P^{4}PCN: Privacy-Preserving Path Probing for Payment Channel Networks

Ruozhou Yu, Yinxin Wan, Vishnu Teja Kilari, Guoliang Xue, Jian Tang, Dejun Yang

***Abstract*—Recent advances in security and cryptography have enabled new paradigms for secure networking in various scenarios. The *payment channel network* (PCN) is a notable example, which has emerged from the combination of the traditional credit network in economics and the latest blockchain technology. PCN provides a secure and efficient way for conducting payments, by addressing both the intrinsic financial risk of the credit network and the scalability issue of the blockchain. A crucial challenge in PCN is routing, *i.e.*, to find a set of paths that fulfill a payment request. Due to the fully distributed and dynamic nature of PCN, existing routing algorithms utilize active probing to improve routing success probability. However, while the payment itself is privacy-preserving through existing protocols, the probing process can leak sensitive information including the location of the sender or the recipient. In this paper, we address the privacy of the users in the path probing process, filling in the last piece of the privacy puzzle in PCN. We propose P^{4}PCN, a cryptographic protocol for anonymous active probing without knowing the identities or public keys of the intermediate nodes, while hiding the locations of sender and recipient as well as any path-related information. Our protocol is lightweight and scales with the number of hops a probe explores. We confirm its performance via real-world implementation and simulation experiments.**

***Keywords*—Blockchain, privacy and anonymity, payment channel network, routing, universal re-encryption**

I. Introduction

Blockchain is a cryptographic mechanism that achieves security via decentralization. As a distributed ledger, blockchain ensures data security through consensus of distributed maintainers, so that no one can manipulate the ledger data without breaking a significant portion of the maintainers. Blockchain security largely depends on the size of the maintainer set. For this reason, cryptocurrency has been introduced both as a killer application and as the incentive mechanism, driving the general crowd to participate in maintaining the blockchain. Since the invention of the blockchain, thousands of cryptocurrencies have been developed, supporting numerous novel applications such as smart contract, supply chain, etc. The total capitalization of the cryptocurrency market tops at over $800B, with even far more implicit economic impact in all business sectors.

As a basic function of cryptocurrency, however, digital payments are encountering a seemingly conflicting situation. On one hand, our conventional centralized financial infrastructure can handle billions of transactions every day, but suffers from intrinsic financial risks and lack of security and transparency. On the other hand, blockchain enhances security and eliminates the financial risk, at the cost of severely degraded efficiency and scalability due to the need for global consensus. This issue has drawn significant attention from academia and industry.

The *payment channel network* (PCN) emerges as a very promising solution to this problem, which combines blockchain with the credit network in economics [6], [8]. Specifically, users establish peer-to-peer channels with deposits, and transfer funds by adjusting deposit allocation on the channels. Honest transactions are stacked in each channel, with only final results added to the blockchain when the channel is closed. For security, each channel is protected by an on-chain smart contract, such that a dishonest off-chain behavior is punished via on-chain arbitration. This way, expensive blockchain operations are limited to the establishment, close-out, and rare dispute arbitration for off-chain channels. A well-connected channel network can enable off-chain transactions for most payments, drastically improving the efficiency and scalability of blockchain. Bitcoin and Ethereum, the two leading cryptocurrencies, are both deploying PCNs to scale their main blockchains [1], [10].

At its core, PCN relies on routing to find payment paths with sufficient fund balances, and employs a multi-hop payment contract to secure indirect payments via the network [10]. The biggest challenge for PCN routing is the distributed and dynamic nature of the PCN, where channel balances are constantly changing with on-going transactions. To improve routing success, many algorithms employ probing, which actively gathers up-to-date network information before making routing decisions [14]–[16]. It has been shown that probing-based solutions significantly improve routing success over solutions based on static or periodically updated information [14], [15].

A major concern of digital payment users is their privacy during transactions. As a blockchain is transparent, providing strong anonymity for on-chain transactions is intrinsically difficult. As a result, existing anonymous blockchains only ensure pseudonymity of the transactions but not unlinkability [13]. PCN has a natural advantage for privacy, as most transactions happen within channels without being published. With the newly developed anonymous payment contracts, information such as user identity or location, and the transaction value, can be hidden from external adversaries and curious intermediate nodes [7], [9], [17]. Yet the problem is not fully solved, as such information may also be leaked in routing. Notably, many probing-based routing algorithms, such as the Spider network [14], Flash [15] and CoinExpress [16], cannot provide anonymity despite being able to greatly improve payment success over static privacy-preserving algorithms [6], [12].

In this paper, we aim to resolve this one last piece of

Yu (ryu5@ncsu.edu) is with North Carolina State University, Raleigh, NC 27606. Wan, Kilari and Xue ({ywan28, vkilari, xue}@asu.edu) are with Arizona State University, Tempe, AZ 85287. Tang (jtang02@syr.edu) is with Syracuse University, Syracuse, NY 13244. Yang (djyang@mines.edu) is with Colorado School of Mines, Golden, CO 80401. This research was supported in part by NSF grants 1704092, 1717197, 1717315, and 1525920. The information reported here does not reflect the position or the policy of the funding agency.

978-1-7281-0962-6/19/$31.00 ©2019 IEEE

puzzle for privacy-preserving PCN. We propose P^{4}PCN, an anonymous path probing protocol for probing-based routing algorithms, which can be combined with privacy-preserving payment contracts to construct a full protocol stack for privacy-preserving payments in PCN. Compared to existing anonymous communication (AC) protocols such as onion routing [4], the biggest challenge for anonymous path probing is that *the sender may not know the path(s) that the probe will traverse in advance*. This breaks the classic assumption of knowing all public keys of nodes on the communication path, rendering all existing AC protocols inapplicable, including but not limited to [2]–[4]. We address this by designing a novel cryptographic protocol. The key idea is to allow each intermediate node to both derive a symmetric key with the sender, and encrypt queried data as well as necessary information for later decryption at the recipient, at the same time. At its core, the Universal Re-encryption protocol is used to re-encrypt the probe at each hop [5], while the symmetric key derivation is inspired by Sphinx [3] and HORNET [2]. Our protocol is both lightweight and scalable. We validated its efficiency and scalability via implementations and contrast experiments against a naive construction based on the *hybrid universal mixing* (HUM) protocol in [5].

Our main contributions are summarized as follows:

- To our best knowledge, no existing work has studied or addressed the anonymous probing problem in networking. We are the first to study and address this problem.
- We design a cryptographic protocol for anonymous probing that preserves sender and recipient anonymity, and ensures the integrity and confidentiality of queried data.
- We thoroughly analyze the security of our protocol, and validate its efficiency and scalability through implementation and contrast experiments.

The rest of this paper is organized as follows. Sec. II presents our system model and security goals. Sec. III presents the detailed design of our protocol. Sec. IV presents the security analysis of our protocol. Sec. V presents our performance evaluation results. Sec. VI discusses how this protocol affects the routing algorithm design, as well as other potential applications of this protocol. Sec. VII concludes this paper.

II. System Model and Security Goals

A. System Model

We consider a fully distributed PCN denoted by $G = (V, E)$, where V is the set of user accounts that constitute the network, and E denotes the set of directional payment channels between nodes. Each channel $e \in E$ is associated with a set of channel status attributes, $\mathsf{attr}_e = \{\mathsf{balance}_e, \mathsf{delay}_e, \mathsf{expiration}_e, \mathsf{fee}_e, \dots\}$. Due to the dynamic nature of the network, some of the attributes, notably the balance of each channel, are constantly changing with on-going payment requests and transactions. As a result, each node only has up-to-date information regarding all the channels adjacent to it, while it has no knowledge of the instantaneous channel status of any remote channel.

A payment request is comprised as $(\mathsf{src}, \mathsf{dst}, \mathsf{val})$, where src and dst are the sender and recipient respectively, and val is the amount to be transferred. Some requests may have additional constraints, for example, a deadline dl, a fee budget cost, etc. Due to the primary constraint on val as well as these secondary constraints, the sender commonly needs to gather instantaneous information from the network to decide on its actual payment paths. Both information gathering and path selection are part of the *payment routing* process. If a guaranteed payment success is preferred, the sender can also reserve the balances on the path(s) until the payment is done, to avoid concurrency issues [16].

In this paper, we primarily focus on the information gathering process, which we call *path probing*. In path probing, the sender sends out probing messages to gather information from network nodes. Each node attaches the queried information onto the probe, and then forwards the probe to one or multiple next hops, until each probe reaches the intended recipient. Note that since the sender has no knowledge of the remote channels, we assume that the actual choice of next hops is at the discretion of each forwarding node. For example, a node may either choose a simple broadcast-based method [16], or guide the selection of forwarding nodes with its local information, such as in imbalance-aware routing [14] or coordinate-based routing [12]. For generality, we do not rely on a specific probing algorithm, and assume that each node v independently decides the set of neighbors $\mathcal{N}_v$ to forward a received probe. We use data_v to denote the data that node v attaches to an on-going probe. Based on the request, data_v may contain balance, congestion, delay, fee, etc. Each node reports data of the same length l_{data}.

B. Threat Model

Existing probing-based routing algorithms do not consider the privacy of the sender and/or the recipient. For example, CoinExpress [16] explicitly involves the sender and recipient nodes in its probes. In this paper, we focus on an adversary who tries to infer the payment patterns of senders and/or recipients in the network. For example, observing a sender sending out a probe that passes through several (corrupted) nodes, an adversary can link this action with a future anonymous payment transaction that goes through the same set of nodes. The popularity of a recipient may also be inferred by observing how many probes are targeting a recipient during a period.

We consider a local adversary that controls a subset of nodes by either inserting malicious nodes or compromising existing nodes. We assume the non-existence of a global adversary that can observe all network traffic, as all communications between peers are conducted via secure and anonymous channels. The adversary can access all the stored secrets and past communications on the compromised nodes, but cannot access such information on non-compromised nodes. For any privacy-concerning user, we assume that the adversary cannot compromise (or does not know if it has compromised) all the adjacent nodes of the user; otherwise, the sender/recipient's privacy can be trivially broken since the adversary can access all the user's in-coming/out-going channels. Note that this assumption realistically holds in PCNs that support private channels [11]. The goal of the adversary is to undermine user privacy instead of launching denial-of-service attacks. Defense against denial-of-service attacks is mainly through detection and prevention, which is out of the scope of this paper.

C. Security Goals

We expect our protocol (or any other anonymous probing protocol) to fulfill the following security goals.

- **Correctness:** Correctness means that a cryptographic protocol correctly implements all the functions of a normal non-cryptographic protocol. In our probing problem, this means that 1) each node (both intermediate node and recipient) can identify its role regarding a received probe, 2) each intermediate node is able to attach queried information onto the probe, and 3) the recipient can obtain all the attached information from the probe.
- **Data Integrity:** The adversary cannot break the integrity of the queried data attached by a non-compromised node without being detected by the sender/recipient.
- **Data Confidentiality:** The adversary cannot access the information attached by a non-compromised node, except for the knowledge that it already has access to, *e.g.*, statuses of channels adjacent to a compromised node.
- **Sender Privacy:** For any payment request between non-compromised users, the adversary cannot infer the identity or location of the sender. It also cannot decide if a non-compromised node is the sender of any request.
- **Recipient Privacy:** For any payment request between non-compromised users, the adversary cannot infer the identity or location of the recipient. It also cannot decide if a non-compromised node is the recipient of any request.
- **Sender-Recipient Privacy:** The adversary cannot decide whether there is any on-going request between any users.

III. Protocol Design

A. Preliminaries

1) Common Cryptographic Primitives: Let $\mathcal{G}$ be a cyclic group of prime order q (with length l_{key}), satisfying the Decisional Diffie-Hellman (DDH) assumption. Let g be a published generator of $\mathcal{G}$. We omit writing the modulus operation for brevity. We use the symbol $\perp$ to denote an empty string, group element or identifier of an arbitrary size. The following cryptographic primitives are used in our protocol:

- $\mathsf{E}(pk, m)$: encryption of message m with public key pk.
- $\mathsf{D}(sk, \xi)$: decryption of ciphertext ξ with private key sk.
- $\mathsf{PRG}(s)$: a secure pseudorandom generator with key s.
- $\mathsf{MAC}(k, m)$: the Message Authentication Code (MAC) of message m under key k, with length l_{MAC}.
- $\mathsf{H_M}(\cdot)$, $\mathsf{H_E}(\cdot)$: two cryptographic hash functions.

2) Universal Re-encryption (URE): URE is a cryptographic protocol for mixnets proposed by Golle *et al.* [5]. In plain words, URE enables mix nodes to re-encrypt an encrypted message *without* knowing the public key used for encryption. The original construction of URE was built upon the ElGamal cryptosystem, utilizing the homomorphic property of ElGamal. Let $(x, y = g^x)$ be a private-public key pair for ElGamal encryption where x is the private key. The ciphertext of message $m \in [1 \dots q-1]$ is $\xi = \mathsf{E}(y, m) = (m \cdot y^k, g^k)$ where $k \in [1 \dots q-1]$ is a secret random number, and the decryption algorithm runs as $m = \mathsf{D}(x, \xi) = \xi[0] \cdot (\xi[1]^x)^{-1}$. Given two ciphertexts $\mathsf{E}(pk, a)$ and $\mathsf{E}(pk, b)$ encrypted under the same key pk, the ElGamal cryptosystem satisfies that $\mathsf{E}(pk, a) \times \mathsf{E}(pk, b) = \mathsf{E}(pk, a \times b)$ for a group operator $\times$. Based on this, a ciphertext in the URE protocol has two components, $\mathsf{E}(pk, m)$ and $\mathsf{E}(pk, 1)$, where the latter can be used to re-encrypt the former without knowing the public key initially used to encrypt it.

Note that the task of the original URE protocol is the opposite of ours. The original URE aims to anonymously transmit a message from sender to receiver through a sequence of known mix nodes, such that no mix node has the knowledge of the sender, the receiver, or any node other than itself and its neighbors on the route. The message itself stays fixed and encrypted through the entire transmission. In our task, not only the path that a probe will traverse is undetermined before probing, but each intermediate node also needs to attach data (local channel information) queried by the sender. Each node's data must be kept secret so that other nodes cannot infer whether this node is on the probed path or not. On the other hand, each node needs to provide enough information for the sender/recipient to decrypt the attached data. To address these, we must modify the original URE protocol to fit our needs.

B. Anonymous Probing with Unknown Paths

Our insight is that, besides using URE for re-anonymization as in mixnets, we can also use part of the URE protocol to carry out a Diffie-Hellman Key Exchange (DHKE) with each node that the probe traverses. The derived key can then be used to encrypt the attached data of the intermediate node, while the node can provide necessary DH-value to help the recipient decrypt the encrypted data. To ensure anonymity, decryption-related information and the data are encrypted within a *reversed onion*: each node wraps a layer of encryption over the received payload plus the newly attached DH-value and data. The recipient gradually derives all the symmetric keys, and uses the corresponding key to "peel off" each encryption layer, revealing all the attached data layer-by-layer.

1) Probing with a Single Path: For simplicity, let us first consider the case where probing is done through just one (unknown) path. Here, each node knows which exact neighbor it will forward a given probe to as next hop. But the sender does not know the public key of any node on this path. Assume this path is represented by $(\mathsf{src} = v_0, v_1, v_2, ..., v_{n-1}, v_n = \mathsf{dst})$. We also assume that the sender and the recipient securely share any information that either of them uses in the protocol. We thus do not distinguish between the cryptographic operations by the sender or the recipient. The probing protocol works as follows:

Algorithm 1: Create Probe (Sender)

Input: Probe ID I.
Output: Initial probe message ρ_0.
Generate random $x, \kappa, k_\beta, k_\gamma \in_U \mathcal{Z}_q$; let $y \leftarrow g^x$;
$s_0 \leftarrow g^{\kappa k_\beta}$;
$m \leftarrow (pk_0, \perp, \perp)$;
$\mathsf{pl}_0 \leftarrow (\xi_0, \mathsf{MAC}(\mathsf{H_M}(s_0), \xi_0), \zeta_0)$, where
  $\xi_0 = m \oplus \mathsf{PRG}(\mathsf{H_E}(s_0)), \zeta_0 = \perp \oplus \mathsf{PRG}(\mathsf{H_E}(s_0))$;
Generate probe $\rho_0 = (a_0, b_0, c_0, d_0, \mathsf{pl}_0)$, where
  $a_0 = y^{k_\beta k_\gamma}$, $b_0 = g^{k_\beta k_\gamma}$, $c_0 = g^{k_\gamma}$, $d_0 = g^{\kappa}$;
return ρ_0.

Create Probe: The sender generates secrets $(x, \kappa, k_\beta, k_\gamma)$ that are shared with the recipient via a secure channel. The probe contains the ElGamal ciphertext $(a_0, b_0) = \mathsf{E}(y, 1)$, an element c_0 containing part of b_0, a random element d_0, and the payload pl_0. Besides being part of the ciphertext, b_0 acts as the DH-value of the sender, used to establish a DH symmetric key with each hop. c_0 and d_0 are both used for per-hop DHKE, so that the recipient can derive the symmetric keys. Specifically, c_0 is

used to pass the common part of the per-hop DH-value to the next hop for constructing the next hop's own DH-value, while d_0 is used to pass the current node's DH-value to the next hop for encryption into the payload. The DHKE with v_i is based on secret keys k_β and x_i (and common randomizing terms), and for sender, though purely for formality, $x_0 = \kappa$. Both k_γ and κ are to make the initial probe indistinguishable from the ones that have already passed some hop(s). Note that both the symmetric key encryption using PRG and the encrypted empty string ζ_0 are to facilitate padding, discussed later on.

Algorithm 2: Process Probe (Intermediate Node)

Input: Node v_i, probe $(a, b, c, d, \mathsf{pl})$, data.
Output: Next probe message ρ_i.
Decrypt (a, b) using its own key pair to see if it is the recipient, and jump to Algorithm 3 if so;
Generate random $k_i, x_i \in_U \mathcal{Z}_q$;
$s_i \leftarrow b^{x_i}$;
$m \leftarrow (pk_i, d, \mathsf{data})$;
$\mathsf{pl}_i \leftarrow (\xi_i, \mathsf{MAC}(\mathsf{H_M}(s_i), \xi_i), \zeta_i)$, where $\xi_i = m \oplus \mathsf{PRG}(\mathsf{H_E}(s_i)), \zeta_i = \mathsf{pl} \oplus \mathsf{PRG}(\mathsf{H_E}(s_i))$;
Construct probe $\rho_i = (a_i, b_i, c_i, d_i, \mathsf{pl}_i)$, where $a_i = a^{k_i}$, $b_i = b^{k_i}$, $c_i = c^{k_i}$, $d_i = c^{x_i}$;
return ρ_i.

Process Probe: Each intermediate node's procedure starts from checking if it is the recipient. This is done by trying to decrypt the ciphertext $(a, b) = \mathsf{E}(y, 1)$ using its own set of private keys whose corresponding requests are expecting in-coming probes. If the node is not the recipient, it proceeds to generate the next probe to be forwarded. The first step is to derive its shared DH key s_i using b as the DH value of the sender. s_i is then used to encrypt a new payload, which contains the previous payload, the old element d as the DH-value of the previous hop (which is essential for the recipient to decrypt the previous payload), and the queried data to be attached. Next is to update all the group elements. The ciphertext (a, b) is re-encrypted using new key k_i. The common part of the DH-value, c, is also updated by key k_i to reflect the update on b. d_i stores the DH-value of the current node, which will be either wrapped within the payload at the next hop if the next hop is still intermediate, or used for decryption if the next hop is the recipient.

Algorithm 3: Decrypt Probe (Recipient)

Input: Probe $(a, b, c, d, \mathsf{pl})$, shared keys x, k_β.
Output: Path p, per-hop data $\mathbf{data} = \{\mathsf{data}\}$.
while $\mathsf{sizeof}(\mathsf{pl}) \geq$ *valid payload segment length* **do**
  $s \leftarrow d^{k_\beta}$;
  $(\xi, \nu, \zeta) \leftarrow \mathsf{pl}$;
  Check if ν is a valid MAC of ξ under s; abort if not;
  $(pk, d', \mathsf{data}, \mathsf{pl}') \leftarrow (\xi, \zeta) \oplus \mathsf{PRG}(\mathsf{H_E}(s))$;
  Add pk to p, and add data to $\mathbf{data}$;
  $d \leftarrow d'$, $\mathsf{pl} \leftarrow \mathsf{pl}'$;
end
return $(p, \mathbf{data})$.

Decrypt Probe: Once a recipient receives a probe targeting itself, it starts decrypting the onion-encrypted payload to obtain all the data. For each layer, the secret key is derived by combining the corresponding node's DH-value, either directly transmitted from the last hop or obtained from the last layer (for previous hops), with the secret k_β of the sender/recipient. The secret key is then used to decrypt the payload, revealing the next layer of payload and DH-value, and the attached data of this layer. Eventually, the path is reconstructed from the attached public keys, and all data are decrypted layer-by-layer.

2) Probing with Multiple Next Hops: Now, let us consider the case where each node can forward a probe to multiple neighbors to increase the probing success probability and path diversity. Forwarding the same probe message to multiple neighbors would enable linkable attacks, if two colluding nodes both receive the same message. For this reason, a new pair of (k_i, x_i) should be generated for every neighbor that the probe is sent to. Due to the re-encryption and the onion encryption at every hop, even if colluding nodes receive the probe sent by the same sender, they cannot relate them or distinguish them from any other probe in the network.

3) Length-based Inference Attacks and Padding: One issue with the original onion routing protocol is that an intermediate node can infer its relative location along the path by observing the length of the encrypted message, if it knows or can estimate the length of the original message and/or the size of each layer. To prevent such an attack, padding is commonly used to keep all encrypted messages of constant length. In our scenario, padding can serve an additional purpose. In payment routing, commonly the sender/recipient has a limit on the length of an acceptable payment path. If a path is too long, both the risk of a failed payment is high, and it may incur a high transaction fee at the sender. For this reason, the sender can pad the onion-encrypted payload up to a pre-defined limit. Each node attaches information by dropping the last π bits of the previous onion, where π is the size of a layer of the payload and is known based on the message format. The recipient can then detect paths exceeding the pre-defined length, if the last layer of encryption does not contain the initial payload sent by the sender, and discard such paths accordingly.

IV. Security Analysis

A. Correctness

First, each node v_i can identify whether it is the recipient of a given probe by trying to decrypt the URE ciphertext (a, b) against its own set of key pairs that are awaiting probes. If the decrypted value for any of the key pairs is 1, the probe belongs to the corresponding probing request. During probing, each node attaches its local data onto the probe by encrypting it with the secret key s_i derived using DHKE with the sender's provided DH-value (element b in the URE ciphertext). Finally, the recipient opens each layer of the onion by deriving s_i. Note that for the i-th hop ($i > 0$), it satisfies that $b_i = c_i^{k_\beta}$ as well as $d_i = c_{i-1}^{x_i}$. Therefore, we have $s_i = b_{i-1}^{x_i} = c_{i-1}^{k_\beta x_i} = d_i^{k_\beta}$. Since d_i is always encrypted in the onion of the next hop (or directly transmitted to recipient at last hop), the recipient (knowing the shared secret k_β) can gradually derive the secret keys to decrypt all layers and obtain all the data. Note that the recipient does not need to decrypt the innermost layer, as any information there can be directly shared between the sender and the recipient, and hence s_0 is purely used to achieve indistinguishability.

B. Data Integrity and Confidentiality

Since each piece of data attached to the probe is MACed and onion-encrypted, it suffices to show that the secret key s_i of

a non-compromised node cannot be obtained by an adversary if both the sender and the recipient are non-compromised. Note that the secret key s_i contains two secret components, the x_i held by the non-compromised node v_i, and the $k_\beta k_\gamma$ initially embedded by the sender. Without either of these two pieces or the key itself (which is discarded after use), no adversary can derive s_i based on the DDH assumption.

C. Sender, Recipient, and Sender-Recipient Privacy

Regarding sender and recipient privacy, we consider several cases. First, a compromised node not adjacent to either the sender or the recipient can only see some probe passing through, but cannot link it to either the sender or the recipient due to the encryption. A node adjacent to the sender may try to infer if the previous hop is the sender, but cannot tell since some other node that connects to the last hop directly or via a path may have originated the probe. Similarly, a node adjacent to the recipient cannot tell whether the probe is targeted to the next hop or some other node afterwards. Furthermore, a node cannot know its own location with regard to either the sender or the recipient in the network, since all probes are of the same length and are re-randomized at every hop. Finally, since the adversary cannot tell whether any probe can be linked to any sender or recipient, he also cannot tell whether a pair of sender and recipient is communicating or not.

V. Performance Evaluation

In this section, we evaluate the performance of our protocol. Since we are the first to propose a protocol for the anonymous probing problem in networks, our only reference solution is a modified version of the *hybrid universal mixing* (HUM) protocol described in the original URE paper [5], which uses a different way for establishing symmetric keys used for encryption. Below, we first describe the modified HUM protocol, and then analyze the performance of these two protocols.

A. Modified Hybrid Universal Mixing [5]

In the HUM protocol, the authors proposed to use symmetric key encryption to encrypt and re-encrypt the initial message, while using the URE protocol to deliver the symmetric keys used by each hop for final decryption. This protocol can be modified to additionally use the symmetric keys to encrypt and re-encrypt the data each node attaches to the probe. Assume that the maximum path length is L. The sender sends out the following message as the initial probe:

$$\rho_0 = (\mathsf{pl}, \mathsf{E}(pk, 1), \mathsf{vec}),$$

where $\mathsf{pl} = (\xi, \mathsf{MAC}(\mathsf{H_M}(s_0), \xi), m_{\mathsf{rand}})$, $\xi = (pk_0, \perp) \oplus \mathsf{PRG}(\mathsf{H_E}(s_0))$, m_{rand} is a random filler string of length $(L-1) \cdot l_{\mathsf{seg}}$ with $l_{\mathsf{seg}} = l_{\mathsf{data}} + l_{\mathsf{key}} + l_{\mathsf{MAC}}$, and

$$\mathsf{vec} = (\mathsf{E}(pk, 1), \ldots, \mathsf{E}(pk, 1), \mathsf{E}(pk, s_0))$$

contains $(L-1)$ copies of the encrypted value 1 plus an encrypted copy of the initial secret key s_0.

When a node v_i receives a probe, it attaches its pk_i and data to the head of the payload while dropping the last l_{seg} bits, picks a new key s_i, and then encrypts the new payload with s_i; the MAC is also computed and attached. It then encrypts s_i with the help of the value $\mathsf{E}(pk, 1)$ at the head of vec (if the path does not exceed length L), and then left-rotates all the components in vec. This can be done by multiplying the first element of $\mathsf{E}(pk, 1)$ with s_i in ElGamal encryption. Finally, it re-encrypts the middle component $\mathsf{E}(pk, 1)$ in the probe, as well as all the other components in vec, using a new random key k_i. If there are more than one next hop, then as in our protocol, new keys need to be generated, and the entire message re-randomized, for each of the next hops of the current node.

Identification of the recipient is also done by decrypting the middle component $\mathsf{E}(pk, 1)$. After that, the recipient decrypts all the secret keys stored in vec from the back, and uses all the keys to gradually decrypt the payload and obtain all the data.

B. Analytical Comparison

Here we analytically compare the overheads of our protocol and the modified HUM protocol. We assume that the network has a path length limit of L intermediate nodes, and each probe has traversed a path of length $P \leq L$ on average. Size of a group element including public/private keys is $l_{\mathsf{key}} = K$ bytes. We focus on the overhead for processing 1 probe at each node, as the number of probes a node receives/forwards is determined by the actual probing algorithm employed, which is out of the scope of this paper. We omit the generation of all public/private key pairs. Table I shows an analytical comparison between the two protocols.

Scheme		Our Protocol	Modified HUM
Comp.	Sender	$(5, 0)$	$(2L+2, 0)$
	Interm.	$(6, 1)$	$(2L+3, 1)$
	Recipient	$(P+1, 1)$	$(P+1, P)$
Communications		$(2L+4)K$	$(3L+2)K$

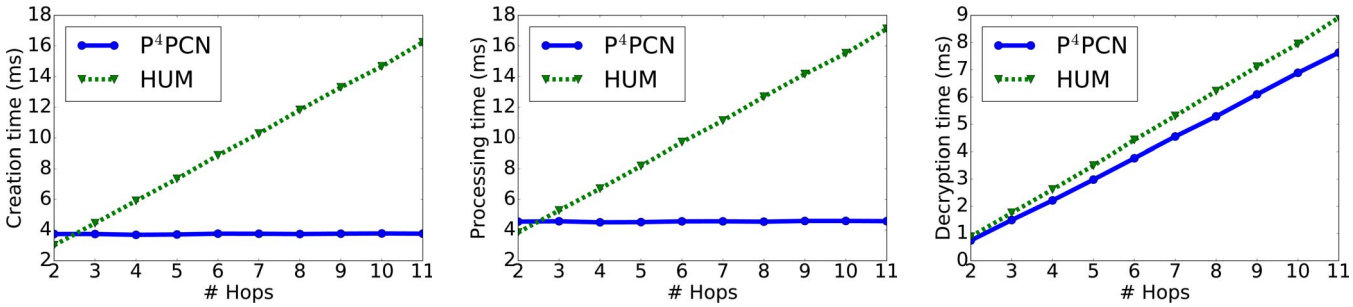
TABLE I: Overhead comparison for one probe. Computation is displayed as $(\mathsf{mod\_exp}, \mathsf{mod\_inv})$, denoting the numbers of modular exponentiation and modular inverse operations respectively; communication overhead is measured by the length of the probe header excluding data payloads and MACs but including the path information (public keys of each hop).

C. Simulation Experiments

We implemented both protocols in Java. The ElGamal key size was 1024 bits (128 bytes). A data segment contained 4 bytes of data and a 128-byte public key identifying this hop. The MAC of a data query was 20 bytes using the HMAC-SHA-1 implementation of Java. The SHA1PRNG implementation of Java is used for the keyed PRG in order for reproducibility of the experiments. Each probe traversed the maximum allowed hops before reaching the recipient. We ran our experiments on a Linux PC with Quad-Core 3.4GHz CPU and 16GB of memory, and repeated each experiment for $10,000$ times.

Fig. 1 shows the average execution times for the creation, processing and decryption of a probe, respectively. Clearly, the creation and processing times of our protocol remain constant with increasing number of hops the probe can explore, while those of HUM increase linearly. For decryption, both protocols have an increasing execution time, but ours is faster than HUM due to less number of modular inverse operations.

Fig. 2 shows the probe size versus the number of hops. HUM has a larger probe size over ours due to the more number of ElGamal ciphertexts used to store all the encryption keys, resulting in higher communication overhead in practice.



(a) Probe creation time vs. # hops. (b) Probe processing time vs. # hops. (c) Probe decryption time vs. # hops.

Fig. 1: Probe creation (sender), processing (intermediate node) and decryption (recipient) execution times per probe.

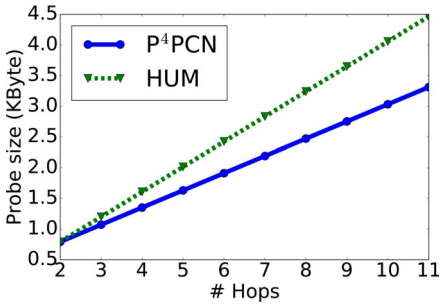


Fig. 2: Probe size vs. # hops.

VI. Discussions

Probing with anonymity: With our protocol, now the intermediate nodes will have no way of knowing the sender and/or the recipient. Unfortunately, this complicates the probing algorithm design, because each node now has limited information in deciding where to forward a probe. For instance, simple broadcasting can lead to the case where every copy of a probe will wander in the network until reaching the recipient, and all the probes generated by a request will most likely overwhelm the entire network. Two promising solutions are probabilistic forwarding and coordinate-based routing. The former naturally preserves privacy, while the latter can be implemented using a privacy-preserving coordinate system to avoid breaking user anonymity [12]. We plan to investigate these in our future work.

Other applications: Beyond PCN, our protocol may also find applications in a wide range of other scenarios. For example, this protocol can be used to anonymously construct a communication path towards a remote location through a dynamic sensor network or a vehicular network, or to find a trust path in a trust-based social network.

VII. Conclusions

In this paper, we studied the problem of anonymous probing in PCN routing, the last piece in building a fully privacy-preserving PCN payment protocol stack. We proposed a cryptographic protocol to ensure both data security and user privacy during probing-based network information gathering. Combined with existing probing-based dynamic routing algorithms and privacy-preserving payment protocols, this can achieve efficient payments with high success probability and full privacy preservation for PCN. Our protocol is both lightweight and scalable. Comparing our protocol with another possible protocol derived from hybrid universal mixing, our protocol has constant probe creation and processing overheads, lower (although linear) probe decryption overhead, and smaller communication overhead. Though we specifically targeted the PCN, our protocol can be used in a broad range of other domains, such as for anonymous data querying in vehicular networks, sensor networks, social networks, and beyond.

References

[1] "Raiden Network." URL: https://raiden.network/

[2] C. Chen, D. E. Asoni, D. Barrera, G. Danezis, and A. Perrig, "HORNET: High-speed Onion Routing at the Network Layer," in *Proc. ACM CCS*, 2015, pp. 1441–1454.

[3] G. Danezis and I. Goldberg, "Sphinx: A Compact and Provably Secure Mix Format," in *Proc. IEEE S&P*, 2009, pp. 269–282.

[4] D. Goldschlag, M. Reed, and P. Syverson, "Onion Routing for Anonymous and Private Internet Connections," *Commun. ACM*, vol. 42, no. 2, pp. 39–41, 1999.

[5] P. Golle, M. Jakobsson, A. Juels, and P. Syverson, "Universal Reencryption for Mixnets," in *Proc. CT-RSA*, 2004.

[6] G. Malavolta, P. Moreno-Sanchez, A. Kate, and M. Maffei, "SilentWhispers: Enforcing Security and Privacy in Decentralized Credit Networks," in *Proc. ISOC NDSS*, 2017.

[7] G. Malavolta, P. Moreno-sanchez, C. Schneidewind, A. Kate, and M. Maffei, "Anonymous Multi-Hop Locks for Blockchain Scalability and Interoperability," in *Proc. ISOC NDSS*, 2019.

[8] P. Moreno-Sanchez, A. Kate, M. Maffei, and K. Pecina, "Privacy Preserving Payments in Credit Networks: Enabling Trust with Privacy in Online Marketplaces," in *Proc. ISOC NDSS*, 2015, pp. 8–11.

[9] O. Osuntokun, "AMP: Atomic Multi-Path Payments over Lightning," 2018. URL: https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-February/000993.html

[10] J. Poon and T. Dryja, "The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments," Tech. Rep., 2016.

[11] P. Rochard, "Lightning Routing Node Starter Pack," 2019. URL: https://medium.com/lightning-power-users/lightning-routing-node-starter-pack-704c0e7d79cb

[12] S. Roos, P. Moreno-Sanchez, A. Kate, and I. Goldberg, "Settling Payments Fast and Private: Efficient Decentralized Routing for Path-Based Transactions," in *Proc. ISOC NDSS*, 2018.

[13] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized Anonymous Payments From Bitcoin," in *Proc. IEEE S&P*, 2014, pp. 459–474.

[14] V. Sivaraman, S. B. Venkatakrishnan, M. Alizadeh, G. Fanti, and P. Viswanath, "Routing Cryptocurrency with the Spider Network," *arXiv:1809.05088*, 2018. URL: http://arxiv.org/abs/1809.05088

[15] P. Wang, H. Xu, X. Jin, and T. Wang, "Flash: Efficient Dynamic Routing for Offchain Networks," *arXiv:1902.05260v1*, 2019.

[16] R. Yu, G. Xue, V. T. Kilari, D. Yang, and J. Tang, "CoinExpress: A Fast Payment Routing Mechanism in Blockchain-based Payment Channel Networks," in *Proc. IEEE ICCCN*, 2018.

[17] Y. Zhang, Y. Long, Z. Liu, Z. Liu, and D. Gu, "Z-Channel: Scalable and Efficient Scheme in Zerocash," in *Proc. ACISP*, 2018.