Techniques for Privacy Over the Interledger

Akash Khosla, Vedant Saran, Nick Zoghb

University of California at Berkeley
Berkeley, CA, 94720, USA

December 12, 2018

Abstract

The Interledger Protocol (ILP) allows for lightweight settlements across differently situated payment systems. However, it reveals all transaction information (sender address, receiver address, transaction amount) to on and off path observers. This paper presents ILP-CEPA, an application layer on top of ILP that uses Tor-style onion routing to obscure this information. It works without modifications to Interledger's underlying protocols, and requires no additional coordination between nodes. Our work shows that ILP-CEPA provides a number of security guarantees, including the one aforementioned, at an acceptably low performance overhead.

1 Introduction

The Interledger Protocol (ILP) is a multi-hop streaming network protocol allowing for micropayments and interoperability between different ledgers. It makes miniscule tradeoffs in security to allow for lightweight settlements (unlike the Lightning Network). ILP can be used for more than just cryptocurrency, but for this scenario, we will focus on its integration with cryptocurrencies.

When a sender sends value to a receiver over the Interledger, the sender connects to the Interledger network sets up a payment channel with a connector, which are analogous to packet routers over a network. Connectors are payment facilitators that assist from getting from various senders to receivers without having to open individual payment channels with each receiver as a sender. They also facilitate atomic currency swaps based on an exchange rate.

Privacy concerns arise because of ILP's packet design in the STREAM protocol. Specifically, connectors are responsible for large flows of smaller values facilitated through STREAM and have access to large datasets from packet analysis on their connected peers. In an Interledger packet, the destination, source address, value, expiration and data are all visible to intermediary connectors, who are responsible for forwarding value from sender to receiver.

With this information, an Interledger connector already knows the sender and receiver. Ideally there should be a way to avoid being victim to a reveal of network topology of all ILP value transfers that link payments with individuals.

1.1 Motivation

The original proposal for this project was conceived by Evan Schwartz, co-inventor of the Interledger Protocol. While the base ILP construction is already deployed and facilitating hundreds of thousands micropayments per day (which is actually hundreds of dollars), there are many payment-related use cases that require additional privacy guarantees in order to be feasible. This commonly manifests itself in scenarios where senders/receivers want to minimize what a passive observer in the network is able to learn about transactions. For example, corporations may want to hide their balance sheet to protect the confidentiality of customers; individuals may want to protect themselves from mass data mining; entities executing large financial transactions may prefer to transact without having to worry about information leaks and the corresponding effects on the market.

The goal of this project, dubbed ILP-CEPA, is to conduct research into the ways in which peer-to-peer payment networks are anonymized and, more specifically, to produce a working implementation of one such solution for ILP. Although these goals could be fulfilled in a multitude of ways, the most pragmatic — given the semesters limited timeframe — was to implement an onion-routed overlay transport protocol as well as a wrapper encoding for base ILP packets that enable onion routing.

2 Related Work

To our knowledge, ILP-CEPA is the first attempt to implement anonymity on ILP. Due to ILPs conceptual similarity to the TCP/IP stack (see Figure 1), implementing

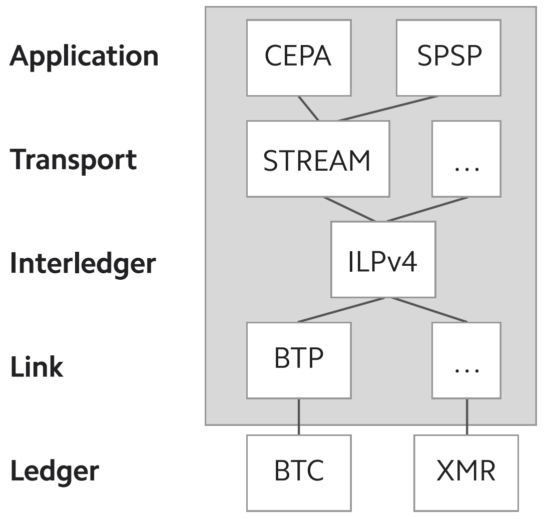

Figure 1: The Interledger Protocol stack. Our solution, CEPA, is implemented as an application layer solution on top of STREAM.

some form of network privacy such as onion routing (or simulating a VPN construction) seemed like a promising technique to test base anonymity guarantees.

Cryptocurrencies like Bitcoin and Ethereum are pseudo-anonymous and all current onboardings from fiat currencies are subject to *Know Your Customer* (KYC) policies and *Anti Money Laundering* (AML) regulation. Additionally, systematic monitoring of Bitcoin's unencrypted peer-to-peer network and analysis [3] of the blockchain can reveal who is using Bitcoin and for what purposes. Existent credit networks such as Ripple and Stellar are also susceptible to the same problems.

From a business and privacy perspective, having leakage on the base layer is bad. The traceability of each coin also puts the fungibility of all coins on a ledger at risk. We believe some of these problems can be mitigated through the use of a privacy layer over Interledger.

Privacy Ledgers Many proposals have been put forward for work on anonymity-focused blockchain ledgers. Complex zero knowledge proofs are in use across a number of projects [15] including ZCash, Monero and similar works. A separate line of works seek to increase the anonymity of Bitcoin by mixing transactions (e.g. CoinJoin, CoinShuffle, CoinSwap). Like Bitcoin, each of these constructions require that all transactions are stored on the blockchain. These solutions are all implemented as their own system, and direct interoperability between them is impossible without any coordination layer.

Interoperability Solutions ILP also falls under this category as it facilitates the transfer of value across multiple ledgers through an atomic-swap-like construction. Comparative work includes the Cosmos Network (and Polkadot), which leverages a construction similar to that of BTC-Relay known as Inter-Blockchain Communication (IBC) in which each Cosmos chain is a light client of every chain its connected to. Cosmos in this sense is capable of doing generic state transitions across chains (e.g. smart contract calls), but is a heavier solution that is not optimal for value transfers.

Payment Channels Networks Payment Channel Networks (PCNs) are network overlay solutions that results in parties transacting off-chain. The Lightning Network (LN) [11] operates as a network of bidirectional payment channels transferring value out of band. The functionality is carried out by Bitcoin scripts that are included in transactions, signed through third-party clients run by users. PCNs do not require transactions on-chain beyond the opening (and closing, if desired) of payment channels.

CEPA is primarily inspired by Lightning-Onion [5, 6, 12], the Lightning Networks native onion routing implementation, with many similar objectives and security considerations. Notably, the Lighting Networkss base payment channels operate differently than ILPs and, as such, require different considerations. For example, on the Lightning Network, on-path transactions share a Hashed Time-Lock Contract (HTLC) pre-image for the entire process, which makes their onion-routing implementation especially prone to collusion attacks. Additionally, when the HTLC is revealed on-chain for a payment channel close, then a single node on the payment path can match the pre-image to the specific transaction and figure out the sender and receiver of the money. Our implementation, CEPA, is also vulnerable to colluding parties, but not to the same degree as the Lightning Network. Since ILP's STREAM layer operates without HTLCs, the vector for collusion attacks on CEPA resembles that of Tor, which is in general harder to execute than on Lightning.

Tor and Cryptocurrencies Several works have proposed routing the native networking protocols of various cryptocurrencies over the Tor network. Many issues have come up in the course of exploring this solution trajectory. One such example is a vulnerability exposed when Tor is used in conjunction with Bitcoin [1] which involves the manipulation of Bitcoins anti-DoS networking rules. Therefore, we opted to implement a Tor-style onion routing protocol on top of native ILP protocols, instead of routing ILP packets through the existing Tor network. This eliminates the vulnerabilities presented in these works.

3 Background and ILP

ILP encapsulates a suite of multiple different protocols building off of one another to form the full payment network stack. Its goal is to consolidate many disparate payment systems. Connectors act as the nodes on the network which make sure packets are routed and forwarded to end recipients correctly. Their incentive is being able to generate incremental revenue via currency bid-ask spread. The bulk of the components that make up ILP are:

Bilateral Transfer Protocol (BTP). The base communication layer between peers. Defines a link-layer protocol that communicates arbitrary information over secure WebSocket and is compatible with a wide range of underlying ledgers through the usage of Plugins.

ILPv4. The titular core protocol (similar to IP) responsible for the construction of packets including safety measures like expiry times (equivalent to traditional TTL), and demarcating currency identifiers (e.g. XRP, BTC, etc.) and amounts. Key to this component are establishing unconditional payment channels (contrast this to LNs conditional payment channels which use Hash Time-Locked Contracts, or HTLCs); these take the form of signed claims against funds held on a ledger.

STREAM. The primary transfer protocol (similar to TCP or UDP). This layer enables packets containing a cryptographic condition whose fulfillment is only known to the recipient to be relayed across a path. If everything goes well and the receiver wants the funds, the receiver provides the fulfillment, which triggers the funds to move between each connector in the payment path. This way, the fulfillment proves that the money was delivered to the intended recipient and also acts as a request to activate payment. Inspired by QUIC, the protocol multiplexes multiple streams of packets over the same connection. STREAM ensures end-to-end delivery over multiple ILP connectors.

Simple Payment Setup Protocol (SPSP). STREAM does not specify how payment details, such as the ILP address or shared secret, should be exchanged between the sender and receiver. SPSP is a minimal protocol that uses HTTPS for communicating these details.

Interledger Dynamic Configuration Protocol (ILDCP). The address configuration protocol (similar to DHCP). Assigns hierarchical addresses to nodes in a parent-child fashion, for example: `g.us.acmebank.acmecorp.sales.199` may refer to a connector held by the sales division of acmebank in the US.

Our system, ILP-CEPA, is implemented as an application layer that sits on top of STREAM.

4 Security

4.1 Adversary and Threat Model

The protocol makes the following assumptions about network adversaries:

1. Off-route passive observers can constantly monitor traffic from every on-route node.
2. Every ILP node can retain message information for an indefinite amount of time.
3. ILP nodes may be controlled by adversaries, but it is statistically unlikely that all nodes along the path are controlled by the same adversary.

4.2 Security Guarantees

Under the above assumptions about adversaries, we require our solution to guarantee the following properties of the system:

1. Participants in an onion-routed circuit don't know their exact position within the circuit.
2. Participants within an onion-routed circuit do not know the identity of any other participants in the circuit, except for their neighboring connectors.
3. Participants cannot distinguish the sender of the payment, nor the receiver of the payment.
4. Participants within a route don't know how many participants were involved in the entire payment route.
5. At best, even the receiver would only know a throw-away wallet address from payment channel state, assuming the user has done best practices outside of the protocol.
6. If a connector is compromised, it won't leak information about messages previously transmitted over the connector.

Notably, we do not require that this protocol is secure against timing analysis, or in situations where onion routers can collude with each other. Tor, for example, is not secure against these two threats. Our model is not either.

5 System Design

We designed our solution as a layer that sits on top of ILP-STREAM. This ensures that connectors can run ILP-CEPA without requiring any modifications to the underlying ILP protocols. The following section outlines the various components of our system:

5.1 Overview

The protocol picks a number of CEPA-compatible connectors, and routes an onion-circuit over them to a destination node. Each link in the onion-circuit is a STREAM connection, which may be routed through ILP connectors as described in the base Interledger Protocol. See Figure 2 for a diagram of the completed route. Messages to the destination are incrementally encrypted with ephemeral symmetric keys established with each of the CEPA-connectors. The details on how this route is established, routed on, and secured is described in detail below.

5.2 Connector Set-up

When a connector boots up, it instantiates a server that listens for incoming ILP-STREAM connectors. This server is defined by an ILP address, and a secret key that is generated as part of the server instantiation process. For any connector to connect to this server, they require knowledge of both the secret key, and the address. For base ILP, this is assumed to be handled out-of-band (For example, the client and destination verbally exchange addresses of accounts they want to transact money between.) This is impractical when defining larger chains with onion-routers, so we require that all connectors publish both their address and their STREAM server secret key to a publicly viewable directory service at set-up time. In addition, each connector generates a public key and private key using 1024-bit RSA, and publishes the public key to the directory service as well. This asymmetric key is used in the key-exchange step to generate ephemeral shared keys. Since base ILP does not define a Public Key Infrastructure, we define our own in this step.

In place of a novel PKI, it would be possible in theory, to just establish Simple Payment Setup Protocol (SPSP) connections between the sender and each of the Onion Routers, and run Diffie-Hellman key exchange over SPSP

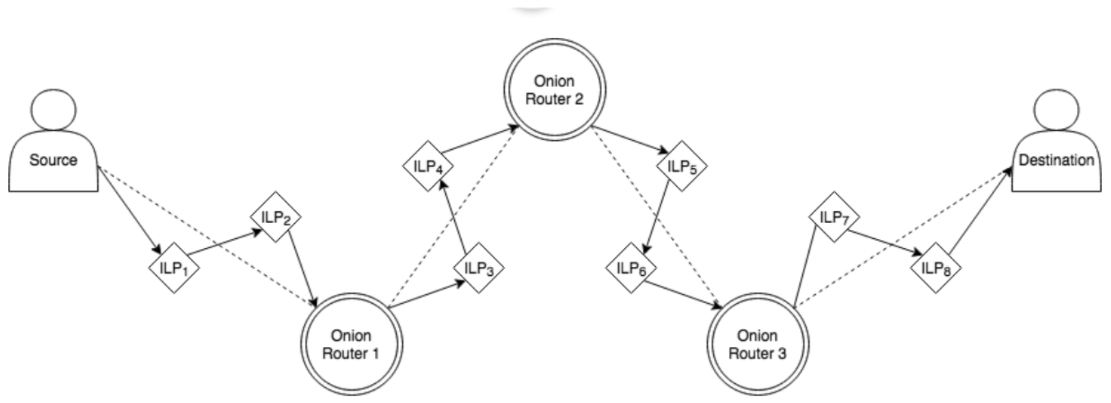

Figure 2: ILP-CEPA circuit.

to establish shared keys. SPSP runs over HTTPS, so it would leverage the PKI of the Internet. However, such a scheme would leak an unordered list of the onion routers on the route, so we decided to implement our own PKI instead. The roadmap for ILP includes a custom-addressed based PKI in the coming year, so we believe that our approach would integrate well into that.

The directory service stores a key-value mapping between ILP addresses of STREAM servers to a tuple containing their shared key and public key.

5.3 Client Setup

When a client wishes to send an onion-routed message to a destination address, the following steps take place:

1. The client queries the directory service for a list of all connectors running the CEPA protocol.

2. The client determines how many hops they wish their onion-routed circuit to contain. This number defaults to 3, but can be increased to any integer up to 10 if the client so desired. The number of hops **cannot** be less than 2.

3. The client chooses 3 (or whatever their desired number of hops) onion routers from the list returned by the directory service. In our base implementation, this is done at random. However, since this is done client-side, clients are free to choose how to determine which onion routers to use. For example, they might know a list of friendly nodes, have certain nodes they dont trust, choose geographically proximate nodes, nodes with best exchange rates, etc.

5.4 Key-Exchange

Before the client can construct the packet payload, they must establish shared keys with each onion router. Establishing pairwise keys with each onion router is not secure, because it would be trivial for an off-path passive observer to correlate network flows to learn the onion routers and sender for a given onion-routed circuit. Therefore, keys are negotiated incrementally through the onion-routed circuit.

5.5 Connector Set-up

We choose Elliptic Curve Diffie-Hellman (ECDH) key exchange as our key-exchange algorithm of choice, because it is fast, requires only two messages over the network, and guarantees forward secrecy.

To create a new circuit, the sender (lets call her Alice) sends a create message to the first node in the onion-routed circuit (call it Bob.) The payload of this create message contains the first half of the Diffie Hellman Key Exchange handshake. That is, g^x, encrypted with Bobs

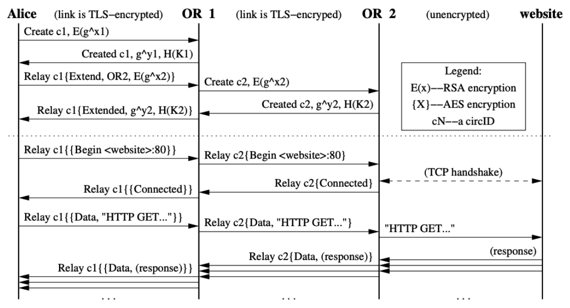

Figure 3: Like Tor, CEPA also negotiates keys incrementally through the onion routed circuit.

public key, which Alice retrieves from the directory service. Bob receives this message, and computes the shared secret $K_{ss1} = g^{xy}$, for some randomly chosen key y. Bob replies to Alice with a payload containing g^y. From this, Alice can compute the same shared secret K_{ss1}. Now, Alice and Bob have established a symmetric key shared secret they can use to encrypt messages between each other. To extend the circuit further, Alice can send an extend message to Bob to tell him to extend the onion circuit to a new connector (lets call her Carol). The payload of this message contains g^x (for a different x than the one she chose to negotiate with Bob) This message is encrypted with Carols public key, such that Bob cannot learn the value of the new x. This payload, along with Carols ILP address, is encrypted with K_{ss1}, Alice and Bobs shared key, and sent to Bob. On reception, Bob unencrypts the message, learns Carols ILP address, and forwards the payload to her. Carol can decrypt this payload using her asymmetric private key, to learn g^x. Carol generates a new y, computes a secret key $K_{ss2} = g^{(xy)}$, which will be the ephemeral key negotiated between Carol and Alice. Carol sends an extend message back to Bob, which contains g^y. Bob forwards this message back to Alice. Just as Bob was used to extend the message to Carol, Carol can be used to extend the circuit forward, up to any number of connectors. The last connector in the route should be the destination connector Alice wishes to ultimately communicate with.

This circuit-level handshake protocol achieves unilateral entity authentication (Alice knows shes handshaking with the OR, but the OR doesnt care who is opening the circuit Alice uses no public key and remains anonymous) and unilateral key authentication (Alice and the OR agree on a key, and Alice knows only the OR learns it). Since it uses Diffie Hellman, It also achieves forward secrecy and key freshness [16].

5.6 Packet Construction

Modeled after Tor's Onion Routing, CEPA packets consist of a payload recursively encrypted with ephemeral keys of the on-path onion connectors, from furtherest away to closest. It also contains an HMAC for integrity, the address of the next hop connector, and is padded to a fixed length. See Figure 3 for illustration.

5.7 Packet Routing

When an connector receives an onion routed packet, the following process take place.

1. The received message is padded to the max length with null bytes to prevent length correlation attacks. Therefore, the connector must strip away all null bytes from the end of the message to get the original payload.

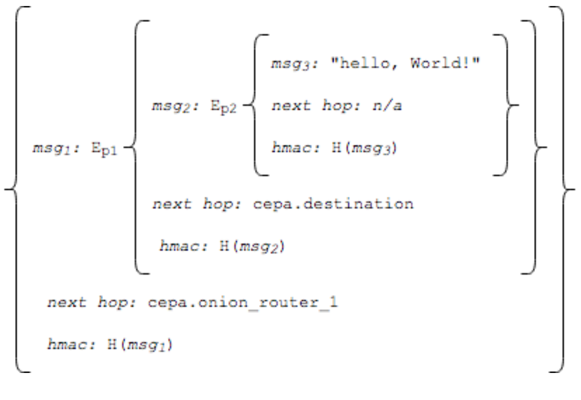

Figure 4: ILP-CEPA simplified packet construction. The base packet is 'wrapped' in the public keys of the routers on-path of the circuit. For visual clarity, this diagram does not show padding.

2. The connector sees from one of the packet header fields that the received packet is an CEPA packet. It uses the ephemeral shared secret K_{ss_i} to decrypt the message.

3. This payload is parsed as a JSON to give a structured collection of key-value pairs that the connector can read. It contains the ILP address of the next hop connector (which may be null), an HMAC checksum, and the payload to be forwarded to the next hop.

4. If the next hop if null, then the payload is the message sent from the sender. The destination can act on this message as desired. This action is outside the CEPA protocol.

5. If the next hop is not null, the connector queries the directory service for the public key and shared secret of the connector corresponding the next hop ILP address. The connector uses the address and shared secret to establish a new ILP-STREAM connection to the next hop.

6. The connector takes the payload, and pads it with null bytes up to the max length. This message is sent along the STREAM connection to the next hop.

6 Implementation

6.1 Client

We built a client that initiates the ILP-CEPA protocol described above in Node.js. We built on top of ILP's base connector repository, which provided a Node.js implementation of a barebones ILP connector. For all the cryptography functions, we used *crypto*, a Node.js package that provides a JavaScript ES6 wrapper to OpenSSL's cryptography functions. For symmetric key encryption, we used AES-256 in CBC mode. For the Diffie-Hellam Key Exchange, we used the fast ECDHE variant, sampling from the secp256k1 elliptic curve as defined in the libsecp256k1 library. Our HMAC used SHA256 as the base hashing function.

6.2 CEPA Connector

We also built an implementation of a CEPA-connector, which is capable of participating in onion-routed circuits. Like the client, this was built in Node.js using Javascript. For all the cryptography functions we used the *crypto* library. This implementation was under 1000 lines of Javascript code.

6.3 Directory Service

The core Interledger team is currently working on designing and developing a Public Key Infrastructure for ILP. Such a design would greatly influence the design of the directory service. Furthermore, we did not believe that the directory service implementation was a core part of our protocol. Tor has an existing, well-studied distributed directory service that our design would look very similar to. The directory service does not introduce major latency or performance implications, so we decided to implement a simple, centralized directory service for the sake of studying our protocol for this paper.

Our directory service was a Flask webserver, written in Python, hosted on an online hosting platform. It exposes a REST API that allowed connectors to: publish their address, public key, and STREAM shared secret. This is used when a CEPA-connector boots up and initializes. retrieve a list of all CEPA-connector addresses, and their related data. This is used for the client to pick an onion route. get the public key and STREAM shared secret for a given address. This is used by intermediate CEPA-connectors to establish connections.

The data was stored in an in-memory Python key-value store. The URL of the webserver was hardcoded into the connector code. The implementation of the directory service was very simple, clocking under 50 lines of Python code.

6.4 Network Simulation

We set up provisioning with Terraform and Python on DigitalOcean, using multiple 1vCPU, 1GB instances stationed across their NYC, SFO, and LON regions. This would allow for us to run several servers, connect a client and send onion messages across to a destination. With more time, we can perform very detailed analysis on ILP in general that would provide more insight into the overhead of ILP-CEPA.

7 Evaluation

Our evaluation strategy was two pronged. First, we designed various attacks, and tested our protocol's defense against them. This was done by spinning up a number of CEPA-connectors on a local testnet, and simulating traffic between nodes. For passive attacks, we studied collected traffic flow logs and looked for vulnerabilities. For active attacks, we assumed that one or many of the nodes were controlled by an adversary (in this case, us), and tried to see what kind of attacks were possible with that amount of access to the network. Our results from these experiments are descibed in the next section. In the following section, we analyze some of the performance characteristics of our protocol. Overall, we are satisfied with the number of threats we can protect against, while acknowledging that there remains certain weaknesses and vulnerabilities.

7.1 Performance Analysis

We studied how the time to complete a payment, measured as the time between the source querying the directory service and the destination receiving the message, and how it varied as the number of hops varied. For each number of hops, we repeated the experiment 100 times and took the average. This was executed on a local testnet, where all the connectors were instantiated on the same computer. Therefore, the time is entirely the result of CPU time, and not The result of this experiment is described in Figure 4.

We see from this graph that, as would be expected, the time taken increases with the number of hops. However, it is not a linear relationship. The delta time increases for every extra hop. This is because AES encryption is slower for larger inputs. AES is the bottleneck in terms of CPU cycles, and since the network latency is zero on the local test network, this encryption/decryption time dominates. This is a problem, performance wise. One would think that clients that only need two hops could only encrypt twice, incurring a much smaller time penalty than more paranoid clients that want to use, for example, 10 hops. However, this opens up the protocol to a length correlation attack. Since each level of encryption increases the message length, messages sent along the first circuit would be significantly smaller in length than those sent along the second. This would allow a passive observer to learn which nodes belong to which circuit (although not the order of the nodes.) Therefore, all messages sent over the CEPA protocol should be padded to the max length of some constant number of hops. Messages that don't use the full length can simply pad with extra null bytes.

A corollary of this design decision is that the protocol must specify some number of hops, and have all messages encrypt/pad to that length. No circuit would be allowed to have more hops than that. Conversely, circuits that need only a small number of hops like 2-3 must incur the time penalty of encrypting a very large payload (since under AES, encrypting null bytes is no faster than encrypting random bytes) This penalty could be several seconds is the maximum number of hops is large (approximately 10 seconds).

8 Attacks and Defenses

In designing ILP-CEPA, we considered a wide number of possible attack vectors. Below we summarize some of them, and describe how well our system can withstand these attacks.

8.1 Passive Attacks

These are attacks in which the attacker reads transit information without altering it, with the intention of analyzing the information for malicious purposes.

Observing Node Traffic Patterns. Traffic analysis on a node will not reveal their destination, or the unencrypted contents of the data they send or receive. However, it will reveal traffic patterns around frequency of data sent and received, as well as the addresses of nodes sending/receiving data to/from it. Correlating these at a connection level would require significant work on the part of the adversary, since multiple connections each with multiple STREAMs could be running simultaneously on a single node.

Observing Content. All content is encrypted, and therefore unobservable to an off-path observer. Due to onion-wrapping, an on-path observer cannot observe the

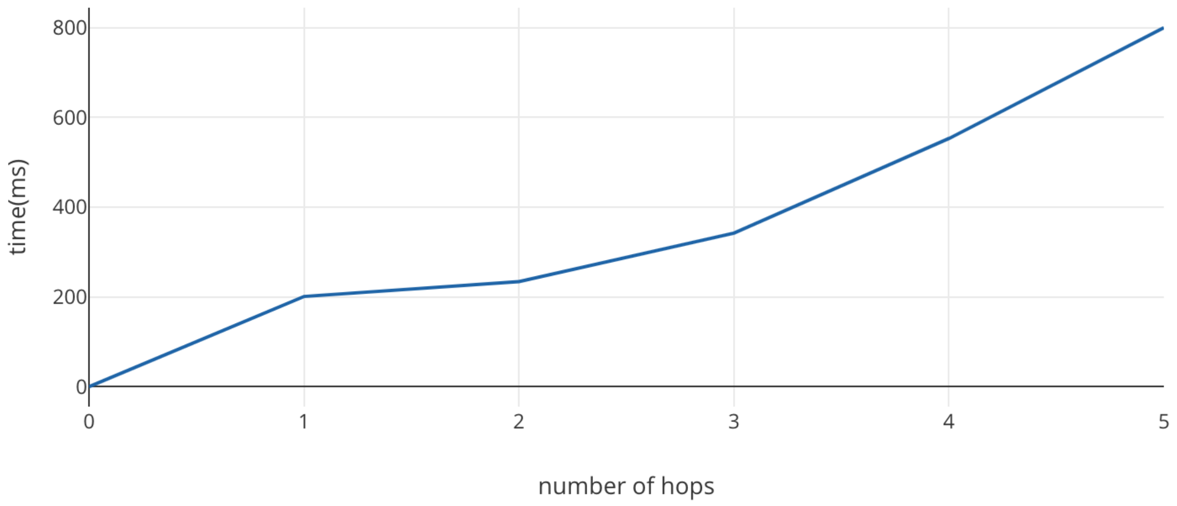

Figure 5: Transaction completion time for varying number of hops.

unencrypted message either, except for the destination node.

Node Failure. STREAM transactions are not atomic, but rather sent as a stream of atomic micropayments. If a node along the STREAM circuit goes down, error messages propagate back to the initiator of the connection. The protocol recovers by automatically constructing a new route (which does not contain the unresponsive node), and proceeds to send the remainder of the transaction amount along that route. In this process, the sender incurs only a very very small amount of monetary loss (enough to be negligible.) If an onion-router dies, no such guarantees are made. The sender will receive a timeout message, and the onus lies with the sender to re-initiate a payment. The only guarantee is that the monetary loss with also be small (enough to be negligible.)

End-to-End Size Correlation. All payloads are padded to equal length. This puts a cap on the maximum number of hops, but prevents guessing onion routers. However, sealing off this attack vector comes with a heavy performance effect. The protocol must hard-code the maximum number of hops, and pad all messages to that length. This means that all transactions need to encrypt this large number of bits, even transactions that elected to have a much smaller number of hops. This trade-off is described in more detail in the next section.

End-to-End Ledger Correlation. ILP plugins allow each connector to have multiple accounts, preventing correlation of ledger transactions. The view an attacker has of the overall transaction space is much and disjointed than on a single ledger, making attacks that attempt to trace a transaction to completion [7] improbable.

Protocol Distinguishability. In the current implementation, it is quite easy for an on or off-path adversary to determine whether a given message is being sent over CEPA or not. All CEPA packets are the same length (due to padding), so that is the easiest way for an adversary to determine how a message is being transmitted. One possible way to mitigate this is to split CEPA packets into arbitrary length, and STREAM those. The CEPA connector would reconstruct the full payload on its end. However, even this could be attacked. CEPA nodes would still take more time to process CEPA messages than not CEPA messages (because of the overhead of AES decryption, padding, etc.), so timing attacks could be used to leak the protocol being used. In the future work section, we describe a major change to the base ILP protocol, involving routing ILP messages themselves over STREAM in a VPN-like configuration, that could be used to fully seal off this attack vector.

Timing Attacks. As mentioned in the Security Guarantees section, this approach does not directly protect against timing attacks. A passive observer could observe data of CEPA-standard length transmitted along an incoming STREAM connection to a node, and then observe CEPA-data along an outbound STREAM connection. With high probability, these two packets would belong to the same onion circuit, and therefore, the passive observer can learn the preceding and following connectors in the circuit. This attack could be made harder by randomizing wait times at each hop, or by sending dummy packets along random STREAM connections, but such tactics only make timing attacks harder, not impossible. Especially with the amount of Big-Data based tools at a large adversary's disposal, timing attacks are very difficult to protect against, even on established networks like Tor. The best defense is to have large networks with high traffic that masks individual circuits in the noise, but this would require widespread adoption of both ILP and our protocol.

8.2 Active Attacks

These are attacks where an attacker compromises a node, or attempts to otherwise modify data for malicious purposes.

Packet Spinning. This attack aims to block other onion routers from being selected in circuits. The attack involves placing a malicious relay node inside an anonymizing system and keeping legitimate nodes busy. We achieve this by creating circular circuits and injecting fraudulent packets, crafted in a way that will make them spin an arbitrary number of times inside our artificial loops. At the same time we inject a small number of malicious nodes that we control into the anonymizing system. By keeping a significant part of the anonymizing system busy spinning useless packets, we increase the probability of having our nodes selected in the creation of legitimate circuits, since we have more free capacity to route requests than the legitimate nodes. This technique may lead to the compromise of the anonymity of people using the system [10].

Having a maximum number of nodes that can be included in a single onion-routed circuit greatly increases

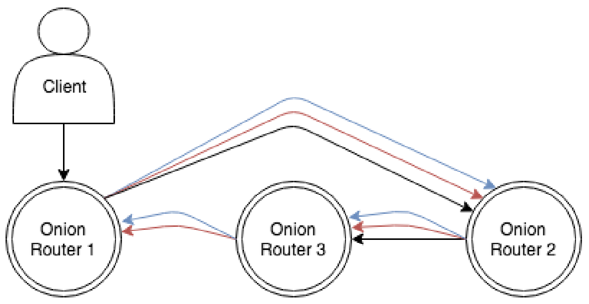

Figure 6: Example of a Packet Spinning Attack. A malicious client constructs a packet wrapped with the same ORs'public keys multiple times.

the amount of resources needed to execute such an attack. However, it is a valid concern and is especially dangerous in the network's early days when the network doesnt have a high number of CEPA connectors running on it. As the network scales, the amount of computational power needed to execute such an attack increases making it prohibitive for an adversary short of nation-state actors.

Denial of Service. Analysis by artificially congesting circuits and evaluating changes in latency is possible. An attacker can obtain the entire path of a user [14, 8]. Tor node traffic loads can be estimated and analyzed against known traffic patterns. This scenario can easily be set up with Packet Spinning-style DoS'ing.

Routing Attacks. Based around limiting a victims routing selection space and then analyzing their circuit construction (again, possible through Packet Spinning). If the attacker is chosen for the circuit, they may be able to analyze route choice and extrapolate based on the default routing algorithm available. CEPA combats this by using a pseudo-random algorithm for the router (but not necessarily ILP) node selection.

Replay Attacks. Replaying one side of the Elliptic Curve Diffie-Hellman key exchange protocol will result in the creation of a different ephemeral shared session key. Therefore, past messages cant be unencrypted even if an adversary had access to a recorded session and compromised a onion router node.

There are also attacks possible on the directory service, but since we only implemented a bare-bones version and expect it to change with the release of ILP's PKI, we did not enumerate vulnerabilities in the directory service.

9 Future Work

One big limitation with the current implementation of ILP-CEPA is that onion routed traffic looks very different from regular, non-onion routed ILP traffic. A consequence of this is that off-path adversaries can, with relatively little effort, determine which nodes are running CEPA onion-routing software, and which are just routing on the base ILP protocol. This doesnt break any of our security guarantees, but it is an unfortunate side-effect of our implementation.

We are aware of a relevant solution to this problem and discuss a brief design of an architecture to mitigate this leakage of information. Taking advantage of ILP's ILDCP parent-child hierarchy, a construction of a virtual private network topology can be constructed. The benefits of this construction allow for the free transport of packets within the VPN after establishing an encrypted point-to-point tunnel with the parent. A scenario where this construction may be useful is in the case where application administrators manage multiple connectors and want a single entry-point for secure payments.

The current status of changing public-key-infrastructure on Interledger is that it's unlikely to do more than what SPSP is doing, which leverages TLS certificates at a specific payment pointer (which can be resolved to an HTTPs URL). Onion routing works by including public keys in the domains which can then be used to establish ephemeral keys, so we could in theory try to create payment pointers via our own version of SPSP that leverages a similar construction.

Although onion routing is tried and tested, there are newer works that work more effectively for a smaller number of nodes, and additional there are claimed solutions to the collusion problem. Ideally we would take these constructions[2, 4, 13] and implement several of them to compare. Alongside improving privacy, there are plently more ways we can make this protocol more efficient. [9] One thing we did was publish 1024bit RSA public keys to the directory service, but we should be using ed25519 keys since they're way more compact and provide similar strength guarantees. We can also make use of CPU cycle analysis and try to make the server function as asynchronously as possible.

Additionally, we would like to formalize the crypto-economic incentives behind running an onion router. There is currently no incentive for someone running an ILP connector to also run an onion router CEPA node, beyond good will and volunteerism. Likewise, Tors relay network consists entirely of volunteers. We believe this is not sufficient. The next step would be to create some of pay structure to encourage participation. This is a non-trivial task since expanding the onion router subset from volunteers to care about the network to the general public opens up the system to attacks from entities trying to make a quick buck at the expense of the network. However, we believe this is possible. The unmodified Interledger protocol incentivizes participation by rewarding all forwarding connectors with a small percentage reward of the transaction they enable. They do this while still guaranteeing transaction completion (although its not fully atomic.) We believe a similar reward scheme could be extended to CEPA connectors, but would require careful consideration of a range of attacks.

10 Acknowledgements

We would like to thank Evan Schwartz from ILP for assisting us with our initial design. We would also like to thank cryptography researchers Pedro Moreno-Sanchez at Purdue, and Dan Boneh at Stanford, for sharing with us their time and resources.

References

[1] Alex Biryukov and Ivan Pustogarov. Bitcoin over tor isn't a good idea. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 122–134. IEEE, 2015.

[2] Philippe Camacho and Fernando Krell. Asynchronous provably-secure hidden services. Cryptology ePrint Archive, Report 2017/888, 2017. `https://eprint.iacr.org/2017/888`.

[3] Mauro Conti, Sandeep Kumar, Chhagan Lal, and Sushmita Ruj. A survey on security and privacy issues of bitcoin. *IEEE Communications Surveys & Tutorials*, 2018.

[4] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. *CoRR*, abs/1503.06115, 2015.

[5] Matthew Green and Ian Miers. Bolt: Anonymous payment channels for decentralized currencies. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 473–489. ACM, 2017.

[6] Lightning-Network. lightningnetwork/lightning-onion, Nov 2018.

[7] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M Voelker, and Stefan Savage. A fistful of bitcoins: characterizing payments among men with no names. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 127–140. ACM, 2013.

[8] Steven J Murdoch and George Danezis. Low-cost traffic analysis of tor. In *Security and Privacy, 2005 IEEE Symposium on*, pages 183–195. IEEE, 2005.

[9] Lasse Øverlier and Paul Syverson. Improving efficiency and simplicity of tor circuit establishment and hidden services. In *International Workshop on Privacy Enhancing Technologies*, pages 134–152. Springer, 2007.

[10] Vasilis Pappas, Elias Athanasopoulos, Sotiris Ioannidis, and Evangelos P Markatos. Compromising anonymity using packet spinning. In *International Conference on Information Security*, pages 161–174. Springer, 2008.

[11] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. `https://lightning.network/lightning-network-paper.pdf`, 2016.

[12] Pavel Prihodko, Slava Zhigulin, Mykola Sahno, Aleksei Ostrovskiy, and Olaoluwa Osuntokun. Flare: An approach to routing in lightning network. `https://bitfury.com/content/5-white-papers-research/whitepaper_flare_an_approach_to_routing_in_lightning_network_7_7_2016.pdf)`, 2016.

[13] Stefanie Roos, Pedro Moreno-Sanchez, Aniket Kate, and Ian Goldberg. Settling payments fast and private: Efficient decentralized routing for path-based transactions. *arXiv preprint arXiv:1709.05748*, 2017.

[14] Juha Salo. Recent attacks on tor. *Aalto University*, 2010.

[15] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy (SP)*, pages 459–474. IEEE, 2014.

[16] Paul Syverson, R Dingledine, and N Mathewson. Tor: The second-generation onion router. In *Usenix Security*, 2004.