight does not keep payments hidden forever. A relay must be able to claim a resolved payment by posting a transaction on the blockchain with the payee's secret (which conditions the coin transfer across the route) in case of dispute with its partner. To support this functionality, the payee chooses a fresh private/public encryption key-pair for the payment's locked contract. The public key is unique to that payment's locked contract and allows hiding its amount. Instead of indicating "failure" when there is insufficient liquidity, the relay outputs a locked contract for the next hop with a hidden amount of zero (indicating no payment). The relay can decrypt the payment amount and redeem it when learning the corresponding private key from the payee, which also allows claiming the payment and is analogous to the HTLC secret from vanilla PCNs (§2). We term these locked contracts *"Private Time-Locked Contracts" (PTLCs)* rather than HTLCs (hash time-locked contracts). PTLCs generalize HTLCs; similarly to HTLCs, they specify an expiration time (e.g., by block depth on the blockchain) and signed by the previous hop (see §2). The main difference between them is that while HTLC payouts are always legible (even if not redeemable without the secret), the payment amounts in PTLCs cannot be read without the secret. This allows providing the PTLC contents specifically for a relay's TEE when the payment propagates to the payee, and preventing the relay from learning the result of its TEE's noisy payment processing from the PTLC it outputs for the next relay. Appendix B.2 gives the details of implementing PTLCs.

Consider a payment from Alice to Bob and the private key that Bob chooses for conditioning this payment. Bob informs Alice of the corresponding public key, which she includes in her payment's PTLC. On the route between Alice and Bob, the enclave running in each relay's TEE must learn the payment amount when it processes Alice's payment, i.e., before Bob confirms it and reveals the secret. To support this functionality, Alice uses non-malleable encryption with a fresh ephemeral symmetric key to hide the payment amount in the PTLC. PTLCs include two ciphertexts of that key: (1) under the next hop's TEE public key, and (2) under the PTLC's public key that Bob chose. The TEE decrypts the amount encrypted under its key and checks for consistency with the second ciphertext by encrypting the ephemeral key it recovers with the PTLC's public key. The second encryption is deterministic to allow this check; it is safe since the ephemeral symmetric keys are never reused.

Alice submits her payment for Bob to the first relay in the route she chooses. We next describe how relays process payments and their interaction with their TEEs, following Figure 3. Our description refers to Appendix B.3 for more details about protocol's messages and processing logic.

TEE input. When a relay calls its TEE to process a payment inside an enclave, it provides the payment message from the previous hop, the corresponding incoming channel ID and its liquidity in the direction of the payment, and any PTLCs for unresolved payments from the previous-hop (these are payments that have not been completed yet, but represent commitments from the previous hop).

TEE processing. The enclave decrypts the amount and performs the consistency check for the two ciphertexts above. It subtracts from this amount the relay's fee. It then computes "uncommitted liquidity", which is the incoming liquidity minus the coins in the relay's unresolved PTLCs. Lastly, it performs noisy payment processing (§4) and evaluates Equation 2 to determine whether to accept the payment.

The noise that a relay induces depends on the current time (§4). However, it is unsafe for the relay to provide the time directly to the enclave. Otherwise, a selfish relay could identify some slot in the tree where the nodes in its covering-set ($\mathcal{N}_t$, see Table 1) sum to very little noise, and then replay this slot's timestamp for all payments to essentially circumvent the noising mechanism. Fortunately, TEEs allow enclaves to read the number of cycles since boot, and guarantee its authenticity, which serves as time [19]. The TEE also allows detecting when a relay reboots and the time initializes to zero: on boot, the TEE draws a random number which Twilight's enclave echos to the relay with every response. The relay must provide this value to the TEE when asking to process a payment. The enclave compares the given value by reading the random number again; if they match, the relay has consistent time since the last response (and recursively since boot) and processes the payment as usual. Otherwise, the relay refuses to process the payment. This ensures users that the relay noises payments correctly throughout the channel's lifetime.

When the relay reboots, the random number changes and the TEE refuses to process more payments. The relay then has to close the channel on-chain by posting the last closing balance message from its partner. We expect relays to have high up-time, to collect fees from all payments that need to route through them; so the cost of the blockchain transaction in such "forced" closures is amortized over a long time. A relay may also use another computer with TEE as backup, which keeps track of time and allows recovery in case of a crash without closing the channel. We describe the details of this extension in Appendix C.

TEE output. The TEE decrypts the top layer of the onion route and creates a new PTLC for the next relay's TEE with the remaining amount (encrypted under a fresh symmetric key for the next hop's TEE and the PTLC public key from Bob). The TEE then signs a message combining the channel ID, its liquidity, all pre-existing unresolved PTLCs and the new PTLC along with an incremented serial number. The next hop checks the TEE's signature on this combination before it continues processing the payment (this signature both ensures that the contents were created by the TEE and hence noised, and that the next hop will be able to claim funds on the blockchain). When the enclave rejects a payment, it

uses zero for the amount in the output PTLC. This ensures that the following relays on the route also use zero amounts, meaning that the payment will not eventually take place.

Payment confirmation and coin transfers. When Bob receives the final PTLC, his client checks that the amount from Alice is sufficient. In this case, Bob reveals the PTLC secret key to the previous relay, which serves as a receipt. It allows the relay to decrypt the PTLC's amount and post the PTLC, if necessary, to the blockchain to ensure the relay gets paid. Each hop then propagates the secret backward, which enables coin transfers across the route (as described in §2).

5.2.1 Side channels

Several exploits in the past illustrate that TEEs can be vulnerable to side channels (see survey on Intel's SGX platform [12]). We architect Twilight to maintain PCN integrity even if attackers completely break the TEE's security promises, but given such a vulnerability, a relay may circumvent Twilight's noisy payment processing and jeopardize its users' privacy. In particular, attackers with physical access to the TEE (the hosting relay in Twilight's case) may exploit channels such as temperature and power consumption readings to learn sensitive information like secret keys. However, TEE manufacturers issued countermeasures in response to known attacks and, using remote attestation, allowed users to learn whether the TEEs they contact run these countermeasures. Users in Twilight, thus, can avoid relays with known TEE vulnerabilities.

Another important type of side channel attack exploits application-specific vulnerabilities within the enclave. Twilight's code inside the enclave must be hardened against such attacks. This is typically achieved by ensuring data-oblivious computation [2, 43] that results in constant processing time, low variance power consumption, etc.

5.3 Channel teardown

When a user leaves the PCN, or a relay needs to replenish liquidity in a channel with another relay, they close their channels (and open another one, in the relay-to-relay case). Closing a channel involves an on-chain transaction to the smart contract, splitting the locked coins between the channel endpoints. Similar to vanilla PCNs, parties sign and exchange closing balance messages after each payment (§2). A relay's TEE signs these messages, as the TEE's secret key is authoritative for the relay's payment channels (§5.1). Each party stores the latest message from their counterpart, so they can post it on-chain to close the channel (even without interacting with their TEE at close-time). These messages have a serial number, local to the channel, and reference unresolved PTLCs on the channel by their hashed values (see Figure 3). The smart contract managing the locked coins in the channel's account allows a short appeal period, where parties may post closure messages with higher serial number, and then allows parties to claim the PTLCs referenced by the last message posting them along with their secrets.

To hide the channel's closing balance split (§3.1), Twilight can deploy over a blockchain that allows for private transactions in its smart contracts. For example, Ethereum supports a rich enough language that allows implementing "shielded transactions" in its smart contracts using zero-knowledge proofs (see [24] for implementation). Such a deployment allows fast off-chain payments with a differential privacy guarantee and avoids exposing information on channel tear-downs. We, therefore, believe this combination makes an attractive privacy-performance trade-off.

6 Analysis

We analyze Twilight's design against its goals from §3.

6.1 Integrity

All payments accepted by a relay must be valid, so that the resulting balances when closing channels may be committed to the blockchain. This property must hold even if all relays break their TEEs' security guarantees (§5), i.e., even in this extreme case, no relay can steal coins from others.

To achieve integrity, it is sufficient to ensure that the relay subtracts non-negative noise from its channel's liquidity (Equation 2). Thus, any payment that the relay accepts is of at most the channel's liquidity amount and does not overspend (so it can be committed to the blockchain). The noise that the relay subtracts is the sum of at most N Gaussian random variables (§4.1). Theorem 1 shows that setting $\mu >> \frac{\sigma}{\sqrt{N}}$ ensures that the chance for negative noise is extremely small.

Theorem 1. *A relay accepts a payment that overspends the channel's liquidity with probability* $\leq GaussCDF(\mu N, \sigma^2 N; 0)$.

Where GaussCDF is the noise cumulative distribution function (CDF) with mean μN and variance $\sigma^2 N$, evaluated at 0. And μ, σ, N are the noise parameters, summarized in Table 1.

Proof. Let us compute the probability that the relay adds negative noise for some time slot. Noisy payment processing adds N random variables to the liquidity, each distributing $Gauss(\mu, \sigma^2)$. Thus, a relay's noise distributes $Gauss(N\mu, N\sigma^2)$. (The sum of Gaussians is also a Gaussian.) The chance for negative noise is $GaussCDF(N\mu, N\sigma^2; 0)$. Namely, the noise CDF evaluated at 0. □

When $N\mu >> \sqrt{N}\sigma$, i.e., the noise's mean is much greater than its standard deviation, the chance of obtaining negative noise is extremely small. It decays proportionally to $e^{-(\frac{\mu\sqrt{N}}{\sigma})^2}$. For example, for $\mu = 10\frac{\sigma}{\sqrt{N}}$ the chance for negative noise error is about 2^{-100}. This property holds regardless of the

TEE's security, i.e., even in case a new vulnerability allows the relays to circumvent TEE protections; Appendix B.4 formally proves integrity under insecure TEEs.

6.2 Privacy

Before we delve into the analysis of Twilight's differential privacy guarantee, we argue that as long as a relay's TEE is secure, that relay performs noisy payment processing. Consider a relay with a TEE that attests to running noisy payment processing on a channel. Any client that routes a payment through the channel ensures that the secret keys for the channel's on-chain account are created in, and never exported from, the TEE (§5.1). When a relay processes a payment's PTLC, it must sign the PTLC for the next hop using the TEE's secret. Hence, the relay must call the TEE, with the PTLC from the previous hop, which executes noisy payment processing and creates the next PTLC according to the result of Equation 2.

The privacy analysis for noisy payment processing proceeds as follows. We first analyze the differential privacy guarantee when the attacker probes a channel just once (Theorem 2). We then reason about hiding a payment under many probes, traversing a route with several channels, and the privacy amplification from choosing one of several available routes (Theorem 3). Lastly, we discuss differential privacy for multiple payments.

Privacy against a single probe. Consider a payment channel's noise tree (Figure 2) and Alice's payment arriving at the relay at time slot t. The only nodes in the tree affected by her payment are the $\log_b T$ ancestors of slot t (see parameters in Table 1). When the attacker issues a probe at time t', only one of these ancestors affects the relay's response (Equation 2), call it node i. Thus, for a single probe, it is sufficient to analyze the privacy loss from the information in a single tree node. For example, the payment at slot 2 in Figure 2 only affects the amounts (denoted by the τ's) that nodes $010, 01, 0,$ and the root record. Node i contributes to the response the sum of its noise and transacted amounts (i.e., $n_i + \tau_i$). Theorem 2 captures, in (ε, δ)-differential privacy terms, the difference in the chance that i contributes the same value whether Alice makes a payment of m coins.

Theorem 2. *Consider the time slot where Alice may make a payment to Bob for m coins. Let i be a node in a channel's tree that is an ancestor of that time slot. Then, except with probability* δ*, the following inequalities hold:*

$$e^{-\varepsilon} \leq \frac{Pr[\tau_i + n_i | Alice \to Bob]}{Pr[\tau_i + n_i | X]} \leq e^{\varepsilon}$$

Where $\varepsilon = \frac{mc}{\sigma}, \delta = 2 \cdot GaussCDF(\mu, \sigma^2; \mu - c\sigma)$.

Informally, δ bounds the probability of drawing extreme noise values (over c standard deviations below the mean). The parameter $c > 0$ allows to trade a larger ε for a smaller δ.

Proof. Alice pays m coins to Bob. So, for any value η that node i might contribute to the calculation of Equation 2 (i.e., the sum $\tau_i + n_i$), we need to bound the ratio:

$$\frac{\Pr[\tau_i + n_i = \eta | Alice \to Bob]}{\Pr[\tau_i + n_i = \eta | X]} = \frac{\Pr[n_i = \eta - \tau_i - m]}{\Pr[n_i = \eta - \tau_i]}$$

Namely, for node i to contribute the same value in both scenarios (Alice pays m coins to Bob vs. Alice does not make this payment), the noise in case Alice pays Bob should be m less than the noise in the case she does not pay him. The noise n_i distributes $Gauss(\mu, \sigma^2)$. For convenience, let us substitute $x_i = \eta - \tau_i$, which is the noise value that the relay should draw if there is no payment from Alice to Bob (s.t. i contributes η). The Gauss distribution PDF gives that the above term equals:

$$= \frac{e^{-\frac{(x_i - m)^2 - 2x_i\mu + 2m\mu + \mu^2}{2\sigma^2}}}{e^{-\frac{x_i^2 - 2x_i\mu + \mu^2}{2\sigma^2}}} = \frac{e^{-\frac{x_i^2 - 2mx_i + m^2 - 2x_i\mu + 2m\mu + \mu^2}{2\sigma^2}}}{e^{-\frac{x_i^2 - 2x_i\mu + \mu^2}{2\sigma^2}}} = e^{-\frac{-2mx_i + m^2 + 2m\mu}{2\sigma^2}} = e^{\frac{2mx_i - m^2 - 2m\mu}{2\sigma^2}} \quad (3)$$

The chance that the relay draws an "extreme" noise value for x_i is low. More precisely, except with probability $\delta_{right} = 1 - GaussCDF(\mu, \sigma^2; \mu + c\sigma) = GaussCDF(\mu, \sigma^2; \mu - c\sigma)$, it holds that $x_i \leq \mu + c\sigma$. Substituting x_i in Equation 3 with this upper-bound, we get that except with probability δ_{right}:

$$\leq e^{\frac{2m(\mu + c\sigma) - m^2 - 2m\mu}{2\sigma^2}} \leq e^{\frac{cm}{\sigma}} = e^{\varepsilon}$$

The computation for the inequality in the other direction, showing that $e^{-\varepsilon} \leq \frac{\Pr[\tau_i + n_i | Alice \to Bob]}{\Pr[\tau_i + n_i | X]}$, is similar. (We derive ε by substituting $x_i \geq \mu - c\sigma$, which holds except with probability $\delta_{left} = GaussCDF(\mu, \sigma^2; \mu - c\sigma)$.) Overall, using the union bound, $\delta = \delta_{left} + \delta_{right} = 2GaussCDF(\mu, \sigma^2; \mu - c\sigma)$. □

Privacy under many probes. Attackers may probe a channel many times; however, there are only $\log_b T$ ancestors in the tree for Alice's payment time slot. Thus, only those nodes might contribute different values to the relay's response (and, therefore, leak information about Alice's payment). We compose, in §6.2.2, the privacy guarantee from Theorem 2 over $\log_b T$ different observations that the attacker might get. This gives the differential privacy guarantee that a single channel provides (even if the adversary continuously probes it).

Privacy over a payment's route. The attacker may probe any inter-relay payment channel on Alice's payment route. Therefore, the level of privacy reduces with the number of inter-relay payment channels that it traverses. Although minimizing the number of hops benefits privacy, the costs that current PCN implementations opt to minimize might cause clients to prefer longer routes. For instance, in Lightnings' most common implementations, routing is not done by choosing the shortest-length route. In the C-Lightning implementation, the route's length is one of the parameters determining which route a payer chooses but in LND and Eclair (the other two most popular implementations), minimizing the route length is not explicit.

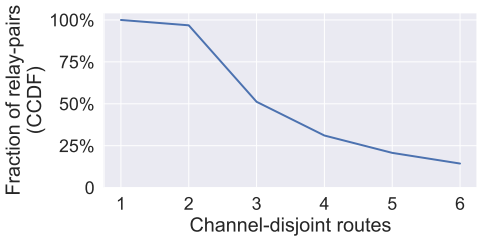


Figure 4: Shortest length channel disjoint routes between relays in the Lightning network topology (from Nov. 7th, 2021).

Usually, however, these costs decrease with the number of hops: having more hops typically results in less preferred routes, e.g., requiring clients to lock coins for a potentially longer time or increasing the total fee the sender would end up paying the relays. Indeed, measurements show that for all the implementations listed above, at least 80% of routes chosen by the routing algorithm in the Lightning Network (the largest PCN today) have no more than three inter-relay channels [42].

6.2.1 Privacy amplification by random route selection

In practice, multiple channel-disjoint routes of the shortest length often exist. For example, we connected a client to the Lightning Network and retrieved its topology (the snapshot was taken Nov. 7th, 2021). We found that there are usually $2-5$ channel-disjoint routes of the shortest length between two relays (nodes with more than one channel); see Figure 4. Twilight leverages this insight about PCN topology to amplify users' privacy. Clients choose one of the shortest channel-disjoint routes uniformly at random. Intuitively, randomized route selection improves privacy since the attacker does not know which of the channels he probes are on the payment route. If Twilight is to be deployed on Lightning clients, it would need to relax the clients' path selection strategies (described above) to consider paths that are "close enough" to the minimal cost. This relaxation would accommodate their current route selection strategies and allow the client to choose from multiple routes to amplify privacy.

6.2.2 Payment privacy quantification

Theorem 3 captures the composition of the arguments made in this subsection and quantifies them in differential privacy terms. It shows that a random selection of 1-out-of-r possible routes of length l in Twilight improves the differential privacy's ε proportionally to $\sqrt{r}$, and that ε impairs proportionally to $\sqrt{\log_b T}$ and $\sqrt{l}$.

Theorem 3. *Let r be the number of channel-disjoint routes with l inter-relay channels between Alice and Bob. Alice's m-coin payment to Bob is (ε,δ) differentially-private where:*

$$\varepsilon = \ln\left(1+\frac{mc\sqrt{l\log_b T}}{\sigma\sqrt{r}}+\frac{l\log_b T m^2}{2\sigma^2}\right),$$
$$\delta = 2\cdot GaussCDF(\tilde{\mu},\tilde{\sigma}^2;\ \tilde{\mu}-c\tilde{\sigma})$$
$$And,\ \tilde{\mu}=rl\log_b T\mu,\ \tilde{\sigma}^2=rl\log_b T\sigma^2$$

Proof. Given in Appendix D. □

To get a fair degree of privacy, $\sigma >> m$ (the variance in the noise hides a payment that Alice might make). In this case, ε is very close to $\frac{mc\sqrt{l\log_b T}}{\sigma\sqrt{r}}$ (i.e., Theorem 3 generalizes Theorem 2; it gives a similar result when $T,r,l=1$).

6.2.3 Multiple payments

Until now, we analyzed differential privacy for one payment. However, if Alice makes multiple sensitive payments, the attacker may try to learn about any of them. This scenario is known as composition in the differential privacy literature. Fundamentally, differential privacy deteriorates with the number of payments Alice wishes to hide, but the composed result remains differentially private. Maintaining the rigorous differential privacy guarantee, albeit with higher ε,δ, can be crucial. For example, a court that should be convinced "beyond a reasonable doubt" requires a very high degree of certainty, making even relatively high ε,δ guarantee valuable.

The literature also provides theorems for computing the ε,δ guarantee of such composition. This quantification is important. It allows users to avoid sensitive payments when exceeding a "privacy budget" (i.e., when ε,δ reflect a risk they deem too high). The most general result states that composing k invocations of an (ε_i,δ_i)-differentially private mechanism results in $(\sum_{i=1}^k\varepsilon_i,\sum_{i=1}^k\delta_i)$-differential privacy [11, Thm. 3.16]. Another, more powerful, composition theorem [21] holds when the noise in the composed invocations is independent, this result states that ε grows with $\sqrt{k}$. In our case, when multiple payments traverse the same route, different invocations of the noisy payment processing mechanism within the same channel are not independent (due Twilight's use of the tree), and thus only guarantee linear growth of ε. However, when payments traverse disjoint routes, the more advanced composition theorem holds, and ε degrades slower. This implies that Alice can increase her privacy level under multiple payments by opening channels to more relays. Intuitively, these channels allow Alice to use more disjoint paths, which helps her in two ways: First, randomizing over more paths provides better ε,δ to begin with (see §7.1 for example ε,δ values for different route lengths and number of available disjoint routes). Second, when paths are disjoint, the stronger composition results hold. Appendix D.1 analyzes this scenario.

7 The Cost of Privacy

The noise that a relay induces artificially reduces its channel's effective capacity: the relay might deny payments that

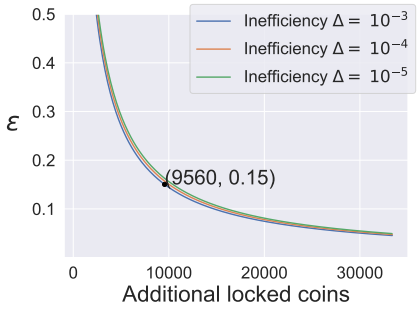


Figure 5: Privacy-efficiency trade-off for a channel with 2^{64} time-slots. Fixing $\delta = 10^{-7}$ and 1-coin payment. Locking more coins allows lower ε.

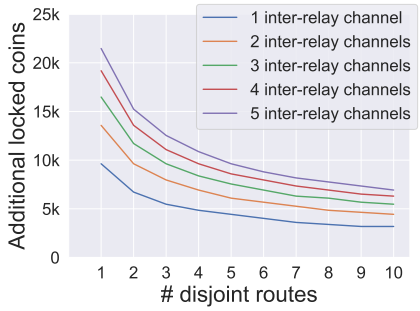


Figure 6: Extra locked coins needed to achieve $\varepsilon = 0.15, \delta = 10^{-7}$ differential privacy, with $\Delta = 10^{-3}$ inefficiency as a function of the number of disjoint routes.

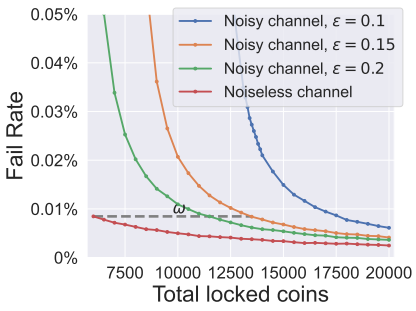


Figure 7: Payment failure rate on channels that provide different level of privacy.

spend amounts close to the channel's liquidity and lose the associated fees. In this section, we study this cost. We present a metric for a channel's ability to process a sequence of payments, illustrate the trade-off between privacy and efficiency, and quantify the cost of operating a Twilight relay.

Definition 1. *A noisy payment channel $\mathcal{C}$ is Δ-inefficient compared to a noiseless channel $\mathcal{C}'$ if any payment accepted through channel $\mathcal{C}'$, is rejected in $\mathcal{C}$ with probability at most Δ.*

We use the inefficiency metric to measure the cost of noisy channels. Theorem 4 quantifies the extra coins that a relay should lock to make the noisy channel Δ-inefficient compared to the noiseless alternative. Let $Gauss(\mu, \sigma^2)$ be the noise distribution for nodes in a relay's noise tree, and N the maximal number of nodes in the covering set for any time slot (Table 1).

Theorem 4. *If the noisy channel $\mathcal{C}$ is initialized with ω more coins in both directions than the noiseless channel $\mathcal{C}'$, then it is Δ-inefficient compared to $\mathcal{C}'$, where:*
$\omega = \mu N + t$, *and* $\Delta = GaussCDF(\mu N, \sigma^2 N;\ \mu N - t)$.

The knob value t allows trading a higher ω for a lower Δ.

Proof. Given in Appendix E. □

The Gauss distribution CDF at $\mu N - t$ falls very quickly with t (proportionally to e^{-t^2}). For example, achieving $\Delta = 0.1\%$ (with $\delta = 10^{-7}, \varepsilon = 0.15$ privacy) requires $\omega = 95608$; see Figure 5. Achieving $\Delta = 10^{-10}$ requires locking only 15% more coins (for the same privacy level, $\omega = 11000$).

Multi-payment sequences and multi-hop routes. The inefficiency metric composes for a sequence of n payments routed over l channels: Consider two sequences of l noisy and noiseless channels, where noisy channel i is Δ_i-inefficient compared to the i^{th} channel in the noiseless sequence. Using the union bound, we find that a sequence of payments that is accepted via the noiseless route might be rejected from the noisy route with a probability of at most $n\sum_{i=1}^{l}\Delta_i$.

7.1 Privacy-efficiency trade-off

Comparing Theorem 3 and Theorem 4, which summarize Twilight's privacy and efficiency properties, allows reasoning about the number of coins that a relay should lock in the channel's smart contract to support a certain level of privacy. Figure 5 illustrates the privacy-efficiency trade-off for one channel, trading higher ω for lower ε. Figure 6 then focuses on a particular privacy setting ($\varepsilon = 0.15, \delta = 10^{-7}$) and explores the trade-off for scenarios where clients have multiple options for disjoint routes, and these routes are of multiple hops. Since clients boost privacy by uniformly selecting routes (§6.2.1), a network that offers many disjoint routes improves the efficiency (requires locking fewer coins for the same privacy level). Appendix E illustrates the trade-off for other δ values.

Payment success ratio. Our analysis thus far refers to the chance that *any* payment fails. However, in practice, a channel may process payments back and forth. Some payments may even fail for over-spending on a noiseless channel, while they would succeed on the noisy channel due to the extra liquidity (ω in Theorem 4) and sufficiently low noise. We use simulation to compare the failure rate on a long sequence of payments. Our simulation, in Figure 7, focuses on a single channel, where we send 1-coin payments left or right with a uniform distribution. We run this simulation for three levels of privacy (values of ε, fixing $\delta = 10^{-7}$). Each data point is the failure rate on a sequence of 10^8 payments. In this simulation, payments route over a single payment channel without concurrency. We observe that as the channel's capacity grows, the noised channel's success ratio converges to that of the noiseless channel. We also see that the extra locked coins needed for achieving $\Delta = 10^{-3}$ on the noisy channel with $\varepsilon = 0.15$ and $\delta = 10^{-7}$ is around $7k$ over the noiseless channel (see dashed horizontal line in Figure 7). This is an improvement over the theoretical bound from Theorem 4 (which is illustrated in Figure 5).

7.2 Quantifying the relay's cost and incentives

Twilight requires relays to run code in TEEs and induce noise when processing payments. Modern commodity processors ship with TEEs (e.g., see Intel's i9 processor spec [18]). Thus, we believe that deploying a TEE should not incur a high cost on the relay's operator and focus on the cost of operating noisy payment processing, which increases a relay's locked coins.

Although Twilight's relays need to lock more coins than the noiseless alternative (say, $9.5k$ additional coins per Figure 5), these coins *return to the relays* when the channel closes, so relay operators only lose any potential interest that they could have gotten for these extra coins. Moreover, the financial impairment rate from locking coins is fixed, and *amortizes over all payments* carried during the channel's lifetime. Given these observations, we can compute the increase in relay fee that would cover the cost of operating a Twilight channel depending on the number of payments it carries. For example, consider the privacy-efficiency trade-off point highlighted in Figure 5 ($\varepsilon = 0.15, \delta = 10^{-7}, \Delta = 10^{-3}$); we find that given a yearly interest rate of 3%,a relay that handles 79 payments per day can cover its operational cost by charging 1% fee from each payment (see analysis in Appendix E) . Previous works on differential privacy (in other contexts) statistically modeled users' actions, treating them as noise (e.g., [6]). This approach reduces the number of coins a relay should lock, thus reducing Twilight's operational cost. However, it requires strong additional assumptions about users that Twilight avoids: that many users are honest and submit payments according to a known distribution.

8 Implementation

There are three components to our prototype of Twilight. The first is the smart contract for managing on-chain channel accounts between two parties, which we implement for Ethereum in Solidty [13] (68 lines of code). A limitation of our smart contract implementation is that it does not use shielded on-chain transactions (e.g., implemented using Solidity in [24]). It, therefore, exposes a channel's closing balance split (i.e., the aggregate amount of payments on the channel through its lifetime). We argue in §3.1 that for long-lived channels this is typically safe in practice. The second component is the enclave running in Intel's SGX [9] as a TEE. We implement the noisy payment processing logic inside the enclave in C++17 (965 lines of code). The last component is the relay, which calls the enclave and implements the networking logic for carrying payments across the route. We implement the relay in Python3.8 (886 lines of code). The clients use the same networking logic as the relays (but do not run noisy payment processing). Our implementation uses ChaCha20 [26] for symmetric encryption, elliptic curve `secp256r1` [39] for public key operations, and SHA3-Keccak [5] as hash function.

Inside the enclave, we generate the TEE's secret key using `sgx_ecc256_create_key_pair` and run the randomized response mechanism (§4.1). Running this mechanism requires the enclave to read the current time; we call `rdtsc` to get the number of CPU cycles since boot [19] which serves as a high-resolution clock. We detect reboots by reading the nonce that `sgx_get_trusted_time` returns, which is initialized at boot time (as discussed in §5.2). The channel's noise tree (Figure 2) has $T = 2^{64}$ time slots to support long-lived channels.

9 Evaluation

We use Twilight's prototype to evaluate the throughput and latency, and to measure the cost of resolving disputes on-chain (§9.1). We use simulations to evaluate the privacy benefits of noisy payment processing under partial adoption (§9.2).

9.1 Performance and cost

Setup. We deploy Twilight's implementation over a `Standard_DC1s_v2` machine type in Azure [8], which has one CPU and 4 GB RAM. This machine has a single `Intel SGX-1` TEE. We deploy Twilight on machines in two Azure regions, across two continents, `eastus` and `northeurope`. To ensure our tests experiment real Internet latencies, the routes for the payments we evaluate alternate between relays in both regions. We measure the average round-trip latency between machines in the two regions to be 84.89ms (with 0.58 standard deviation). In the following experiment results, each data point is an average of 40 repetitions. We use error bars to show the standard deviation from the mean.

Throughput. In Figure 8 we measure the rate of completed payments as a function of the rate of issued payments for different route lengths. The throughput continues to grow until around 820 resolved payments/sec, which is over twice of a relay's throughput in the Lightning PCN [20] (about 358 payments/sec with one relay). We attribute this performance improvement to batch-processing payments (cf., in Lightning, each payment requires expensive invocations of an underlying Bitcoin client). When the payment issuance rate exceeds 800 payments/sec, the relays' backlog grows and eventually causes the throughput to start decreasing. We, therefore, cap the relay's backlog at 3000 payments to allow handling bursty payments and avoid congestion collapse. Since PCNs are horizontally scalable, the more relays join Twilight, the more disjoint routes are available (proportionally growing Twilight's throughput).

Latency across a route. Figure 9 evaluates payment latency in Twilight by sending payments across paths of different lengths and using different payment issuance rates. After a minute, when the system is in steady-state, we measure the latency for completing payments. Before backlogs start forming in the relays, the latency is under 1.1s even with 4 relays (e.g., when 800/sec payments are issued). We attribute about half of this latency to the network RTT time (510ms from

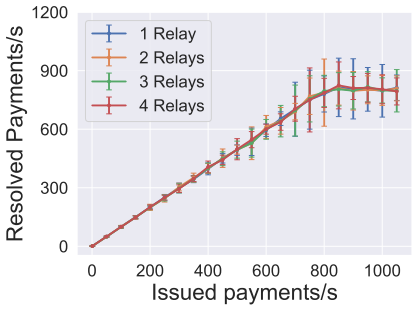


Figure 8: Throughput by payment issuance rate for different route lengths.

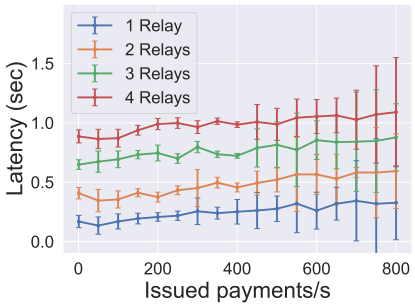


Figure 9: Latency by payment issuance rate for different route lengths.

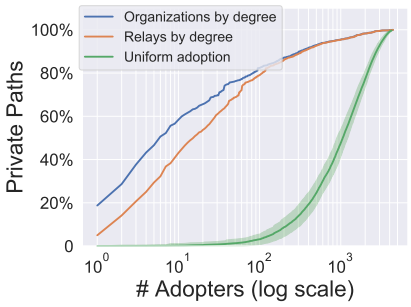


Figure 10: Privacy gain under partial noisy payment processing adoption.

Alice to Bob via the 4 relays). The rest is due to Twilight's processing costs inside the TEE. From this, we conclude that Twilight does not substantially increase the latency over a vanilla PCN (that routes payments via a similar number of hops). Even when payments are issued at a relatively high rate (e.g., 800 payments/sec) Twilight's processing does not dominate the latency over the network RTT.

Smart contract gas cost. We use Ganache [14] to evaluate the gas price of running the smart contract managing the channel's account on Ethereum. We gather the gas prices using "Web3.estimateGas" (an Ethereum API). The total gas cost for handling each disputed PTLC is less than 2.4k gas. The major pieces comprising this cost are: (1) 700 gas to validity check of the dispute, (2) 800 gas to decrypt the disputed PTLC, and (3) 900 gas to parse the plaintext and update the balance.

Each PTLC is encoded into 28B, which users can post to the smart contract to dispute the channel's closing balance split (along with the partner's signature over that PTLC and a matching secret key). This encoding comprises 8B for the amount, 4B for the PTLC's timeout (in blocks), and 16B for the authentication code that ensures the secret key is correct. Storing the PTLC on the blockchain costs 15k gas and dominates its processing cost.

9.2 Partial noisy processing adoption

Performing noisy payment processing involves costs for the relay (§7). We consider the effect of weakening Twilight's requirements through a compatibility mode where relays can participate in the network without noising payments in TEEs. Clients can tell which relays do noisy payment processing in TEEs using remote attestation and prefer routes where all relays do so. We evaluate the fraction of payer-payee pairs of nodes with a fully adopting path. As an example network, we use the Lightning network topology (snapshot from Nov. 7^{th}, 2021), the largest PCN today. We that assume the payer and payee run Twilight's client, and evaluate three scenarios regarding Twilight's adoption (in its full form) in the network: (1) relays adopt by descending order in the number of channels they have, (2) organizations (companies operating relays) adopt across all their relays by descending order in the number of channels they have (we use relay lists from [7, 29]), and (3) uniform random adoption across the network.

Figure 10 shows that if the largest 5 organizations in the network adopt noisy payment processing, then 47% of pairs of nodes in the network will have a connecting route with full adoption. The largest organization, "LN-BIG" [29] alone connects around 19% of the pairs. More generally, we see that adoption on a small number of relays or organizations at the core of the network can protect routes between a large portion of the network's nodes, giving a tangible path to significant privacy improvement. Uniform adoption is less effective, and several hundreds of relays should adopt before the benefit becomes significant.

10 Related Work

Several works point out that today's PCNs do not offer much privacy [15, 17, 30], and Kappos et al. evaluate such attacks in practice [22]. Bolt [15] gives a strong privacy guarantee by establishing payment channels over ZCash, but its architecture is restricted to just one relay. Namely, a hub that connects everyone in the network, which limits the scalability of the design and risks availability in the case that hub goes offline.

Malavolta et al. [30, 31] propose a privacy-preserving HTLC for PCNs. Instead of using the same HTLC secret for every hop, secrets across the route are cryptographically linked but appear random. Thus, malicious relays along the route cannot link payments through their HTLC secrets. This construction is compatible with Twilight's PTLCs and allows avoiding such "secret-correlation" attacks. SpeedyMurmurs [38] modifies the client's routing algorithm to split payments across several paths. In this manner, an attacker that does not control a relay on all routes cannot learn the exact payment amount or uniquely identify the payer and payee, but he can learn information about the users' "direction." As discussed in [31], these solutions [30, 31, 38] are only partial to the privacy problem in PCNs. In particular, since channels cannot transfer more funds than their liquidity, the attacker can

measure channels' liquidity on inter-relay links by requesting relays to carry payments. He can then correlate liquidity changes across channels to track the users' payments [22], the problem that Twilight tackles. Quantifying and bounding statistical information leakage to an active adversary that probes channels to deduce payment routes is challenging, which Twilight achieves through differential privacy.

Joancomarti et al. [17] show how malicious clients can monitor changes in channel liquidity. Tang et al. consider a PCN that continuously advertises channel liquidity with fresh noise [40]. They do not specify a noise mechanism or means to enforce it but show that the adversary can quickly learn the liquidity on every channel. Twilight addresses this problem by utilizing ideas from the differential privacy literature; it can avoid continuously publishing channel-liquidity with fresh noise since it is safe to reuse noise values when hiding the same payments set. This allows Twilight to provide a rigorous differential privacy guarantee for its users.

TEEchain uses TEEs to build a high-throughput PCN [28] and relies on the security of its relays' TEEs for integrity. In particular, there are no disputes since TEEs are trusted to close channels at the correct balance. Moreover, TEEchain does not protect against attacks on its users' privacy. In contrast, Twilight is focused on privacy and uses TEEs only to ensure relays perform randomized response to hide users' payments.

11 Conclusion

We presented Twilight, a new PCN that is focused on privacy. Twilight hides a user's payments from other users in the network using differential privacy. Relays convince users that they will hide their payments by leveraging TEEs. Our analysis shows that Twilight provides rigorous privacy and incurs moderate costs. We implemented Twilight and tested its performance across a route of relays in two continents and evaluated it under partial adoption using simulations. Our evaluation shows that it provides solid performance compared to Lightning, today's most popular PCN (providing no privacy), and gives a tangible path to payment privacy in PCNs.

Availability

Our code is available online along with instructions for reproducing the evaluation results, see link in [41].

Acknowledgments

We thank Adam D. Smith and Katrina Ligett for insightful discussions on differential privacy and its applications, and our shepherd Stefanie Roos. Yossi Gilad was partially supported by the Alon fellowship and the Hebrew University cyber security research center and a gift from Microsoft. Aviv Zohar, Maya Dotan, and Saar Tochner were partially supported by grants from the Israel Science Foundation (grants 1504/17 & 1443/21) and by a grant from the Hebrew University cyber security research center.

References

[1] 1ML. Lightning network statistics. https://1ml.com/statistics, 2020.

[2] Adil Ahmad, Kyungtae Kim, Muhammad Ihsanulhaq Sarfaraz, and Byoungyoung Lee. Obliviate: A data oblivious filesystem for intel sgx. In *NDSS*, 2018.

[3] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from Bitcoin. In *IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society, 2014.

[4] Iddo Bentov and Ranjit Kumaresan. How to use Bitcoin to design fair protocols. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO*, volume 8617 of *Lecture Notes in Computer Science*, pages 421–439. Springer, 2014.

[5] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The keccak sha-3 submission. *NIST*, 6(7):16, 2011.

[6] Raghav Bhaskar, Abhishek Bhowmick, Vipul Goyal, Srivatsan Laxman, and Abhradeep Thakurta. Noiseless database privacy. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT*, volume 7073 of *LNCS*, pages 215–232. Springer, 2011.

[7] Bitfinex. The nodes of bitfinex. https://ln.bitfinex.com/, 2022.

[8] Marshall Copeland, Julian Soh, Anthony Puca, Mike Manning, and David Gollob. Microsoft azure. *Apress: New York, NY, USA*, 2015.

[9] Victor Costan and Srinivas Devadas. Intel SGX explained. Report 2016/086, Cryptology ePrint Archive, February 2016.

[10] Cynthia Dwork, Moni Naor, Toniann Pitassi, and Guy N. Rothblum. Differential privacy under continual observation. In Leonard J. Schulman, editor, *STOC*, pages 715–724. ACM, 2010.

[11] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3–4):211–407, 2014.

[12] Shufan Fei, Zheng Yan, Wenxiu Ding, and Haomeng Xie. Security vulnerabilities of sgx and countermeasures: A survey. *ACM Computing Surveys*, 54(6):1–36, 2021.

[13] Ethereum Foundation. Solidity programming language. https://docs.soliditylang.org/en/latest/.

[14] Ganache. https://www.trufflesuite.com/docs/ganache/overview.

[15] Matthew Green and Ian Miers. Bolt: Anonymous payment channels for decentralized currencies. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *CCS*, pages 473–489. ACM, 2017.

[16] Greg Brockman. Stellar. https://stripe.com/blog/stellar, 2014.

[17] Jordi Herrera-Joancomartí, Guillermo Navarro-Arribas, Alejandro Ranchal Pedrosa, Cristina Pérez-Solà, and Joaquin Garcia-Alfaro. On the difficulty of hiding the balance of lightning network channels. In Steven D. Galbraith, Giovanni Russello, Willy Susilo, Dieter Gollmann, Engin Kirda, and Zhenkai Liang, editors, *AsiaCCS*, pages 602–612. ACM, 2019.

[18] Intel product specifications. https://ark.intel.com/content/www/us/en/ark/products/199332/intel-core-i910900k-processor-20m-cache-up-to-5-30-ghz.html. Accessed: 2021-12-21.

[19] Intel. SGX software developer's manual. https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3d-part-4-manual.pdf.

[20] Joost Jager. Lightning node performance: Exploring the path to 1 000 tps. https://bottlepay.com/blog/bitcoin-lightning-benchmarking-performance/.

[21] Peter Kairouz, Sewoong Oh, and Pramod Viswanath. The composition theorem for differential privacy. In *International conference on machine learning*, pages 1376–1385. PMLR, 2015.

[22] George Kappos, Haaroon Yousaf, Ania Piotrowska, Sanket Kanjalkar, Sergi Delgado-Segura, Andrew Miller, and Sarah Meiklejohn. An empirical analysis of privacy in the lightning network. In *Financial Cryptography*, 2021.

[23] George Kappos, Haaroon Yousaf, Ania Piotrowska, Sanket Kanjalkar, Sergi Delgado-Segura, Andrew Miller, and Sarah Meiklejohn. An empirical analysis of privacy in the lightning network. In *International Conference on Financial Cryptography and Data Security*, pages 167–186. Springer, 2021.

[24] Chaitanya Konda, Michael Connor, Duncan Westland, Quentin Drouot, and Paul Brody. Nightfall protocols for private transactions on the ethereum blockchain using zk-snarks. https://github.com/EYBlockchain/nightfall.

[25] Satwik Prabhu Kumble, Dick Epema, and Stefanie Roos. How lightning's routing diminishes its anonymity. In *The 16th International Conference on Availability, Reliability and Security*, pages 1–10, 2021.

[26] Adam Langley, W Chang, Nikos Mavrogiannopoulos, Joachim Strombergson, and Simon Josefsson. Chacha20-poly1305 cipher suites for transport layer security (tls). *RFC 7905*, 2016.

[27] Ninghui Li, Min Lyu, Dong Su, and Weining Yang. *Differential Privacy: From Theory to Practice*. Synthesis Lectures on Information Security, Privacy, & Trust. Morgan & Claypool Publishers, 2016.

[28] Joshua Lind, Oded Naor, Ittay Eyal, Florian Kelbert, Emin Gün Sirer, and Peter R. Pietzuch. Teechain: A secure payment network with asynchronous blockchain access. In Tim Brecht and Carey Williamson, editors, *SOSP*, pages 63–79. ACM, 2019.

[29] LN-BIG. The nodes of ln-big. https://lnbig.com/#/our-nodes, 2022.

[30] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Srivatsan Ravi. Concurrency and privacy with payment-channel networks. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *CCS*, pages 455–471. ACM, 2017.

[31] Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. Anonymous multi-hop locks for blockchain scalability and interoperability. In *NDSS*. The Internet Society, 2019.

[32] Ania M. Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. The loopix anonymity system. In Engin Kirda and Thomas Ristenpart, editors, *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017*, pages 1199–1216. USENIX Association, 2017.

[33] Joseph Poon and Thaddeus Dryja. The Bitcoin Lightning Network: Scalable Off-chain Instant Payments. https://lightning.network/lightning-network-paper.pdf, 2015.

[34] Wahbeh H. Qardaji, Weining Yang, and Ninghui Li. Understanding hierarchical methods for differentially private histograms. *Proc. VLDB Endow*, 6(14):1954–1965, 2013.

[35] Fergal Reid and Martin Harrigan. An analysis of anonymity in the Bitcoin system. In *SocialCom/PASSAT*, pages 1318–1326. IEEE Computer Society, 2011.

[36] Torkel Rogstad. Lightning network 101: Privacy. Medium post, 2019.

[37] Dorit Ron and Adi Shamir. Quantitative analysis of the full Bitcoin transaction graph. In Ahmad-Reza Sadeghi, editor, *Financial Cryptography*, volume 7859 of *Lecture Notes in Computer Science*, pages 6–24. Springer, 2013.

[38] Stefanie Roos, Pedro Moreno-Sanchez, Aniket Kate, and Ian Goldberg. Settling payments fast and private: Efficient decentralized routing for path-based transactions. In *NDSS*. The Internet Society, 2018.

[39] secp256r1. https://neuromancer.sk/std/secg/secp256r1.

[40] Weizhao Tang, Weina Wang, Giulia C. Fanti, and Sewoong Oh. Privacy-utility tradeoffs in routing cryptocurrency over payment channel networks. *Measurement and Analysis of Computing Systems*, 4(2):29:1–29:39, 2020.

[41] Saar Tochner, Maya Dotan, Aviv Zohar, and Yossi Gilad. Twilight prototype implementation. https://github.com/saart/Twilight, 2022.

[42] Saar Tochner, Aviv Zohar, and Stefan Schmid. Route hijacking and DoS in off-chain networks. In *AFT*, pages 228–240. ACM, 2020.

[43] Jiyong Yu, Lucas Hsiung, Mohamad El Hajj, and Christopher W. Fletcher. Data oblivious isa extensions for side channel-resistant and high performance computing. Cryptology ePrint Archive, Report 2018/808, 2018. https://ia.cr/2018/808.

Algorithm 1: Time-slot covering. The algorithm finds the minimal set of nodes in the noise tree that cover the time-slots preceding time t.

Result: $\mathcal{N}_t$, the minimal covering of $[0,t-1]$
Input: A channel's noise *tree* and t.
$\mathcal{N}_t \leftarrow \emptyset$
`/* traverse the tree and find the smallest set of nodes covering [0,t-1] */`
for $level \in [0, tree.depth]$ **do**
  if $allOnes(t[level : tree.depth])$ **then**
    `/* cover all subtree slots */`
    $\mathcal{N}_t$.add($channel.Tree[t[0 : level]]$)
    return $\mathcal{N}_t$
  else if $t[level] =$ "1" **then**
    `/* cover left child subtree */`
    $leftChildLabel \leftarrow t[0 : level] +$ "0"
    $\mathcal{N}_t$.add($channel.Tree[leftChildLabel]$)
end
return $\mathcal{N}_t$

A Minimal Time Slot Covering

A.1 Proof of correctness for algorithm 1.

We need to show that for each $0 \leq t \leq T$ the algorithm finds the minimal covering of time-slots $[0,t-1]$. Let $0 \leq t \leq T$, and let b_t be the binary representation of t.

We first show that it outputs a correct covering of $[0,t-1]$. Let C_{Alg} be the covering returned by algorithm 1. We prove that C_{Alg} is correct by induction on T. Let $0 \leq t \leq T$. If $T = 0$, then the algorithm returns the empty covering which is correct. Assume by hypothesis that the algorithm is correct for every $0 \leq T' < T$. If $t < \frac{T}{2}$, the algorithm turns to handle the left subtree in the first step (since the most significant bit in t is 0. From now on, the run of the algorithm on the last $log(T)-1$ digits of t is correct by the induction hypothesis. So C_{Alg} is a correct covering of $[0,t-1]$. Else, $\frac{T}{2} \leq t < T$ so the algorithm adds the root of the left subtree to the covering and turns to handle the right subtree (since the most significant of t is 1), this covers the interval $[0, \frac{T}{2}-1]$. The run of the algorithm now proceeds to the rest of b_t, i.e., excluding the first bit in the binary representation; denote this number t'. $b_{t'}$ is of length $log(T)-1$, and the subtree is of size $\frac{T}{2}$. From the induction hypothesis we have that the algorithm finds a correct covering of $[0,t']$ in the left subtree, which translates to a covering of $[\frac{T}{2}, t-1]$ in the whole tree. This is added to the first node in C_{Alg}, which covers the first half of the interval. Therefore, algorithm 1 is a correct covering of $[0,t-1]$.

We now show that C_{Alg} is minimal. Assume toward a contradiction that there exists some covering of $[0,t-1]$ which contains strictly less nodes than C_{Alg}, denote it C_{opt}. Since C_{Alg} is non optimal, there are two intervals that can be replaced with one. This translates to two node trees that can be replaced with a mutual ancestor. Therefore, at some iteration of the loop the algorithm turned to handle the right subtree without adding the left sibling to the covering. Instead it added the two descendants of that sibling. This is a contradiction to the operation of algorithm 1.

B Payment Protocol Details

This appendix provides a more detailed view of the protocol including precise definition of the channel's state, the payment protocol's messages, and the smart contract API. Finally we include a proof of payment security and show that all parties receive pay even if relays break their TEEs.

B.1 The Smart Contract

We now describe the smart contract that is used when opening and closing channels. The contract we use is largely similar to vanilla PCNs, with the main difference being that PTLC values are encrypted, and are thus handled a bit differently. The smart contract's pseudo-code is provided in algorithm 2.

As in regular PCNs, a relay R_i may choose to unilaterally close one of its channels by publishing the channel state to the blockchain (e.g., in case his counterpart no longer responds). His counterpart in the channel has a time window of τ blocks during which it can submit a newer signed state to the smart contract. The smart contract compares serial numbers and checks the TEEs' signatures to ensure that a rogue relay cannot use an old state.

Claiming a PTLC Once a channel closes successfully, each party may claim incoming PTLCs by revealing the appropriate secret s. The secret allows the smart contract to decrypt the amount hidden in the PTLC, and to transfer these amounts to the relay.

Claiming the rest of the funds After all PTLCs are claimed or expire, the remaining liquidity $L_{i,i+1}, L_{i+1,i}$ can be claimed. This liquidity cannot be claimed prior to expiration because some unknown amount of it is supposed to be locked in the s.

B.2 The Channel's State and PTLCs

Throughout this section we denote relay i by R_i and its TEE by E_i. The relay stores the channel's state. At any given time, the state of the channel between two relays is defined by the current liquidities on both sides of the channel, the unresolved PTLCs in the channel, a serial number and a timeout parameter (for resolving on-chain disputes). Formally, given two parties, R_1, R_2 we denote the state of the channel between them as:

$$S = (sn, L_{1,2}, L_{2,1}, ((PTLC_k)_{k=1}^{m}))$$

Algorithm 2: A channel's smart contract.

def *open_channel*(pk_1, pk_2)**:**
- /* pk_1, pk_2 are the channel peers' public keys. */
- state $\leftarrow \emptyset$, serial_num $\leftarrow -\infty$, timeout $\leftarrow \infty$
- $owner_1 \leftarrow pk_1$, $owner_2 \leftarrow pk_2$
- $liq_1 \leftarrow 0$, $liq_2 \leftarrow 0$, $ptlcs = \emptyset$

def *add_funds*(*owner, added_funds*)**:**
- /* added_funds is an amount of money attached to the transaction */
- **if** *owner* = $owner_1$ **then**
  - $liq_1 \leftarrow liq_1 + added\_funds$
- **else if** *owner* = $owner_2$ **then**
  - $liq_2 \leftarrow liq_2 + added\_funds$

def *close_channel*(*S*, Sig_1, Sig_2)**:**
- /* S is the channel state, Sig_1, Sig_2 are the signatures of the two peers. */
- $(sn, L_{1,2}, L_{2,1}, (PTLC_k)_{k=1}^n) = S$
- **require** check_sig((*S*, channel_ID), Sig_1, $owner_1$)
- **require** check_sig((*S*, channel_ID), Sig_2, $owner_2$)
- **require** $sn >$ serial_num
- **require block_height()** $\leq timeout$
- state $\leftarrow S$, serial_num $\leftarrow sn$,
- timeout $\leftarrow$ **block_height()** $+ \tau$,
- $liq_1 \leftarrow L_{1,2}$, $liq_2 \leftarrow L_{2,1}$, $ptlcs \leftarrow (PTLC_k)_{k=1}^n$

def *claim_ptlc*(*ind, secret*)**:**
- /* *ind* is the PTLC index and *secret* is the secret associated with it */
- $(enc, verify, t, direction) = ptlcs[ind]$
- **if** $timeout \leq$ **block_height()** $\leq timeout + t$ **then**
  - **require** $verify = Hash(secret)$
  - $x = decrypt(enc, secret)$
  - **del** $ptlcs[ind]$
  - $liq_{direction} \leftarrow liq_{direction} + x$
  - $liq_{\neg direction} \leftarrow liq_{\neg direction} - x$

def *claim_remaining_liquidity*(*owner, owner_sig*)**:**
- /* *owner* is the public key of the endpoint claiming funds, *owner_sig* is its signature */
- **require** check_sig(tx, owner_sig, owner)
- **foreach** $(enc, h, t, d) \in ptlcs$ **do**
  - **require block_height()** $> timeout + t$
- **end**
- **if** *owner* = $owner_1$ **then**
  - $send\_amount = liq_1$, $liq_1 \leftarrow 0$
- **else if** *owner* = $owner_2$ **then**
  - $send\_amount = liq_2$, $liq_2 \leftarrow 0$
- **else raise** Exception;
- **send_funds** $(send\_amount, owner)$

Where $L_{1,2}$ is the liquidity of the channel between relay R_1 and relay R_2 that currently belongs to R_1 (before the PTLCs are applied), m is the number of unresolved PTLCs in the channel, *sn* is the serial number.

We additionally associate with each channel a constant timeout for appeals, denoted τ, which is the amount of time (measured in block creation events) that the channel closure transaction can be appealed (this is agreed upon in advance by the two parties that are creating the channel, and not changed). Once the channel's smart contract is deployed, its address on the blockchain serves as a unique channel ID that we concatenate to channel related messages prior to signing them (to prevent message replay attacks across different channels).

Each PTLC in the channel state consists of the following:

$$PTLC_j = (Enc_{E_d, Hash(s_j)}(x_j), Hash(s_j), t, d)$$

where,

- $Enc_{E_d, Hash(s_j)}(x_j)$ is an encryption of the payment amount to the public keys of the next enclave and the public key derived from the payment secret s_j. This encryption is achieved via a non-malleable encryption with a fresh ephemeral symmetric key to hide the payment amount, to which we add two ciphertexts of that key: (1) under the next hop's TEE public key, and (2) under the PTLC's public key (derived from the secret).
- $d \in \{R_1, R_2\}$ denotes the direction of the PTLC (towards relay 1 or 2)
- t denotes the expiration of the PTLC (measured in blocks) after the transaction to close the channel is posted to the blockchain. After this timeout elapses, the funds revert back to their sender and can no longer be claimed in exchange for the secret.

As in the standard lightning protocol, timeouts should decrease along the route for the protocol to remain secure and trustless [33]. In order to be considered valid by the smart contract, the state must be signed by both parties.

There are two state transitions use by the protocol, each of which increments the serial number by one and changes the state:

- **Adding a PTLC:** Incorporating a new $PTLC_{n+1}$ (after receiving one from a previous hop):

$$(sn, L_{1,2}, L_{2,1}, (PTLC_k)_{k=1}^m) \rightarrow (sn+1, L_{1,2}, L_{2,1}, (PTLC_k)_{k=1}^{m+1}) \quad (4)$$

- **Resolving a PTLC:** Once the secret s_j for $PTLC_j$ propagates back along the path, we move the amount x_j from R_1 to R_2:

$$(sn, L_{1,2}, L_{2,1}, (PTLC_k)_{k=1}^m) \rightarrow (sn+1, L_{1,2} - x_j, L_{2,1} + x_j, (PTLC_i)_{i=1, i\neq j}^m) \quad (5)$$

A similar (symmetric) state transition for s in the other direction adds liquidity in the opposite direction.

Each relay maintains the latest view of the channel state, along with a signature on this state by its counterpart's TEE, and its own TEE, which is exactly the information that allows it to close the channel unilaterally.

B.3 Payments in the TEE Protocol

A payment in the TEE protocol consists of two main stages - (1) route establishment, in which PTLCs propagate forward towards the recipient (2) payment completion in which the PTLC's secret propagates back and the PTLC is removed from each channel along the path. The details of each of these parts is listed below.

We assume two clients Alice and Bob wish to transact via relays in the system. Alice picks a route

$$\text{Alice} \to R_1 \to \ldots \to R_k \to \text{Bob}.$$

We assume Alice is aware of the public keys of TEEs on this path (E_i is the public key of R_i's TEE). She informs Bob of the amount of money x she wishes to send. Bob picks a secret s and sends $Hash(s)$ to Alice. Next, Alice begins the route establishment process by sending a message to the first relay in the route. This message is similar to the one that later propagates between relays along the route.

B.3.1 Route establishment

We now describe the route establishment process from the perspective of a single relay R_i along the path. Prior to receiving a message from the previous relay, we assume R_i's incoming channel has state

$$S_{in} = (sn, L_{i-1,i}, L_{i,i-1}, (PTLC_k)_{k=1}^{m})$$

and its outgoing channel has state

$$S_{out} = (sn', L_{i,i+1}, L_{i+1,i}, (PTLC_k)_{k=1}^{m'})$$

The following sequence of events takes place at R_i:

1. R_i is notified of a new PTLC in the channel when it receives a message from R_{i-1}. The message includes The new payment $PTLC_{m+1}$, given this PTLC, the new incoming channel state S'_{in} can be deduced (by applying state transition 4 on S_{in} essentially adding $PTLC_{m+1}$ and incrementing the serial number).

   The message additionally includes a signature of E_{i-1} on the new channel state S'_{in} together with the channel ID, as well as an Onion encrypted payload that denotes the next hop (by noting the public key of the next hop's TEE E_{i+1}) and onion encrypts the rest of the route to Bob.

2. R_i passes to its TEE the new (proposed) channel state S'_{in}, the signature of the previous TEE on this state, the current outgoing channel state S_{out} and the onion encrypted payload that identifies the next hop.

3. The TEE verifies the signatures of the preceding TEE on on the incoming channel state (and that it was created by a TEE). It then decrypts the payment amount x, and verifies that it is consistently encrypted to the new PTLC's public key $Hash(s)$. If one of these checks fails, the TEE aborts.

4. The TEE then draws noise according to the noise addition protocol in algorithm 1, and performs a liquidity check with this extra noise with the balance of the incoming channel's liquidity (here the TEE provides the algorithm with time from its internal clock, and a seed for a pseudo-random number generation that determines the noise at each node of the tree within the algorithm). If the liquidity check passes, it sets the amount x as it was. Otherwise, it sets x to be 0 and proceeds.

5. The TEE decrypts the onion routing message and discovers the public key of the next hop's TEE E_{i+1}

6. The TEE encrypts the payment amount x again (this time for the next hop's key E_{i+1}, and for $Hash(s)$ using the scheme discussed above). Using this encrypted value, it forms the new PTLC of the outgoing channel and derives from it the new outgoing channel state S'_{out}.

7. The TEE signs both states together with the correstponding channel ID: (S'_{in}, ID_{in}), and (S'_{out}, ID_{out}).

8. Finally, the TEE outputs: the identity of the next hop, the signature the states, and the new PTLC for the outgoing channel.

9. Relay R_i then forwards to the next hop R_{i+1} the following data items in a message (similar to the one it received from its predecessor): the new PTLC on the channel between R_i and R_{i+1}, the signature of E_i on the new channel state and its ID, and the onion encrypted route.

We note that since Alice does not herself own a TEE, the first relay's onion encrypted message also contains a bit notifying it that it is the first relay, and that it does not need to ensure that the incoming PTLC is signed by another TEE, but rather by a client's public key.

The last public key in the route is simply Bob's public key, and he can thus decrypt the hidden amount himself (Twilight does not assume that Alice or Bob have TEEs).

B.3.2 Payment completion

Upon the last $PTLC$ reaching Bob, he checks the amount is as promised by Alice, and validates the signature of the previous

relay's TEE. If these are valid he proceeds to send the secret s to the previous relay.

We describe the steps taken by relay R_i upon receiving the secret s pertaining to some $PTLC$:

1. Relay R_i decrypts the amount in the PTLC. If decryption was unsuccesful, it aborts. Otherwise it proceeds as follows.
2. R_i forwards the secret s to the previous relay R_{i-1} in the corresponding route.
3. R_i then applies state transition 5 to its outgoing channel to remove the PTLC. This also adjusts the channel liquidity. The new state needs to be signed by R_i's TEE. The TEE generates this signature only if it is provided with the old state, the secret, and its own signature on the old state (to ensure that the relay is requesting a signature on some arbitrary state).
4. R_i sends a message to R_{i+1} notifying it that it accepted the secret and provides the signature on the new state of the channel (without the PTLC, and with the adjusted liquidity).
5. R_i expects to receive a message from R_{i-1} removing the PTLC from the channel. If no such message is sent after some sufficient timeout elapsed, the channel should be unilaterally closed.

Once the secret reaches Alice, the payment is complete, and s acts as proof of payment.

B.4 PCN Integrity Despite Broken TEEs

Theorem 5. *Consider a payment from Alice to Bob via Twilight's relays. Twilight provides the following guarantee to each participant, even if all other participants collude against them and all relays break their TEEs:*

- *If Bob provides a receipt, it gets Alice's pay.*
- *If Alice pays the first relay, then she has Bob's receipt.*
- *An honest relay that pays on its outgoing channel, receives at least as much pay on its incoming channel.*
- *Any party can unilaterally close their channels and claim their funds on-chain at any time.*

Proof. If Alice pays money, either she has agreed to teardown the PTLC (which she only does upon receiving the secret s) or the first relay closed the channel and submitted the current state to the smart contract (if an old state was used Alice would appeal and win with a newer state). This state includes the PTLC that was not resolved. To claim these funds, the relay submits the secret s to the blockchain which means that Alice learns the secret from the blockchain.

Bob only releases the secret s upon receiving a PTLC from the last relay which is included in a state signed by the relay's TEE. Bob also checks that the amount encrypted in this PTLC matches the amount that Alice was supposed to send. This means that one of two options can happen: (1) either the last relay tears down the $PTLC$ and finalizes the payment, or if the PTLC is about to expire, (2) Bob submits the updated state containing the PTLC and the secret s to the blockchain, and is awarded the funds.

Now any relay along the path can always obtain any funds it loses on an outgoing link because it only sends and signs an PTLC on the outgoing link once it receives a valid one on the incoming link. If a relay is honest, then it uses the same payment public-key in the outgoing PTLC, the same encrypted transfer amount (or lower if it failed the liquidity check). Thus, the outgoing PTLC is valid only if the incoming one is valid, and is always of an amount that is sufficient to cover the outgoing payment.

Then, the relay only agrees to complete the payment on its outgoing edge upon receiving the secret s which allows it to claim the funds. Either by getting the previous relay on the path to update the state of the channel (upon receiving s) or by closing the channel and directly submitting s to the smart contract. □

C Extension for Recovering TEE's Time

Twilight supports a simple protocol that allows convincing one "source" TEE that another "replica" TEE kept its time in case of a reboot. First, the two TEEs exchange public keys, these keys are authenticated using remote attestation. Next, the replica sends a challenge to the source and measures the time until it receives a response which contains a signed timestamp from the source with the replica's challenge. If more than $\alpha/2$ time elapses, it rejects the response. Otherwise, the replica starts counting time since it receives the source's timestamp.

When the source reboots, and wishes to recover it runs a similar protocol in the reverse direction. It asks the replica for its time along with a challenge, and if it receives a signed response from the replica's TEE within less than $\alpha/2$ time, it sets its clock according to the response and then waits α time before accepting more requests to process payments from the hosting relay. The reason for waiting α time is to ensure that the relay cannot shift time backward by stalling messages for $\alpha/2$ in each direction. We envision α be on the order of a few seconds to a minute.

D Privacy Analysis

We provide the proof for the Theorem 3 reasoning about Twilight's privacy guarantees given in §4. Our proof uses a probabilistic bound, which hold except when drawing extreme val-

ues for noise. We quantify these probabilities with $\delta_{\text{left}}, \delta_{\text{right}}$, which mark the probabilities of obtaining extremely low or high values, so the differential privacy $\delta = \delta_{left} + \delta_{right}$.

Proof of Theorem 3. For each l-channels route r_i of the r available routes, consider the attacker's observations, o_i, on the route's channels over the lifetime of Twilight's operation. The attacker observes all channels along all routes, that is, he sees the vector $o_1, \ldots, o_r$. We are interested in bounding the following ratio:

$$\frac{Pr[o_1, \ldots, o_r | A \to B]}{Pr[o_1, \ldots, o_r | \mathsf{X}]} \tag{6}$$

Alice's client selects one route at random, hence:

$$= \frac{1}{r} \sum_i \frac{Pr[o_1, \ldots, o_r | A \to B \text{ via } i]}{Pr[o_1, \ldots, o_r | \mathsf{X}]}$$

Since all routes are disjoint and induce i.i.d. noise,

$$= \frac{1}{r} \sum_i \frac{\prod_j Pr[o_j | A \to B \text{ via } i]}{\prod_j Pr[o_j | \mathsf{X}]}$$
$$= \frac{1}{r} \sum_i \frac{Pr[o_i | A \to B \text{ via } i] \prod_{j \neq i} Pr[o_j | \mathsf{X}]}{\prod_j Pr[o_j | \mathsf{X}]}$$

Denote by $x_{i,k,t}$ the noise value that relay k in route i draws for ancestor t in the noise tree of Alice's transaction time slot. $x_{i,k,t}$ distributes $Gauss(\mu, \sigma^2)$. Let m be Alice's payment amount. Using the calculation from Equation 3 ($\log_b T$ is the number of ancestors of a payment slot in the tree):

$$= \frac{1}{r} \sum_{i=1}^{r} \frac{e^{\frac{\sum_{k=1}^{l} \sum_{t=1}^{\log_b T} 2m x_{i,k,t} - m^2 - 2m\mu}{2\sigma^2}} Pr[o_i | \mathsf{X}] \prod_{j \neq i} Pr[o_j | \mathsf{X}]}{\prod_j Pr[o_j | \mathsf{X}]}$$
$$= \frac{1}{r} \sum_{i=1}^{r} \frac{e^{\frac{\sum_{k=1}^{l} \sum_{t=1}^{\log_b T} 2m x_{i,k,t} - m^2 - 2m\mu}{2\sigma^2}} \prod_j Pr[o_j | \mathsf{X}]}{\prod_j Pr[o_j | \mathsf{X}]}$$
$$= \frac{1}{r} \sum_{i=1}^{r} e^{\frac{\sum_{k=1}^{l} \sum_{t=1}^{\log_b T} 2m x_{i,k,t} - m^2 - 2m\mu}{2\sigma^2}} \tag{7}$$

For any α, $e^{\alpha} \geq 1 + \alpha$, therefore:

$$\geq \frac{1}{r} \sum_{i=1}^{r} 1 + \frac{\sum_{k=1}^{l} \sum_{t=1}^{\log_b T} 2m x_{i,k,t} - m^2 - 2m\mu}{2\sigma^2}$$
$$= 1 + \frac{-rl \log_b T m^2 - 2mrl \log_b T \mu + 2m \sum_{i=1}^{r} \sum_{k=1}^{l} \sum_{t=1}^{\log_b T} x_{i,k,t}}{2r\sigma^2}$$

Similarly to the probabilistic bounds for Theorem 2, we upper bound $\sum_{i=1}^{r} \sum_{k=1}^{l} \sum_{t=1}^{\log_b T} x_{i,k,t}$ which distributes $\text{Gauss}(rl \log_b T \mu, rl \log_b T \mu \sigma^2)$ with $\sum_{i=1}^{r} \sum_{k=1}^{l} \sum_{t=1}^{\log_b T} x_{i,k,t} \leq rl \log_b T \mu + c\sqrt{rl \log_b T}\sigma$ from below, which holds except with probability $\delta = GaussCDF(rl \log_b T \mu, rl \log_b T \mu \sigma^2; rl \log_b T \mu - c\sqrt{rl \log_b T}\sigma)$ and get:

$$\geq 1 + \frac{-rl \log_b T m^2 - 2mrl \log_b T \mu + 2mrl \log_b T \mu - c\sqrt{2mrl \log_b T}\sigma}{2r\sigma^2}$$
$$= 1 - \left(\frac{mc\sqrt{l \log_b T}}{\sigma\sqrt{r}} + \frac{l \log_b T m^2}{2\sigma^2}\right) = e^{-\varepsilon}$$

To obtain the upper bound on e^{ε} we use a related inequality, for $\alpha < 1.79$, $e^{\alpha} \leq 1 + \alpha + \alpha^2$ and apply it to Equation 7. In our case, $\alpha_i = \frac{\sum_{k=1}^{l} \sum_{t=1}^{\log_b T} 2m x_{i,k,t} - m^2 - 2m\mu}{2\sigma^2}$ Notice that α_i is a linear combination of normally distributed random variables and therefore, it is also a normally distributed variable. We upper bound $\sum_i \alpha_i$ using the Gauss CDF as we have shown before. The sum $\sum_{i=1}^{r} \alpha_i^2$ is a sum of i.i.d normally distributed random variables and therefore follows the chi-squared distribution with r degrees of freedom. We probabilistically upper bound $\sum_{i=1}^{r} \alpha_i^2$ using the chi-squared distribution CDF. The result is smaller than the upper bound on $\sum_i 1 + \alpha_i$.[1] □

To get an intuition about the behaviour of ε, for small values it holds that $\varepsilon \approx \frac{mc\sqrt{l \log_b T}}{\sigma\sqrt{r}} + \frac{l \log_b T m^2}{2\sigma^2}$.

D.1 Privacy Analysis of Multiple Payment

We next analyze the intuitive arguments from section 6.2.3, that is, that taking advantage of more disjoint routes not only gives a better ε for one payment (Theorem 2), it also gives benefits in composition of multiple payments. Assume that there are r disjoint routes of length l between Alice and Bob and that Alice routes k payments to Bob. Denote by s the maximal number of payments routed through any one path and by ε', δ' the privacy for a single payment that arises from Theorem 3. Next, we statistically bound s as follows: for each payment, Alice's client chooses the route uniformly at random so it chooses each route with probability $1/r$. The chance that it chooses the i^{th} route for more than s times over the k payments is $1 - BinomialCDF(k, \frac{1}{r}; s)$ which quickly goes to 0 as s grows above $\frac{k}{s}$ (the average number of payments in one route). Across all routes, the chance that any single route has more than s payments is thus bounded by $\hat{\delta} = r(1 - BinomialCDF(k, \frac{1}{r}; s))$.

This means that for each route, the privacy composition is linear, achieving $s\varepsilon', s\delta' + \hat{\delta}$ differential privacy for all s payments. Since all r paths are disjoint, we can utilize the advanced composition theorem from [21] across the r differentially-private sequences of s payments.

[1] The exponent α must be less than 1.79 to use the $e^{\alpha} \leq 1 + \alpha + \alpha^2$ inequality. Since the Gauss distribution is additive, α distributes Gauss with a negative mean ($\frac{-l \log_b T m^2}{2\sigma^2}$) and low variance ($\frac{lN}{4\sigma^2}$). We add the probability that it does not hold, $1 - GaussCDF(\frac{lN}{2\sigma^2}, \frac{lN}{4\sigma^2}; 1.79)$, to the differential privacy delta. (For reasonable parameters such as $\sigma = 100, \log_b T \leq 64, l \leq 5$ it is extremely small, less than 10^{-14}.)

E Efficiency Analysis

We next prove Theorem 4 and analyze the cost for operating a Twilight channel.

E.1 Proof of Theorem 4

We begin by proving a lemma that aids in the proof.

Lemma 1. *Let c_1, c_2 be two bi-directional noiseless channels with capacities C_1, C_2 respectively. Assume that in c_1, c_2 the liquidity from left to right is L_1, L_2, and from right to left is R_1, R_2 respectively. So $C_1 = R_1 + L_1, C_2 = R_2 + L_2$. Assume further that $R_2 \geq R_1$ and $L_2 \geq L_1$ and that $C_2 = C_1 + \omega$ for some $\omega > 0$ (this means that at least one of the inequalities is strong). Let $P = p_1, \ldots, p_n$ be a sequence of payments that are approved in c_1. Then $p_1, \ldots, p_n$ are approved in c_2.*

Proof. Let $p_+ = \{p_i \in P | p_i > 0\}$ and let $p_- = \{p_i \in P | p_i < 0\}$ (p_+ are all payments moving money left to right, and p_- right to left). Assume towards a contradiction that there is some payment $p_i \in P$ that is rejected in c_2. Now, since p_i was rejected by c_2 if $p_i \in p_+$ then $L_2 - \sum_{j=1}^{i-1} p_j \geq 0$ but $L_2 - \sum_{j=1}^{i} p_j < 0$. However, since p_i was approved in c_1 it holds that $L_1 - \sum_{j=1}^{i} p_j \geq 0$. But it also holds that $L_2 > L_1$ which is a contradiction. The proof for the case $p_i \in p_-$ is identical.

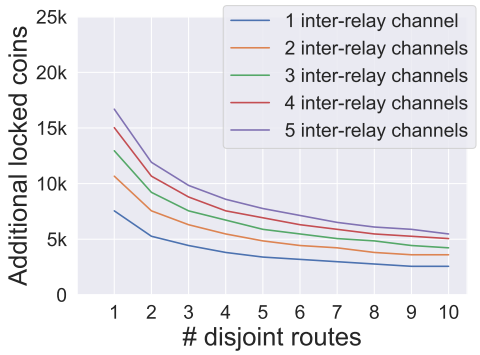
Therefore, it holds that for all $j \in [n]$ $\sum_{i=1}^{j} p_j = \sum_{i \leq j, p_i \in p_+} p_i - \sum_{i \leq j, p_i \in p_-} p_i < C_1 < C_2$. So $p_1, \ldots, p_n$ will be approved in C_2. □

From now on we will assume that all channels are bidirectional, and that if $C_2 = C_1 + \omega$ for two channels c_1, c_2 then it means that in the initialization of the channel, it holds that $L_2 \geq L_1$ and $R_2 \geq R_1$. Since generalizing the unidirectional proofs to the bidirectional case is always done as in Lemma 1, we will forgo this and prove the unidirectional case.

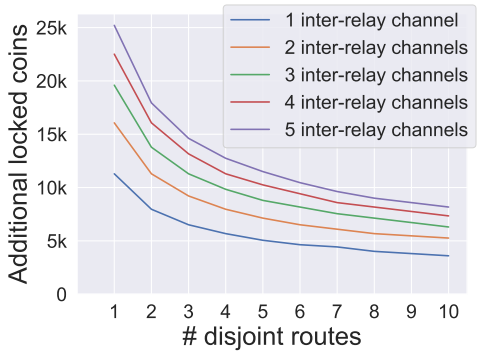
Proof of Theorem 4. From Lemma 1 we know that if we add capacity $\omega = \mu N + t$ to a channel, and if the total amount of added noise by the noise mechanism is no more than ω, then any payment that would have been approved without noising will also be approved after noising with the added capacity. The probability that the mechanism adds more than ω noise is $1 - \text{GaussCDF}(\mu N, \sigma^2 N; \omega) = 1 - \text{GaussCDF}(\mu N, \sigma^2 N; \mu N + t) = \text{GaussCDF}(\mu N, \sigma^2 N; \mu N - t)$. This probability, therefore, bounds the chance that the noisy channel rejects a payment that the noiseless channel approves.

E.2 Efficiency Analysis Illustrated

Extending the analysis in Section 7.1, we present figure 12 which illustrate the trade-off between efficiency to privacy for additional values of δ, as well as figure 11 which presents the effect of the number of disjoint routes and the route length



(a) Privacy $\delta = 10^{-5}$



(b) Privacy $\delta = 10^{-9}$

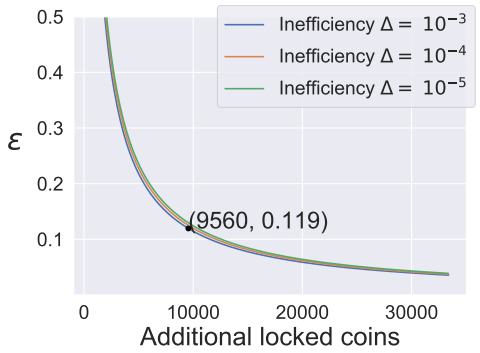
Figure 11: The number of extra coins locked needed to achieve a privacy level of $\varepsilon = 0.15$ and $\Delta = 10^{-3}$ as a function of the number of disjoint routes. Extending the results from Figure 6 to more values of delta.

on the overhead of escrow needed to achieve a privacy level of $\varepsilon = 0.15$, for more values of δ. The figures show the same trends as illustrated in figures 5 and 6 in the main body of this paper.

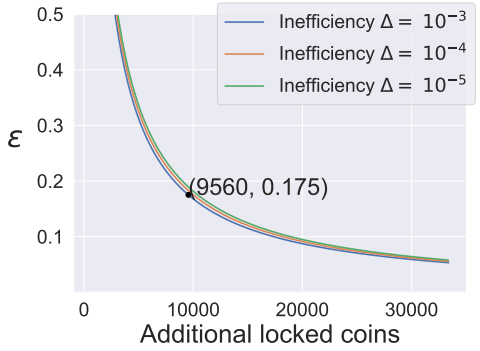
E.3 Quantifying and covering operation costs

The trade-off between privacy and efficiency (§7.1) quantifies the extra coins a Twilight relay's operator locks for a given level of privacy (ε, δ) and payment failure probability (Δ). In financial terms, since the relay's operator receives back the locked coins, their deposit loses value at the inflation rate. Since the deposit amount is independent of the number of payments over a channel, a relay may cover this cost through fees. Next, we evaluate the payment volume that would cover Twilight's operational costs.

Denote the average daily payment volume on the relay's channel by v coins. Let $f \in [0,1]$ be the payment fee, which is a portion of the payment amount that is split evenly across the route. Let $i \in [0,1]$ be the daily inflation rate and assume that a relay locks $M+$ the amount of coins m in the payments it



(a) Privacy $\delta = 10^{-5}$



(b) Privacy $\delta = 10^{-9}$

Figure 12: Privacy-efficiency trade-off, extending results from Figure 5 to more values of δ.

aims to hide. The daily cost for operating a Twilight channel is $\frac{(Mm)i}{365}$ coins. Comparing the network's income and expense in the two equations above, we get that a channel profits it charges fees of at least:

$$vf > \frac{(Mm)i}{365} \Rightarrow f > \frac{(Mm)i}{365v} \tag{8}$$

Consider the privacy-efficiency trade-off point highlighted in Figure 5 ($\varepsilon = 0.15, \delta = 10^{-7}, \Delta = 10^{-3}$). Here, a relay hiding a m-coin payment needs to add to the escrow around $9560 \times m$ coins. With a typical inflation rate of $i = 3\%$, we get: $f > \frac{0.79}{v} m$. Thus, a payment channel that, on average, carries at least $v = 79 \times m$ every day (i.e., $79\times$ the payment amount it wishes to hide), covers the cost of operating Twilight in this privacy configuration by charging 1% payment fee.