Walking Onions: Scaling Anonymity Networks while Protecting Users

Chelsea H. Komlo, *University of Waterloo;* Nick Mathewson, *The Tor Project;* Ian Goldberg, *University of Waterloo*

https://www.usenix.org/conference/usenixsecurity20/presentation/komlo

This paper is included in the Proceedings of the 29th USENIX Security Symposium.

August 12–14, 2020

978-1-939133-17-5

Walking Onions: Scaling Anonymity Networks while Protecting Users

ARTIFACT EVALUATED usenix ASSOCIATION PASSED

Chelsea H. Komlo
University of Waterloo

Nick Mathewson
The Tor Project

Ian Goldberg
University of Waterloo

Abstract

Scaling anonymity networks offers unique security challenges, as attackers can exploit differing views of the network's topology to perform epistemic and route capture attacks. Anonymity networks in practice, such as Tor, have opted for security over scalability by requiring participants to share a globally consistent view of all relays to prevent these kinds of attacks. Such an approach requires each user to maintain up-to-date information about every relay, causing the total amount of data each user must download every epoch to scale linearly with the number of relays. As the number of clients increases, more relays must be added to provide bandwidth, further exacerbating the total load on the network.

In this work, we present *Walking Onions*, a set of protocols improving scalability for anonymity networks. Walking Onions enables constant-size scaling of the information each user must download in every epoch, even as the number of relays in the network grows. Furthermore, we show how relaxing the clients' bandwidth growth from constant to logarithmic can enable an outsized improvement to relays' bandwidth costs. Notably, Walking Onions offers the same security properties as current designs that require a globally consistent network view. We present two protocol variants. The first requires minimal changes from current onion-routing systems. The second presents a more significant design change, thereby reducing the latency required to establish a path through the network while providing better forward secrecy than previous such constructions. We implement and evaluate Walking Onions in a simulated onion-routing anonymity network modelled after Tor, and validate that Walking Onions indeed offers significant scalability improvements for networks at or above the size of the current Tor network.

1 Introduction

When participants in an anonymity network hold different views of the network's membership and topology, an adversary can exploit these differences in knowledge to distinguish clients' behaviours [12] or intercept users' traffic [46]. Anonymity networks in practice [13] have prevented these attacks by requiring all participants to share a *globally consistent view* of the entire state of the network, and giving clients complete control over selecting relays for their paths. While this approach prevents the described attacks, requiring a globally consistent view results in quadratic bandwidth growth as the number of clients increases [26], because the number of relays must also increase to provide more capacity, and all parties must download information about all relays. While today's Tor network requires only approximately half a percent of its total bandwidth to serve network state [39, 41], increasing the number of clients and relays by one order of magnitude would result in the consumption of roughly five percent of the network's (ten times larger) total bandwidth simply to distribute network state; an increase by two orders of magnitudes results in the consumption of *half* of the network's (hundred times larger) total bandwidth. Clearly, requiring a globally consistent network view for all participants is an obstacle to anonymity networks reaching the scale of modern-day browsers [37].

Moreover, some use cases benefit immediately from improved scalability. For anonymity networks requiring a globally consistent view, a client joining the network on initial startup must download the complete network state, which can be prohibitive for mobile users or those in areas with poor connectivity. Further, "on-demand" clients—applications that are used only occasionally—still incur bandwidth overhead, as the client must either continue to fetch network state when idle, or bootstrap from scratch after becoming active.

To safely address these scalability issues, we present *Walking Onions*,[1] a set of novel protocols and algorithms to reduce the amount of data clients must maintain in onion-routing anonymity networks from linear to constant relative to the number of relays. Thus, as the number of clients increases, the load to the network to distribute relay information to clients increases linearly with the number of new relays, as opposed

[1]The Walking Onion, or *Allium* × *proliferum*, is a charming edible plant that spreads by growing a cluster of new bulbs on a stalk, until the onion becomes so top-heavy that the bulbs flop over and take root somewhere new.

to quadratically. Notably, unlike prior designs with similar scalability goals [26, 29], our protocols maintain the same security properties as designs that require clients to maintain up-to-date information about every relay in the network.

Our design includes a novel path-selection and circuit extension protocol, wherein clients obliviously choose random paths through the network without prior knowledge about the network's membership or topology, and yet can verify the correctness of their paths after they are constructed. We present two variants of our protocol. One variant does not change the existing security model for Tor, and makes minimal changes to a generic onion-routed design. The second protocol variant improves upon state-of-the-art single-pass onion-routing protocols by relaxing the forward secrecy for path selection from immediate to windowed, but preserves forward secrecy for content. In turn, this design reduces client latency by building upon a technique from Sphinx [11] to create circuits with only a single round trip between the client and all relays on the path. This improvement reduces the latency of circuit creation from quadratic to linear with respect to path length.

We focus on the applicability of Walking Onions to Tor [14], but note that Walking Onions can be used by other anonymity networks with similar threat models and scalability concerns, such as HORNET [10].

Contributions. We present a novel set of efficient path-selection and circuit-extension protocols for anonymity networks using onion routing. Our work builds upon a prior Tor proposal [24] written by a co-author of this paper. Our contributions include:

- Two novel protocols that require clients of an anonymity network to maintain only a constant-sized amount of network information, while remaining secure against route capture and epistemic attacks (described in Section 2).
- Primitives to efficiently and verifiably transmit relay information, and a comparison of authentication mechanisms.
- Techniques and protocols to enable clients to constrain relay selection to a subset of relays fulfilling some attribute, while not performing this filtering locally.
- Implementation and evaluation of these protocols' bandwidth and CPU consumption in a simulated anonymity network modeled after Tor.

Organization. We present background material in Section 2. We give an overview of Walking Onions in Section 3, and describe how it distributes network information in Section 4. We present novel path selection and circuit extension protocols in Section 5, and techniques to enforce path requirements in Section 6. We evaluate the performance of these protocols in Section 7, and conclude in Section 8.

2 Background

Throughout this work, we use Tor as a case study work to demonstrate the applicability of our protocols; however, we present an alternative application of Walking Onions to HORNET [10] in Appendix A.

Existing Tor Protocol. Tor is a low-latency anonymity network with 2.5 to 11 million unique users [22, 42] and roughly 6,500 volunteer-run relays [40]. Tor's protocol requires clients to keep up-to-date information about every relay in the network. This information is provided by *directory authorities*, a trusted set of servers administered by core members of the Tor community. Every epoch (in Tor, one hour), relays upload information about themselves to the authorities, who then vote on the relays' statuses. From these votes, the authorities compute a multisigned consensus directory document representing their conclusions. By checking the signatures and timestamp on the consensus document, clients and relays ensure the validity and timeliness of the latest consensus.

Once a client has obtained the latest consensus, the client selects a list of relays (typically three) to use for a multi-layered encrypted communication tunnel, called a *circuit*, to route traffic through the network. Tor clients do not select relays uniformly at random; instead, for load balancing, they choose each relay with probability related its measured bandwidth and intended position on the path.

Tor uses a *telescoping* technique [14] in which circuits are built one hop at a time. The client uses each partially completed circuit to perform a handshake with the next relay in the circuit, until the circuit is complete. Building circuits via telescoping allows forward secrecy against key compromise, as the client negotiates a shared session key with each hop, but also increases latency, as constructing an n-hop circuit needs n round trips.

Path-based Attacks Against Anonymity Networks. Tor's consensus mechanism defends against certain well-known attacks based on how anonymity network information is distributed and used for path selection. Such attacks include *epistemic attacks*, in which an adversary deduces information about a client from information leaked by the client's choice of relays [12], and *route capture attacks*, in which a client's chosen path is replaced or influenced by a malicious intermediary [46]. These attacks can pass undetected if clients' views of the network are not consistent and authenticated.

Designs for Managing Network Information. In a survey by Shirazi et al. [36], anonymity networks fall into two categories based on how paths are built. Paths can be either *source-routed*, in which the client holds complete control over path selection, or *hop-by-hop*, in which intermediate nodes are allowed to influence the next relay selected. However, the latter approach can allow hostile intermediate nodes to influence the client's path to their advantage.

Crowds [33] relies on a peer-to-peer protocol. Paths through the network are determined using a "coin-flipping" random-walk technique, in which each node forwards requests either to another intermediate node, or directly to the intended recipient, depending on a weighted coin flip. Crowds is trivially vulnerable to route capture attacks because of its hop-by-hop routing: a single hostile node can choose a hostile node (or no node at all!) as its successor, thereby ensuring that the rest of the path will be hostile.

ShadowWalker [29] uses a peer-to-peer random-walk protocol and a distributed hash table (DHT) to distribute relay information. To extend a circuit, the client sends a randomly selected index to an intermediate node, which the node uses to perform a lookup in its finger table of known neighbours. ShadowWalker protects against routing attacks by requiring relays to commit to routes by distributing finger tables to deterministically chosen "shadow nodes", which the client then uses to verify the response for their selected index. In spite of these techniques, ShadowWalker is still vulnerable to route capture attacks [35], and the probability of epistemic attacks grows relative to the length of the path.

The Invisible Internet Project (I2P) is a peer-to-peer, fully decentralized anonymous overlay network [47]. Like Tor, I2P implements source-based routing. However, I2P does not rely on central authorities to distribute network directory information. Untrusted "netfill" routers maintain a DHT representing all network information, and "gossip" updates to peers. However, partitioning attacks are possible, as a participant cannot verify the information served by any netfill router [19].

PIR-Tor [30] allows clients to download information about a small subset of relays from a set of authorities using private information retrieval (PIR), thereby preventing observation of *which* relays a client requests from the authorities (or any other intermediary). While PIR-Tor maintains security within Tor's existing threat model, the designs either have undesirable performance or complexity tradeoffs when considering their use in anonymity networks at scale. Specifically, Computational PIR is not scalable due to the high computational cost to the servers that must perform these operations for every client multiple times per epoch. Information-Theoretic PIR requires multiple non-colluding parties and thus changes the structure of existing protocols, adding additional complexity. ConsenSGX [34] improves upon the resource usage of PIR-based relay selection by using trusted execution environments like Intel's SGX. However, adoption of ConsenSGX (or any design requiring trusted execution environments) undesirably changes Tor's threat model to include trust in the hardware, and so risks compromise due to vulnerabilities in trusted environments, as seen in the past [6].

Designs For Single-Pass Circuit Creation. Reducing the number of round trips needed to create circuits in anonymity networks has been addressed using a variety of approaches. Below we highlight a few that have been proposed for Tor; however, these designs make undesirable tradeoffs in security and efficiency, which we also discuss.

Øverlier and Syverson [31] propose a scheme in which the user has access to an authentic copy of every relay's public Diffie-Hellman key from the consensus. Each circuit's session keys are derived from a relay's static key and an ephemeral key provided by the client. However, this approach compromises the forward secrecy of data exchanged over the circuit, due to the use of the relays' static keys.

Kate et al. [21] and Catalano et al. [8, 9] present variants of single-pass schemes using identity-based encryption. The drawbacks of these schemes include the requirement of a central authority for key distribution and the lack of forward secrecy for client communication.

The above designs also do not defend against linking public key material exposed in transit. Sphinx [11] presents a provably secure packet format for anonymity networks, providing cryptographic unlinkability between incoming and outgoing packets. The session key for each node on the circuit is computed from the server's private key and a blinded element initally provided by the client and re-blinded at each hop, thus ensuring unlinkability of public key material between hops. Kate and Goldberg [20] assess the security and efficiency of Tor-preDH, pairing-based onion routing, and Certificateless Onion Routing using Sphinx as the underlying packet format. Unfortunately, undesired tradeoffs remain with each construction, such as the loss of forward secrecy for session keys or the requirement of a complex central PKI.

Cryptographic Sortition. Cryptographic sortition—verifiably randomly selecting participants from a global set—will be used in our work to determine the relays selected for a path through an anonymity network. Sortition typically uses Verifiable Random Functions (VRFs) [28]. VRFs accept an input string and a private key, and produce both a deterministic but unpredictable (to those without the private key) output along with a proof of the correctness of the output. This proof can be verified using the corresponding public key. Sortition has a number of applications in distributed networks; for example, the Algorand Byzantine agreement protocol uses sortition to verifiably select nodes at random to participate in a consensus protocol [16].

3 Walking Onions Overview

To address the scalability of Tor and similar anonymity networks, we present Walking Onions, a design that allows clients to obliviously select relays for paths through the network and establish a secure circuit with these relays. As such, the design of Walking Onions ensures that as new clients and relays join the network, the resulting total load over all network relays scales roughly linearly,[2] as opposed to Tor's quadratically. Further, the design of Walking Onions ensures that clients can begin building circuits after downloading only a constant amount of network data, as opposed to the entire network directory document. Our design protects against route-capture and epistemic attacks, and does not require clients to maintain complete network state information.

[2]There is a tiny quadratic term, because all *relays* (not clients) must still learn about all relays in Walking Onions. The coefficient of the quadratic term in our experiments is 2200× smaller than in today's Tor, however: see 7.1.

3.1 Threat Model

We assume the threat model of the Tor network [14]; see that work for more details. As an overview, Tor's threat model assumes independently operated relays of which a subset can be malicious but the majority are not. A malicious relay is not bound to operate under any particular protocol, and can behave arbitrarily. Furthermore, the threat model includes adversaries that can observe only a *subset* of network traffic. Tor's threat model does not include end-to-end attacks where the adversary can observe information at both edges of the network where specific messages enter and leave, such as timing or volume of packets.

This threat model assumes the anonymity network has a root of trust to produce authenticated network directory documents. We refer to this trust anchor as the *authority*. The instantiation of this authority can vary, so long as its output is verifiable and trusted by all participants. For example, the authority may be distributed among several *voters* who participate jointly. In Tor, the authority is a set of *directory authorities*, of which a threshold number are assumed to be honest [38]. Walking Onions can also support alternative consensus mechanisms for anonymity networks other than Tor, such as verifiably selecting at random a subset of nodes to generate the network directory document. If the anonymity network is completely decentralized (i.e., no entity holds complete information about the network), another possible authentication mechanism is to require each relay to deterministically select t other relays to validate its information for the current epoch.

3.2 Goals

Scalability Goals. As additional clients join the network, a proportional number of relays is typically needed to provide sufficient bandwidth. For scalability, we require that even as the number of relays grows, the cost to clients in bandwidth and memory remains constant. We later relax this requirement to allow for logarithmic growth for clients, in order to improve bandwidth for relays (see Section 7). Further, we require that the latency (in terms of round trips) experienced by a client to establish a new circuit is no worse—and ideally better—than current onion-routing protocols.

Security Goals. We require our designs to fulfill the following security properties:

Correctness. The client must be able to establish a valid circuit through the network; a valid circuit entails that each relay is selected corresponding to the client's path requirements, and corresponds to a valid entry in the current network directory document produced by the authority for the anonymity network. Furthermore, clients must be able to establish a secure shared session key with each relay on the circuit.

Security. The client must be able to verify that the selection of relays on its path has not been influenced by an intermediary in such a way as to result in epistemic or route capture attacks. Specifically, the client must be able to ensure that the relays selected for the client's path were selected at random from a given distribution, and that the distribution itself has not been maliciously modified or influenced. Finally, we require that a malicious relay acting in isolation cannot compromise a client's security or ability to access the network.

Privacy. A user's participation in an anonymity network remains private so long as a network adversary with a limited view of the network [14] cannot gain useful information about the user from observing traffic. Furthermore, an adversary monitoring incoming and outgoing traffic for a specific relay should not be able to perform a linking attack by comparing exposed packet contents.

3.3 Key Design Insights

Walking Onions uses the following key design insights to scale anonymity networks:

Oblivious path selection. In Walking Onions, clients do not maintain a list of relays. Instead, to build a path through the network, a uniform random integer i from a fixed range is selected either by the client directly, or at least in a manner provably uninfluenceable by any intermediary. This i serves as an *index* into a probability distribution generated by the authority over the relays with properties required for that path. After later learning the relay corresponding to i, the client will verify that it was in fact chosen correctly. So long as the client can reliably validate that the resulting relay corresponds to i, the client does not need any relay information beforehand.

Post-hoc identity verification. As a second insight, we note that to extend a circuit to a given relay, some cryptographic handshakes—such as one-way authenticated handshakes based on Diffie-Hellman [17]—can be easily modified to not require knowledge of the other party's public key up front. Notably, a client can initialize a cryptographic handshake with a relay by sending only their own ephemeral public key, and verifying the relay's public key material *after* the relay has responded. Consequently, a client can extend a circuit to a specific relay without any identity or cryptographic information for that relay beforehand.

We next develop these insights into a set of protocols.

4 Network Information in Walking Onions

We begin by outlining notation and terminology, and then we discuss how network directory documents are encoded and distributed in Walking Onions.

4.1 Notation and Terminology

Let α be an integer that determines the precision with which we can represent node selection probabilities. Relays will be selected via random integers i with $0 \leq i < \alpha$. We recommend $\alpha = 2^{32}$.

A *network directory document* is an authenticated document made up of information representing all relays, such as their cryptographic identity keys and IP addresses. These directories are regenerated once every *epoch*.

A *network parameters document* is a constant-sized authenticated document that includes information such as supported protocol versions and network parameters. Such a document is used in practice, for example, to coordinate all clients to switch to a new protocol at the same time, to avoid fragmenting the clients' anonymity set.

A *path* through the network is an ordered list of relays. A *circuit* represents the cryptographic instantiation of the path, in which the client shares a different set of negotiated session keys with each relay in the path.

4.2 Encoding Network Directory Documents

In the Walking Onions design, once every epoch, the authority generates an authenticated network directory document reflecting the current state of all relays. We call the authenticated network directory document an *Efficient Network Directory with Individually Verifiable Entries*, or ENDIVE. We call each entry in the ENDIVE a *Separable Network Index Proof*, or SNIP. Each SNIP corresponds to a single relay; consequently, the ENDIVE comprises the complete set of all SNIPs.

ENDIVEs. Importantly, in Walking Onions, *only relays* need to download ENDIVEs. Relays are required to fetch the entire ENDIVE at bootstrap; afterwards, relays fetch only the changes to the ENDIVE once per epoch. Clients download a constant-size network parameters document once per epoch (if the anonymity network requires this), but do *not* require a complete list of relays to build circuits.

Each ENDIVE contains the set of all SNIPs for the epoch and an authentication tag over this set. We describe options to generate this authentication tag in Section 4.3. We present a walked-through example of an ENDIVE in Appendix B.

SNIPs. In anonymity networks, a *relay entry* is the information about a single relay distributed in a network directory document. A relay entry includes routing information about the relay such as its public keys, IP address(es), and supported features or versions. As described in Section 2, Tor distributes this information in a network directory document, which includes the set of relay entries that are valid for the current epoch, and a signature over the entire document.

SNIPs differ from relay entries by including three *additional* fields, which we now describe.

First, each SNIP includes an *index range*: a range of integer values whose size is proportional to the desired probability of selecting this relay. In Tor, for example, the size of each relay's range would be proportional to its bandwidth. When generating the ENDIVE for each epoch, the authority computes and assigns index ranges for each SNIP in the ENDIVE. The index ranges must be chosen so that every possible index (between 0 and $\alpha - 1$ inclusive) corresponds to exactly one relay. As we describe further in Section 5, these index ranges enable clients to indicate which relay to extend their circuit to without maintaining the ENDIVE locally.

Second, each SNIP includes *its own authentication tag* generated by the authority over *only* the content in the SNIP (we describe several options to perform this authentication in Section 4.3).

Third, each SNIP includes two timestamps indicating when the SNIP was created and when the SNIP expires.

Because SNIPs are individually authenticated, clients can validate them without downloading the entire ENDIVE. We describe how clients build upon this capability to securely perform path selection and circuit extension in Section 5.

Weighted Relay Selection. The index selection mechanism described above, where each relay is assigned to a portion of the index proportional to its selection probability, allows the authority to specify any desired probability distribution over relays. By giving some relays a larger range of index values than others, the authority can cause those relays to be selected more often than others, as is typically desired. Later in Section 6 we will show how to extend this mechanism to encode *multiple different* probability distributions, so that, for example, the last relay in a path can be chosen from a different distribution than a middle relay.

Alternative topologies. For simplicity, in this work we assume a full clique network topology, in which every relay can connect to any other relay. However, Walking Onions is applicable to alternative topologies by issuing multiple ENDIVEs, in which a pre-determined partitioning scheme could assign relays to a specific ENDIVE. Such an approach could be used to integrate Walking Onions into mix networks like Katzenpost [1], which relies upon a stratified topology, in which relays are partitioned into distinct layers and client paths contain one relay per layer.

4.3 Authenticating ENDIVEs and SNIPs

In order for clients to verify SNIPs without downloading the full ENDIVE, each SNIP includes an authentication tag produced by the authority over the information *only* within the SNIP. We now survey several authentication mechanisms, and evaluate the performance of the more promising approaches in Section 7.

One signature per voter. When the authority for an anonymity network comprises multiple voters, one simple solution is to include one signature from each voter in each SNIP. In this case, the number of signatures in the ENDIVE is equal to $N_V N_R$, where N_V represents the number of signing voters and N_R represents the number of relays.

Aggregate/Threshold Signatures. Joint signatures, in which a single signature represents n signers, offer an attractive option for authenticating ENDIVEs and SNIPs in Walking Onions, as multiple signing voters can coordinate to issue a single authentication tag. Aggregate signatures [2,3,25] provide an n-out-of-n scheme in which all voters must participate to produce a joint signature, while threshold signatures [4] provide a t-out-of-n trust model, requiring only a threshold number of voters to produce the signature.

In comparison to other signing mechanisms, threshold signatures offer a compelling alternative when the authority for an anonymity network comprises multiple voters, yet a subset of these voters may be offline at any time.

Merkle Trees. Another authentication approach is to use Merkle tree proofs [27] to ensure a specific SNIP is within the ENDIVE signed by the authority for the network. A client can download the Merkle root for the most-recent ENDIVE in the (constant-sized) network parameters document, which itself is authenticated by the authority. During circuit construction, the client receives a Merkle tree proof along with the SNIP to prove inclusion of the SNIP in the ENDIVE, demonstrating a path from the SNIP to the Merkle root. Note that these proofs can be constructed by relays locally and consequently do not require distribution in the ENDIVE itself.

Merkle trees are particularly attractive for bandwidth savings because of the small amount of new information required for relays to maintain an up-to-date ENDIVE. With the other mechanisms, a relay needs to download an new signature for each SNIP every epoch, whether the SNIP's information has changed or not: the timestamp will have changed and the signatures thus cannot be reused. But with a Merkle tree, the relay can download only the bodies of SNIPs that have changed, plus one unpredictable signature for the tree's root. (The non-leaf, non-root nodes of the Merkle tree can be recomputed, and do not need to be downloaded.)

The savings in relay downloads with Merkle trees, however, is offset by the increased size in SNIPs: they now must contain $\lceil \lg N_R \rceil$ digests, where N_R is the total number of relays in the ENDIVE. We examine this tradeoff more in Section 7.1.

Authenticating ENDIVEs. Because ENDIVEs are simply the set of SNIPs valid for the current epoch, a relay can validate an ENDIVE by verifying the authentication tag for each SNIP in the ENDIVE, and checking no SNIPs are missing by looking for gaps across the SNIPs' index ranges. However, a more efficient validation mechanism is to include an additional authentication tag over the ENDIVE as a whole, for relays to check after downloading the most-recent ENDIVE. Conventional mechanisms can be used for this purpose, such as the one-per-voter signing strategy, as the overhead for signatures is negligible in comparison to the document size.

5 Walking Onions Path Selection and Circuit Extension

We now present two separate protocols allowing clients to obliviously yet verifiably select relays and extend circuits through anonymity networks. We call the first *Telescoping Walking Onions*, and the second *Single-Pass Walking Onions*. We introduce both protocols and discuss their tradeoffs, after a few preliminaries.

5.1 Preliminaries

Let g be a generator of a group of prime order in which the Decisional Diffie-Hellman problem is hard.

Circuit bootstrap. Walking Onions presents efficient path selection and circuit extension protocols, but assumes the client holds trustworthy information about the first hop. We discuss several mechanisms to establish the first hop in a circuit in Section 5.6.

Authenticated key exchange. We assume the existence of a one-way-authenticated two-party key exchange with a post-specified peer, in which the initiator authenticates the responder after both supply ephemeral keys. We follow a similar approach to Canetti and Krawczyk [7] by assuming the idealized functionality of such a protocol with similar assumptions. As part of this idealized authenticated key exchange, we assume the following functions:

$KeyGen_{Auth}(1^\lambda) \rightarrow (x, g^x)$: Generates an ephemeral private/public keypair with security parameter λ.

$ComputeSecretAndAuth(g^x, y, b) \rightarrow (S, A)$: Computes the shared secret S and a value A used to authenticate the relay using the relay's long-lived private key b and ephemeral private key y, along with the client's ephemeral public value g^x.

$ComputeSecretAndValidate(x, g^y, g^b, A) \rightarrow (S, \{0,1\})$: Computes the shared secret value, and authenticates the resulting value using the peer's long-lived public key. Outputs the shared secret S and a Boolean value indicating if the handshake is valid.

Verifiable Random Functions. Single-Pass Walking Onions uses an idealized version of a Verifiable Random Function similar to the VRF standard submitted to the IETF for review [18]. We require the following VRF operations:

$KeyGen_{VRF}(1^\lambda) \rightarrow (x, g^x)$: Generates a private/public keypair with security parameter λ.

$Prove(c, \tau) \rightarrow (\beta, \pi)$: Computes a deterministic output β and a proof π, given the VRF private key c and an input τ.

$Verify(g^c, \beta, \tau, \pi) \rightarrow \{0,1\}$: Verifies the correctness of the VRF output using the VRF public key g^c. Outputs a Boolean value indicating if the proof is valid.

Vanilla Onion Routing. We describe our protocols with reference to a generic Tor-like onion-routing protocol. We call it *Vanilla Onion Routing*, and give a definition in Appendix C.

5.2 Telescoping Walking Onions

We now present Telescoping Walking Onions, a protocol to *extend* an existing circuit by a single hop, and describe the protocol using a step-by-step approach in Definition 1.

Description of protocol. Let R_n represent the last relay in the client's current circuit, and R_{n+1} represent the relay the client will extend the circuit to.

To extend a circuit in Telescoping Walking Onions, instead of selecting a next hop R_{n+1} directly, the client selects a random index i such that $0 \leq i < \alpha$. This index will fall within an index range in the most recent ENDIVE, as described in Section 4. The client sends i to the last relay R_n in their circuit, along with the client's half of the circuit extension handshake. To find the next relay to extend the circuit to, R_n looks up the client's chosen i in the ENDIVE for the current epoch, obtaining the unique SNIP whose index range contains i; this SNIP Σ_{n+1} corresponds to the relay that will become R_{n+1}. The relay R_n starts by relaying the client's handshake in a circuit extension request to R_{n+1}. Upon receiving the response handshake from R_{n+1}, R_n relays that response to the client, along with Σ_{n+1}. The client verifies Σ_{n+1} is authentic and valid (see Section 4.3). Further, the client verifies that R_{n+1} was selected honestly, by checking that i falls within the index range for the SNIP. Finally, the client uses the public keys for R_{n+1} in the SNIP to authenticate the handshake response from R_{n+1}.

We present Telescoping Walking Onions in Definition 1, building upon a generalized circuit extension protocol.

Definition 1. (Telescoping Walking Onions) We label each step with P to denote a path selection operation, and K to denote a key exchange operation for circuit extension. Steps that differ from Vanilla Onion Routing (see Appendix C) are underlined for emphasis.

Let (b_n, g^{b_n}) denote the long-term key for relay R_n.
When the client extends an existing circuit:

1. [P] Select $0 \leq i < \alpha$ uniformly at random.
2. [K] Generate an ephemeral keypair $(x, g^x) \leftarrow KeyGen_{Auth}(1^\lambda)$.
3. [P,K] Send (i, g^x) to R_n over the existing circuit.

When R_n receives a circuit extension request (i, g^x):

4. [P] Obtain the SNIP Σ_{n+1} whose index range contains i in the most recent ENDIVE. This determines the relay that will serve as R_{n+1}.
5. [K] Send the client's g^x to R_{n+1}.
6. [P, K] Wait for a response from R_{n+1}, and send it to the client, along with Σ_{n+1}.

When R_{n+1} receives the circuit extension request g^x:

7. [K] Generate an ephemeral keypair $(y, g^y) \leftarrow KeyGen_{Auth}()$; compute the shared secret and authentication value $(S, A) \leftarrow ComputeSecretAndAuth(g^x, y, b_{n+1})$.
8. [K] Reply with (g^y, A); derive circuit keys from S.

When the client receives a reply indicating the circuit was extended:

9. [P] Verify the received SNIP Σ_{n+1} is timely and corresponds to the chosen i. Verify the authentication tag included in Σ_{n+1}. If the SNIP is not valid, abort. Otherwise, extract $g^{b_{n+1}}$ from Σ_{n+1}.
10. [K] Complete the handshake: Compute $(S, V) \leftarrow ComputeSecretAndValidate(x, g^y, g^{b_{n+1}}, A)$. If $V \neq 1$, abort; otherwise, derive circuit keys from S.

Scalability Goals. Telescoping Walking Onions fulfills the scalability goals described in Section 3.2, as clients do not need to maintain the complete network directory document. It also maintains the same latency overhead for circuit construction as Vanilla Onion Routing, since the network traffic pattern remains the same, and uses the same number of round trips.

Next, we assess the extent to which Telescoping Walking Onions achieves its security goals (described in Section 3.2).

5.2.1 Analysis of Security Goals

Correctness: To maintain correctness, Telescoping Walking Onions must ensure that each relay corresponds to the value i provided by the client. Because each SNIP contains an index range, along with an authentication tag, the client can validate that their choice of i falls within the relay's index range and that the SNIP is generated by the authority for the anonymity network. Furthermore, the client can check the timeliness of the SNIP to ensure the SNIP is valid for the current epoch.

Security: To prevent the attacks described in Section 2, Telescoping Walking Onions must ensure a client can validate that their path has not been influenced by an intermediary. As previously established, a client can validate that their choice of i corresponds to the SNIP of the relay selected for the path. Furthermore, as i can be selected from the full distribution range up to α, the client can select any SNIP in the ENDIVE. Consequently, a malicious on-path relay or intermediary cannot constrain the client to select R_{n+1} from only a subset of all available relays. Finally, while a malicious on-path relay can arbitrarily drop client connections to perform a denial-of-service attack that can influence the final path [5], this behaviour is the same as for existing onion-routing networks.

Privacy: To prevent information leakage to an observer, messages sent in the clear must be unlinkable. (We consider only bitwise unlinkability in our analysis, as protecting against timing-based correlation is outside the scope of our threat model.) As the client selects a fresh randomly generated (i, g^x) for each extension of the circuit, an intermediate node will not be able to derive any further information about other relays in the path (beyond the nodes immediately preceding and following). Furthermore, as the client's messages containing

(i, g^x) are encrypted within the circuit connection between the client and R_n, an adversary observing the network will not be able to link the client and the circuit extension request sent from R_n to R_{n+1}.

5.3 Single-Pass Walking Onions

While Telescoping Walking Onions presents minimal protocol changes to an existing onion-routing network, it also requires the same number of messages to iteratively create a new circuit as Vanilla Onion Routing. We now present *Single-Pass Walking Onions*, a path-selection and circuit establishment protocol with the same scalability benefits as Telescoping Walking Onions, but using only a linear number of total messages relative to the path length. Further, the client now only sends one and receives one message when building a circuit.

The key insight to Single-Pass Walking Onions is this: if the client can be assured that i was selected at random without interference by an intermediary, then the client does not need to select i directly; this responsibility can be shared with intermediate relays in the circuit so long as the client can verify the choice of i was not influenced by any intermediary.

Building upon this insight, we next describe how this random index i is generated in Single-Pass Walking Onions in such a way that the client can verify no intermediary has influenced it. To start, the client generates an ephemeral path-selection keypair (d, g^d), and sends its public value $D = g^d$ to the first hop in the circuit (as mentioned above, we assume the first hop in the circuit is already bootstrapped). Each relay R_j holds a semi-ephemeral path-selection keypair (c_j, g^{c_j}). Recall from before that each relay maintains a long-lived key (b_j, g^{b_j}). The index i is derived using contributions from both the client and the relay, such that relay's contribution remains fixed within a single epoch to prevent the relay from manipulating its input after observing the client's input. The first hop R_j computes $(i, \pi) = Prove(c_j, D^{b_j})$—relay R_j's VRF output (using its semi-ephemeral key c_j) corresponding to the input D^{b_j}, which itself is the Diffie-Hellman shared secret between the client's (d, D) and the relay's long-term key. Relay R_j then blinds D using the Sphinx [11] technique: it computes a blinding value $v_j = H(D^{b_j})$ and changes D to D^{v_j} before passing it along to the relay selected by i.

We will further assess security properties of Single-Pass Walking Onions in Section 5.3.1, but note here that the relay's path selection key is semi-ephemeral to prevent relays from brute-forcing a favourable i by continuously re-generating path-selection keypairs. Further, we bind knowledge of the path-selection key to the relay's long-lived key using the VRF.

This technique of reblinding the client's public key at each hop using a shared secret key as the blinding factor was first introduced by Sphinx [11], and results in the client (and only the client) having the capability to derive the corresponding private key. In this way, Single-Pass Walking Onions departs from Vanilla Onion Routing and Telescoping Walking Onions by not requiring an iterative circuit establishment approach.

Description of protocol. We first describe some additional key points required to understand Single-Pass Walking Onions, and then present the protocol in more detail in Definition 2, building upon a generalized circuit extension protocol.

To prevent a relay from biasing path-selection towards chosen (for example, colluding) relays, we require each relay to publish its path-selection public key in its SNIP, consequently binding the relay to its key for as long as the SNIP is valid. If a relay is compromised and an adversary learns its private path-selection key, the clients' path selections in previous epochs enjoy forward secrecy because we require path-selection keys to be rotated periodically. Because fresh keys are generated for each key rotation, compromise of a path-selection key will not impact the keys from past epochs. However, it will reveal the paths selected by circuit creation through the compromised relay *during* the time the current path-selection key was valid, even if the circuit was created before the compromise itself. (It will *not* reveal the communication encryption keys, however; those still enjoy immediate forward secrecy.) Such a "windowing" approach to forward secrecy is well established for privacy protocols in practice [32,45], and allows for a slight relaxation in forward secrecy in exchange for improved performance or functionality.

To indicate when circuit extension should terminate, the client will also send a TTL (time to live) integer value θ along with sending g^x and g^d. Each hop on the circuit will decrement θ by one. The relay that receives $\theta = 0$ will be the final relay, and will not extend the circuit further. Note that while θ is sent in cleartext, the ability for an adversary to effectively use this information becomes more difficult as the network size and number of participating clients grows, as TTL information is only useful so long as the adversary can perform an end-to-end correlation attack simply by observing exposed network information.

Definition 2. (Single-Pass Walking Onions) As before, we label each step with P to denote a path selection operation, and K to denote a key exchange operation for circuit extension. Let n denote the desired length of the circuit. Recall that the client begins with authenticated information about the relay R_1 (see Section 5.6).

Let (b_j, g^{b_j}) denote the long-term key and $(c_j, g^{c_j}) \leftarrow KeyGen_{VRF}(1^\lambda)$ denote a path-selection keypair (rotated periodically) for the j^{th} relay in a given path, where $1 \leq j \leq n$.

Let H denote a cryptographic hash mapping to $\mathbb{Z}_q^*$, where q is the prime order of the group generated by g.
When the client initializes a circuit:

1. [K] Generate an ephemeral Diffie-Hellman keypair $(x, g^x) \leftarrow KeyGen_{Auth}(1^\lambda)$. This key will be used for deriving *circuit-related* session keys.
2. [P] Generate another ephemeral Diffie-Hellman keypair $(d, g^d) \leftarrow KeyGen_{VRF}(1^\lambda)$. This key will be used for deriving *path-related* VRF inputs.

3. [P] Select a circuit extension time to live (TTL) $\theta = n - 1$, where n is the desired circuit length
4. [P, K] Send (g^x, g^d, θ) to R_1

When R_j $(j \geq 1)$ receives a circuit extension request $(X, D, \theta > 0)$, where X and D are the iteratively reblinded versions of the client's original g^x and g^d public keys:

5. [K] Calculate an ephemeral Diffie-Hellman circuit keypair $(y_j, g^{y_j}) \leftarrow KeyGen_{Auth}(1^\lambda)$; compute the circuit shared secret and authentication value $(S_j, A_j) \leftarrow ComputeSecretAndAuth(X, y_j, b_j)$
6. [P] Derive the VRF output $(\beta_{j+1}, \pi_{j+1}) \leftarrow Prove(c_j, D^{b_j})$ using the relay's path-selection private key c_j and private key b_j. Obtain $i_{j+1} = \beta_{j+1} \bmod \alpha$ to determine the next relay in the path R_{j+1} within the required index range.
7. [P] Obtain the SNIP Σ_{j+1} whose index range contains i_{j+1} in the most recent ENDIVE. This determines the relay that will serve as R_{j+1}.
8. [P] Compute the blinding value for the circuit public key $r_j \leftarrow H(S_j)$
9. [P] Compute the blinding value for the VRF input $v_j \leftarrow H(D^{b_j})$
10. [P, K] Send $(X^{r_j}, D^{v_j}, \theta - 1)$ to the next relay R_{j+1}
11. [P, K] Wait for a response ρ_{j+1} from R_{j+1}; reply to the circuit extension request with $\rho_j = (g^{y_j}, A_j, E_j[\Sigma_{j+1}, \beta_{j+1}, \pi_{j+1}, \rho_{j+1}])$, where E_j is authenticated encryption with circuit keys derived from S_j.

When R_n receives a circuit extension request $(X, D, 0)$:

12. [K] Generate an ephemeral Diffie-Hellman circuit keypair $(y_n, g^{y_n}) \leftarrow KeyGen_{Auth}(1^\lambda)$; compute the circuit shared secret and authentication value $(S_n, A_n) \leftarrow ComputeSecretAndAuth(X, y_n, b_n)$
13. [K] Reply with $\rho_n = (g^{y_n}, A_n)$, and derive circuit keys from S_n.

Recall that the client knows g^{b_1} and g^{c_1}, as above. When the client receives a reply indicating the circuit has been constructed, for $1 \leq j \leq n$, do:

14. [K] Extract (g^{y_j}, A_j) from ρ_j and compute the shared secret S_j with this relay as $(S_j, V_j) \leftarrow ComputeSecretAndValidate(x \cdot \prod_{k=1}^{j-1} r_k, g^{y_j}, g^{b_j}, A_j)$, aborting if $V_j \neq 1$. (Note that the product simply evaluates to 1 if $j = 1$.) Derive circuit keys from S_j. If $j = n$, stop here; the circuit was successfully built.
15. Compute the VRF input as $\tau_j = (g^{b_j})^{\delta_j}$, where $\delta_j = d \cdot \prod_{k=1}^{j-1} v_k$, and compute the blinding value $v_j = H(\tau_j)$. (Recall d was chosen in Step 2.)
16. [K] Decrypt the remainder of ρ_j using the circuit keys. Abort if the decryption fails or if $Verify(g^{c_j}, \beta_{j+1}, \tau_j, \pi_{j+1}) \neq 1$. Otherwise compute the blinding value $r_j \leftarrow H(S_j)$.
17. [P] Verify the the authentication tag of SNIP Σ_{j+1} and that its index range contains $(\beta_{j+1} \bmod \alpha)$, aborting if not.
18. [K] Extract $g^{b_{j+1}}$ and $g^{c_{j+1}}$ from Σ_{j+1}.

Scalability Goals. As with Telescoping Walking Onions, Single-Pass Walking Onions fulfills the scalability goals of Section 3.2, as clients do not require a complete network directory document. Furthermore, in Single-Pass Walking Onions, clients experience only a single round trip, which can be important on a high-latency connection.

We next assess the extent to which Single-Pass Walking Onions achieves its intended security goals.

5.3.1 Analysis of Security Goals

Correctness: While the client does not directly select the next relay for the circuit, the client does receive proof that the relay was selected at random according to the desired relay distribution, and that the selection was generated using the client's original randomly selected ephemeral g^d and the relays' long-term and path-selection keys.

Security: As with Telescoping Walking Onions, so long as each relay on the path is selected from a random distribution in a way that only depends on the client's choice of randomness (and not a specially crafted value from an intermediary), the client can be assured that no intermediary has influenced their path selection. Here, it is important that each relay's path-selection key is committed to in the SNIP corresponding to that relay *before* the relay is ever sent the client's ephemeral key material, so that the relay cannot bias the VRF output. Furthermore, clients are protected against epistemic attacks, as all clients select any given relay with the same probability. Finally, as above, a malicious relay can bias the distribution of a client's path by performing a selective denial-of-service attack against the client's request to extend their circuit, thereby increasing the probability that a client's successfully established path includes malicious relays [5]. However, this risk in a Single-Pass Walking Onions setting is no worse than in existing onion routing schemes where on-path relays refuse client connections.

Privacy: As Single-Pass Walking Onions uses the Sphinx reblinding technique to modify the client's public key material seen by each hop on the path, an intermediary with access to all messages passing through the anonymity network will not be able to bitwise correlate public key material for separate hops in the same circuit. Furthermore, only the client (with knowledge of their private key d) can derive the VRF input for all hops in the path, so long as the relays' private keys are not compromised (and the relays do not collude).

5.4 Tradeoffs Between Protocols

We now discuss the performance and security tradeoffs between Telescoping and Single-Pass Walking Onions relative

Table 1: Tradeoffs: Telescoping, Single-Pass, Current Tor

●=achieved; ○=not achieved; ◐=partially achieved
◇=performance property; †=security property

		Telescop.	Single-Pass	Current Tor
◇	Constant-size client download	●	●	○
◇	One round trip per circuit built	○	●	○
†	Complete client control of relays selected	◐	○	●
†	Forward-secret relay selection	●	◐	●
†	Forward secrecy for data	●	●	●
†	Relays unaware of their positions in paths	◐	○	◐

to the path-selection and circuit-construction protocols used by Tor (further described in Section 2). We summarize these tradeoffs in Table 1.

Performance Tradeoffs. We evaluate performance of the Walking Onions protocols in Section 7, but summarize these tradeoffs here. As Telescoping and Single-Pass Walking Onions do not require clients to maintain a network directory document, both protocols offer improved performance over current Tor in bandwidth and storage requirements for clients, as the number of relays increases. However, Telescoping Walking Onions requires a quadratic number of messages and a linear number of round trips from the client to construct a circuit relative to the number of hops in the circuit. As such, Telescoping Walking Onions matches the message complexity of Tor for circuit construction. Conversely, Single-Pass Walking Onions creates circuits with a linear number of messages and a single round trip from the client, requiring less latency from a client's perspective. However, Single-Pass Walking Onions requires additional computation at each hop due to additional key-blinding operations.

Security Tradeoffs. Telescoping Walking Onions offers partial client control over the selection of relays, as the client can select only i but has no information about the relay, unlike current Tor. This tradeoff may be consequential if the client maintains many path restrictions and thus requires more information about relays during path selection (see Section 6). Telescoping Walking Onions provides the same levels of forward secrecy as current Tor for client communication as well as the selection of relays for a path. Similarly, Single-Pass Walking Onions provides complete forward secrecy for client communications, but windowed forward secrecy for relay selection after a predetermined period after which relays' path-selection keys are rotated. Because fresh path-selection keys are generated for each key rotation in Single-Pass Walking Onions, compromise of a path-selection key will not impact the security of paths outside of the window of time which the compromised key is used. Notably, Single-Pass Walking Onions improves upon past single-pass circuit designs [9,21,31] (as further described in Section 2) by ensuring immediate forward secrecy for client communication.

Any relay in the first or last position of a circuit can learn its position from its incoming and outgoing traffic. Consequently, in a three-hop path, relays occupying the middle position can also learn their position by process of elimination. However, when paths are longer than three hops, Single-Pass Walking Onions offers a slightly weaker property than Telescoping Walking Onions or Vanilla Onion Routing, as Single-Pass Walking Onions exposes to each on-path relay its distance to the end of the path by revealing the TTL indicator θ. In practice, the ability for an adversary to use this information is correlated to characteristics of the anonymity network.

5.5 Hybrid Walking Onions Protocol

While the Single-Pass Walking Onions protocol allows a client to optimistically build a new circuit, a fallback mechanism is important when a client requires relays with specific properties. For example, a client may require a relay to be in a specific geographic location (see Section 6 for more details on selecting relays restricted by some specific property). To support this case, a hybrid approach can be used, where instead of building the complete circuit with Single-Pass Walking Onions, the client uses Single-Pass Walking Onions to only select $n-\ell$ relays for the circuit, and then uses the Telescoping approach to specify the remaining ℓ relays. With this approach, the client trades some of the performance benefit from Single-Pass Walking Onions for additional control over the selection of the ℓ relays in which Telescoping is used to extend the circuit.

5.6 Bootstrapping the First Connection

Walking Onions assumes clients have sufficient information to establish a connection to the first hop in the path. This problem is not unique to Walking Onions; all anonymity networks require that new clients have a mechanism to connect to the network. As such, anonymity networks using Walking Onions have several options for clients to bootstrap. Here we describe two options, but note that such bootstrapping is comparatively infrequent, as clients in networks like Tor fix long-lived "Guard" relays for this first position for up to six months at a time to prevent enumeration attacks [15,43].

Building from trusted relays. Many anonymity networks, such as Tor, hardcode a list of stable relays in the client software as a bootstrapping mechanism [13,23]. To bootstrap a first connection in Walking Onions, clients can build a circuit using one of these pre-configured relays as the first hop. After the circuit is complete, clients can extend it to an additional relay that is suitable to serve as the first hop for future circuits. (We discuss how clients can choose relays for different criteria in Section 6.) The client can then remember this relay, throw away the circuit, and build fresh circuits with this relay as their first hop. This mechanism relies on the same security assumptions as when sending traffic through multi-hop circuits; that is, that the cost to perform end-to-end correlation between the client and its destination is sufficiently high.

Private Information Retrieval. Using PIR for client bootstrapping in anonymity networks is well established in the literature (as further described in Section 2), such as using a set of PIR servers to serve a subset of relay information to Tor clients [30, 34]. A similar approach can be used to bootstrap the first hop of a circuit with Walking Onions. For example, the client software can include a set of PIR servers, which clients can query to obtain one or more SNIPs to use as the first hop.

6 Complex Path Requirements

Until this point, we have presented path-selection protocols assuming that all clients would always be selecting their next relay from a single probability distribution; for example, weighted by bandwidth. However, clients often have more complex path requirements. For example, many networks restrict which relays can be selected for specific positions in a client's path, such as for the entry or for the exit positions. We now present several mechanisms to accommodate complex path requirements in Walking Onions.

Optimistic Attempt and Discard. A simple approach that is acceptable when most relays support a given property is to allow the client to optimistically build a circuit, but discard it if the resulting path is not suitable. For example, if a client requires a circuit whose final relay supports a common property such as forwarding traffic to port 80 (http), they can simply discard circuits ending with relays that do not fulfill this property. However, if a client requires a less well-supported port, such as port 25 (smtp), this approach is more costly.

Multiple Index Ranges. As discussed in Section 4.2, every SNIP assigns a relay to a range of index values. The size of the index range in a relay's SNIP corresponds to the probability of a client selecting that relay. Clients, however, may need to select relays according to different probability distributions, depending on the purpose the relay is to serve. In that case, each SNIP would contain ranges for multiple indices, each index corresponding to one of the probability distributions.

For example, Tor clients currently pick relays for the first and last hops of their paths with different probability distributions than they use when choosing middle nodes. Walking Onions might implement this by giving each SNIP a separate index range for each of the three probability distributions. (See example in Appendix B.)

Note that if there are multiple index ranges, the client must send not only the index value i during circuit construction, but additionally an identifier ψ indicating *which* index to use.

Grouping by Class. When the number of properties grows, however, representing each property as its own index range would result in a linear growth of the SNIP. For example, in Tor, relay operators can list ports to which their relay will forward traffic exiting the network. If each index range is encoded as 8 bytes in a SNIP, representing 65,535 TCP ports as separate index ranges would increase *each* SNIP's size by 524.28 KB: clearly too much. In this case, we can reduce the number of properties by grouping exit ports into *port classes*, putting two ports are into same class when every relay either allows both or denies both. In an analysis of a snapshot of the current Tor network, we find that the 65,535 ports can be grouped into only 220 port classes. Each property in the SNIP then corresponds to a port class. Furthermore, the designers of an anonymity network can enforce the number of properties to be below some threshold by restricting the flexibility of which combinations are allowed.

Representing Properties in Merkle Trees. The strategy of using one index range to represent a class of properties in the SNIP still results in linear growth relative to the number of properties represented in the SNIP. To address this issue, and to keep the size of the SNIP independent of the number of properties, the authority need not place the per-property index ranges directly into each SNIP. Instead, for each relay, the authority can construct a Merkle tree whose leaves are the per-property index ranges for that relay, and only encode the root of that tree into the relay's SNIP. (The default index range for each relay, not associated with a particular property, should always explicitly appear in the SNIP for reasons we will see shortly.) To enable relays to construct proofs demonstrating that other relays fulfill a specific property, the authority should additionally encode the properties each relay supports into the ENDIVE for the network. Using each property and every SNIP in the ENDIVE, relays can deterministically recompute the per-property index ranges locally to reproduce the Merkle trees for each relay in the network.

With this approach, SNIPs remain constant sized as the number of properties grows, while the bandwidth needed for circuit construction when specifying a particular required property increases by the length of the Merkle path, logarithmic in the number of properties.

Delegated Verifiable Selection. Sometimes, the client does not wish to reveal their required property ψ to an intermediate relay R_n for the selection of a relay R_{n+1}. For example, a client wishing to connect to a relay that allows forwarding to a specific port number may not wish to reveal this property to an intermediate relay in the path during circuit construction. We now discuss how to handle this case using a technique we call *Delegated Verifiable Selection*.

After extending a circuit to relay R_{n+1}, if the client requires R_{n+1} to fulfill a specific property ψ, the client sends ψ to R_{n+1} through the circuit (thereby ensuring no intermediate relay can learn ψ). If R_{n+1} supports ψ, the circuit extension to R_{n+1} is considered complete and usable. Otherwise, R_{n+1} computes an index i^* (taken $\bmod \alpha$) derived from the hash of the client's messages to R_{n+1} so far in the protocol. Relay R_{n+1} then selects the SNIP Σ whose index range for property ψ contains i^*. R_{n+1} replies to the client with Σ. Upon receiving the recommended relay represented by Σ, the client will destroy the circuit and then build a fresh circuit using the relay corresponding to Σ in the $(n+1)^{\text{th}}$ position. The client describes this relay using a new index i' sampled from Σ's default index range, to avoid linkability with i^*.

Note that a limitation of this technique is it requires either Telescoping or Hybrid Walking Onions, so the client can directly specify the relay corresponding to their desired index.

7 Evaluation

We now compare the performance of Walking Onions protocols to Vanilla Onion Routing. We test the hypothesis that Walking Onions offers lower bandwidth use than Vanilla Onion Routing, and comparable performance in terms of CPU. Further, we assess the extent to which Single-Pass Walking Onions improves circuit creation latency for clients.

All of our code, data, and analysis scripts are available at https://git-crysp.uwaterloo.ca/iang/walkingonions.

7.1 Bandwidth Evaluation

In order to compare the bandwidth used by our Walking Onions protocols to that of Vanilla Onion Routing, we implemented an onion routing network simulator, which we describe next.

Description of Simulator. The simulator we implemented in Python for this evaluation (available at the url above) models authorities, clients, and relays, using either Vanilla Onion Routing or one of the Walking Onions protocol variants we describe above. The simulator allows relays to register their descriptors with the authority for the network each epoch, and simulates both client and relay churn. Parties obtain consensus documents or ENDIVEs (depending on the protocol) at each epoch, using diffs if they have been online recently. Clients construct circuits through the network using Vanilla Onion Routing, Telescoping Walking Onions, or Single-Pass Walking Onions, using either the Merkle or threshold signature SNIP authentication methods from Section 4.3. We do not simulate sending data through the constructed circuits, as that operation is unchanged between Vanilla Onion Routing and Walking Onions. Our simulator sends messages among authorities, clients, and relays, serializing the messages using Python's `pickle` functionality. Messages that we simulate include those to download consensuses and ENDIVEs (and their diffs), as well as those to create, extend, and tear down circuits, whose instantiations depend on the circuit-extension protocol under use, and many others.

Our simulator can be configured with a number of empirical parameters, outlined in Table 2, and discussed next. We safely estimated the values of these parameters on the current Tor network. We configure the scale of the simulator with the parameter ζ, such that $\zeta = 1$ corresponds to 6,500 relays and 2,500,000 clients, roughly the scale of today's Tor network.

Table 2: Empirical parameters for our onion routing network simulator, with values modeled after those in the current Tor network. ζ is a free parameter specifying the scale of the simulation.

A	Number of authority voters	9
R	Target number of relays	$6,500 \cdot \zeta$
C	Target number of clients	$2,500,000 \cdot \zeta$
N_H	Number of hops per circuit	3
η_R	Relay churn rate	0.010
σ_R	Relay churn stddev	0.003
η_C	Client churn rate	0.16
σ_C	Client churn stddev	0.04
γ	Average number of circuits created per client per epoch	8.9
P_Δ	Average size of consensus diff compared to full consensus	0.019

We denote the number of authority voters as A, to which every relay must upload its descriptor every epoch. R and C are the target steady-state number of relays and clients respectively, which will depend on ζ. Circuits are N_H hops long. To model relay churn, in each epoch, each relay will leave the network with probability η_R, and a number of relays selected from the normal distribution $N(\eta_R \cdot R, \sigma_R \cdot R)$ will join the network. (This causes the steady-state number of relays to be R on average, as desired.) The parameters η_C and σ_C similarly model client churn. The average number of circuits created by each client per epoch is γ: the higher this parameter, the more "active" clients are. The average fraction of data required to send a diff of a consensus (or ENDIVE, in Walking Onions) is P_Δ, as compared to sending the entire document. We measure the number of bytes sent and received by each client and relay in each epoch. We only consider the measurements once the network has reached steady state; that is, we ignore the initial epochs in which *all* relays and *all* clients join the network and all have to bootstrap at once.

In addition to using our simulator to measure the number of bytes actually transmitted by each relay and client per epoch, we also compute *analytical* formulas to predict what these measurements should be, in terms of the network scale ζ, the empirical parameters in Table 2, and the byte sizes of each message type when serialized by `pickle`. The complete formulas are in the file `analytical.py` at the URL above.

We note that our `pickle`-based serialization is much more efficient than the actual Tor message format for consensus documents, which is text-based. For example, a real Tor client requires approximately 2.5 MB to bootstrap a complete consensus on the current Tor network, and 48 KB to keep it up to date each epoch. The more efficient `pickle`, on the other hand, allows our Vanilla Onion Routing clients to bootstrap a complete consensus in just 1.4 MB and keep it up to date with 27 KB per epoch (at $\zeta = 1$). Our byte counts therefore

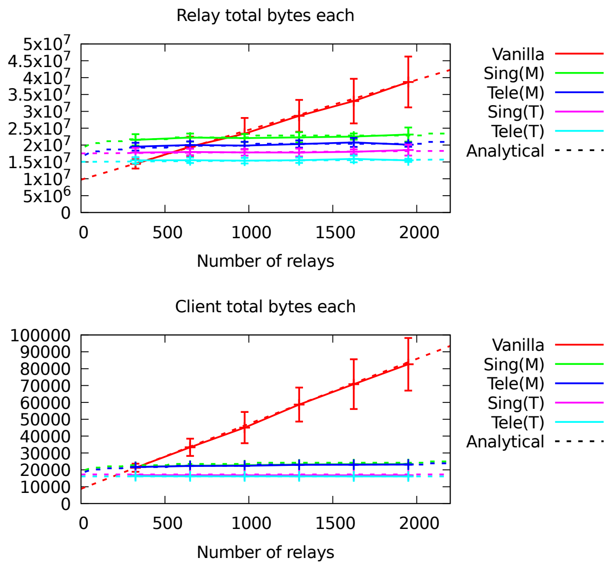

Figure 1: Per-epoch total bytes used for each relay and client, for *Vanilla* Onion Routing, and *Sing*le-Pass and *Tele*scoping Walking Onions. (M) indicates Merkle authentication, and (T) indicates threshold signature authentication (Section 4.3). We plot the means and stddevs from the simulation with solid lines, and the analytical formulas in dashed lines. Note that the Single-Pass and Telescoping lines are almost coincident in the client graph.

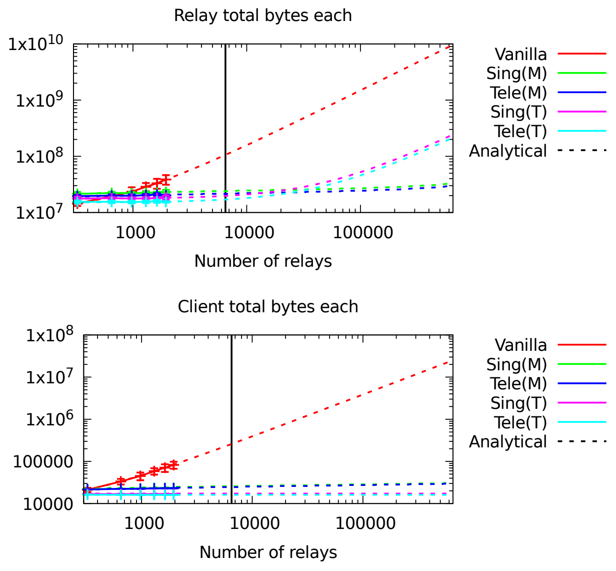

Figure 2: A zoom out of Figure 1, showing the asymptotic behaviour of each of the circuit construction protocols. The vertical line is the size of the current Tor network. Note the log-log scale.

underestimate actual Tor usage, but are directly comparable with *each other*. Importantly, if Walking Onions beats Vanilla Onion Routing in our measurements, it is even that much better than current Tor.

Evaluation Results. In Figure 1 we plot both the numbers of bytes per epoch measured in our simulations, as well as the analytical formulas. The largest simulations we ran were with $\zeta = 0.30$; simulating each Vanilla Onion Routing epoch at that scale took a little over one day and up to 80 GB of RAM. We find excellent agreement between the simulation results and the formulas over the range of ζ values we simulated, supporting that our formulas do not miss any important terms. We then use the formulas to analyze the bandwidths used by Walking Onions circuit creation for larger network sizes in Figure 2 (note the log-log scale).

As we can see in Figure 2, for relays, at the current network size, each relay already uses 4.4–6.2$\times$ (depending on which version of Walking Onions is used) less bandwidth to bootstrap, keep up to date, and construct circuits for clients, as compared to Vanilla Onion Routing. As the network size grows, the difference becomes even more stark. At 10 times the current network scale, each relay uses 24–41$\times$ less bandwidth for these tasks with Walking Onions than with Vanilla Onion Routing. While all five formulas for the average bandwidth used per relay are technically asymptotically linear, the coefficients are very different: almost 15,000 additional bytes per epoch per relay are used by Vanilla Onion Routing, and just 6.75 (2200$\times$ less) by Telescoping Onion Routing with Merkle authentication. As seen in Figure 2, the Merkle versions of Walking Onions are basically constant up to networks two orders of magnitude larger than today's Tor.

For clients, the situation is even better for Walking Onions. For Walking Onions with threshold signatures, the per-client per-epoch cost is *constant*, and with Merkle signatures, it is *logarithmic* in the number of relays in the network. Vanilla Onion Routing, however, is *linear* in the number of relays. Even at the current network size, Walking Onions uses 10–16$\times$ less bandwidth then Vanilla Onion Routing for bootstrapping, keeping up to date, and constructing circuits. At 10$\times$ the current network size, that jumps to 90–155$\times$.

Further, we note that when considering a network with many "on-demand" clients—those which infrequently use the network but must still be prepared to construct circuits—Vanilla Onion Routing proves more costly as more relays are added to the network, due to the fact that idle clients must continue to sync network state. On the other hand, Walking Onions maintains a constant overhead for on-demand clients regardless of a client's usage pattern. Considering such on-demand client behaviour is important for mobile applications that are used infrequently, unpredictably, or have low bandwidth, such as browsers [37] or messaging clients.

7.2 Latency Evaluation

We now evaluate the latency incurred by clients in both Vanilla Onion Routing and Telescoping and Single-Pass Walking Onions. We assess the latency that is experienced by clients downloading an up-to-date copy of the latest network directory, and then as the latency experienced by clients when building circuits (after they have bootstrapped).

Latency Incurred During Bootstrap. As observed in Figure 2, the asymptotic per-client traffic between Vanilla Onion Routing and Walking Onions diverges significantly as the number of relays in the network increases. The size of the network directory in Vanilla Onion Routing is linear in the number of relays, and this behaviour reflects the amount of data that clients must download upon bootstrap. Consequently, a client bootstrapping using Vanilla Onion Routing will incur significantly higher latency as the network size grows, assuming a constant rate of bandwidth over the client's connection. In comparison, clients in Walking Onions download only a constant amount of information during bootstrap, and thus the latency incurred for clients bootstrapping in a Walking Onions setting remains unchanged as the network size grows.

Latency Incurred During Circuit Build. The primary savings we expect for Single-Pass Walking Onions over Telescoping or Vanilla Onion Routing is in the latency experienced during circuit construction. The improvement of Single-Pass Walking Onions is the same as that of other single-pass circuit construction proposals: whereas Vanilla Onion Routing and Telescoping Walking Onions both need a total of $N_H(N_H+1)$ messages before the circuit can be constructed, Single-Pass Walking Onions uses only $2N_H$.

For clients with high-latency connections, the expected benefit is even greater: the number of messages sent and received by the client over their local link is just 2 in Single-Pass Walking Onions, as opposed to $2N_H$ in Telescoping Walking Onions or Vanilla Onion Routing.

7.3 CPU Evaluation

We now evaluate the CPU cost for circuit extension as well as the cost in CPU and memory of generating and validating ENDIVEs. We do not consider the CPU overhead for clients and relays to download and validate network directory documents, as circuit-related operations will dominate for workloads modelled after a live network in production.[3] Finally, we evaluate only public-key group operations, and assume the overhead for symmetric-key and hashing operations is negligible in comparison.

We summarize our analysis for circuit extension in Table 3 and ENDIVE generation and validation in Table 4.

CPU Cost to Extend a Circuit. For specificity, we instantiate the generic authenticated key exchange and VRF functions from Section 5 with ntor [17], requiring two public key operations for both clients and relays, and the IETF-proposed VRF [18], which requires three each.

Notably, Telescoping Walking Onions incurs the same number of group operations for both clients and relays participating in a circuit extension as Vanilla Onion Routing. For Single-Pass Walking Onions, additional group operations are required for the blinding and VRF computations.

[3]In a live network, the total number of circuits created by clients will typically far exceed the number of relays.

Table 3: Number of group operations to construct a circuit, not considering SNIP validation cost

	Protocol	Per circuit
Clients	Vanilla Onion Routing	$3N_H$
	Telescoping WO	$3N_H$
	Single-Pass WO	$6N_H-2$
Relays	Vanilla Onion Routing	3
	Telescoping WO	3
	Single-Pass WO	3 (last relay); 9 (other relays)

Table 4: Costs of SNIP Generation/Validation, and Auth Size

CPU cost measured in public-key operations

	Cost to generate (per voter, per ENDIVE)	Cost to validate (per SNIP)	Authentication tag size (per SNIP)
One-Per-Voter	N_R	N_V	N_V signatures
Merkle Proof	1	0	$\lceil \lg N_R \rceil$ digests
Threshold Signature	N_R	1	1 signature

As described in Table 4, the cost to the client to validate SNIPs after each circuit extension depends on the type of signature included within the SNIP. One point to note is that certain signatures allow for batch processing, allowing the client to jointly verify all SNIPs in Single-Pass Walking Onions; we do not account for this optimization in the above table.

CPU Cost for ENDIVE Generation and Validation. We now evaluate the CPU performance of the authentication mechanisms that we present in further detail in Section 4.3.

The most costly signature to both generate, validate, and store is the One-Per-Voter approach, in which each SNIP is signed individually by each of N_V voters. Note that the cost to each voter grows linearly with the number of relays, and the cost to clients grows linearly with the number of voters.

Merkle signatures are smaller than Threshold Signatures or the One-Per-Voter approach when considering the cost to *transmit* an ENDIVE, as only a single root hash need be authenticated and encoded in the ENDIVE; relays will recompute the Merkle tree on receipt of the ENDIVE to verify the root hash. Furthermore, clients perform fewer public-key operations during SNIP validation, as the Merkle root can be validated just once per epoch when the client receives and authenticates a network parameters document. After this step, validation of SNIPs requires clients to only use hashing operations to validate the Merkle proof to demonstrate inclusion of the SNIP in the ENDIVE.

Threshold signatures provide an attractive option as the cost to validate a threshold signature is a single public-key operation for clients, while the size of the signature remains constant even as the number of voters attesting to the integrity of the SNIP grows. However, the total cost to a single voter to generate a threshold signature for each SNIP scales linearly in the number of relays.

7.4 Comparisons to Other Designs

We now compare the scalability of Walking Onions to PIR-based designs with similar goals to improve the scalability of anonymity networks such as Tor. We include in our analysis PIR-Tor [30] instantiated with both Computational PIR (C-PIR) and Information-Theoretic PIR (IT-PIR) designs, as well as ConsenSGX [34], which relies on trusted hardware. Recall that we expand on the design and security and efficiency tradeoffs of each of these designs in Section 2.

We assume clients request guard node information via an out-of-band mechanism. However, each additional relay role requires a separate PIR database. Consequently, PIR-Tor using C-PIR requires two PIR queries per circuit, as does ConsenSGX. We assume an optimization for PIR-Tor using IT-PIR [30] in which the client performs PIR queries for only the exit node. We furthermore assume that PIR queries can be performed in batch, such that one client request can contain multiple PIR queries.

Bandwidth cost. While PIR-based designs require performing at minimum one PIR query per circuit, Walking Onions requires transmitting the SNIP for each relay during circuit establishment. While the overhead for each PIR design will vary, the bandwidth overhead for Walking Onions will not be greater than C-PIR or IT-PIR based designs, as each design requires the client to perform at minimum one PIR query for each new circuit. For PIR designs based on trusted hardware, Walking Onions queries will be slightly larger as SNIPs contain the index ranges (Section 4.2) not required by these PIR designs. (The PIR designs still require the per-entry authentication tags and validity time fields, however.)

Computational cost. While the CPU cost per circuit construction in Walking Onions remains constant for clients and effectively constant (there may be a binary search to look up the next relay whose SNIP contains the requested index) for relays as the network scales, a server performing IT-PIR or C-PIR must perform work linear in the number of relays. As such, even if the computational cost of these PIR schemes were acceptable today in a network the current size of the Tor network, the cost for these designs increases as the network scales, unlike Walking Onions. Note that the computational cost for ConsenSGX similarly remains constant relative to the number of relays.

Summary. IT-PIR and C-PIR based schemes both scale linearly for client bandwidth and computation as the size of the network grows, while the cost to clients in Walking Onions remains constant. Although ConsenSGX has similar performance benefits to Walking Onions, the dependence of ConsenSGX on trusted hardware is undesirable to many real-world security-critical projects.

8 Conclusion

To provide privacy to everyone on the Internet, anonymity networks must be able to accommodate hundreds of millions, if not billions, of users. To reach these numbers, today's anonymity networks must adopt more efficient protocols.

As a step towards this goal, we present Walking Onions, a set of protocols to remove the per-relay cost to clients in bandwidth and memory as the number of relays grows, and to reduce the latency for new circuit construction. Notably, our protocols offer the same security against route capture and epistemic attacks as prior work requiring a globally consistent network view. We present mechanisms to safely offload path selection from clients to intermediate relays in the client's circuit—even when the client maintains more complex path requirements—without requiring the client to download the full consensus. We evaluate these protocols in terms of bandwidth and CPU relative to a generic onion-routing protocol. Overall, we demonstrate that Walking Onions presents compelling scalability improvements to anonymity networks, allowing such networks to scale while maintaining constant-size bandwidth and memory requirements for network information downloaded by users.

Most importantly, the improvements we present are not just theoretical; The Tor Project has already begun the specification work necessary to integrate Walking Onions into the Tor protocol [44]. We look forward to the future of Tor that will be able to scale to meet the demand of its future users.

Acknowledgments

We thank the reviewers and the artifact evaluation committee for their helpful comments in improving this paper and the accompanying simulator. We thank Peter Palfrader for his original design in proposal 141, and to the designers of PIR-Tor, both of which inspired aspects of this Walking Onions design; David Goulet, Teor, George Kadianakis, and Sajin Sasy for feedback on the earlier Walking Onions proposal; and Jack Grigg for the observation that in a completely decentralized network, SNIPs can be authenticated using a randomly chosen set of relays as an authority mechanism.

This research was supported in part by NSF grants CNS-1526306 and CNS-1619454. We also thank the Royal Bank of Canada and NSERC grant CRDPJ-534381 for funding this work. This research was undertaken, in part, thanks to funding from the Canada Research Chairs program. This work benefitted from the use of the CrySP RIPPLE Facility at the University of Waterloo.

References

[1] Yawning Angel, George Danezis, Claudia Diaz, Ania Piotrowska, and David Stainton. Katzenpost Mix Network

Specification. https://katzenpost.mixnetworks.org/docs/specs/mixnet.html, 2017. last accessed 2019-12-16.

[2] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact Multi-signatures for Smaller Blockchains. In *Advances in Cryptology – ASIACRYPT 2018*, pages 435–464, 2018.

[3] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In *Advances in Cryptology — EUROCRYPT 2003*, pages 416–432, 2003.

[4] Dan Boneh, Ben Lynn, and Hovav Shacham. Short Signatures from the Weil Pairing. *Journal of Cryptology*, 17(4):297–319, Sep 2004.

[5] Nikita Borisov, George Danezis, Prateek Mittal, and Parisa Tabriz. Denial of Service or Denial of Security? In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, pages 92–102, 2007.

[6] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. In *27th USENIX Security Symposium (USENIX Security 18)*, page 991–1008, August 2018.

[7] Ran Canetti and Hugo Krawczyk. Security Analysis of IKE's Signature-based Key-Exchange Protocol. In *In: Proc. CRYPTO'02, Springer LNCS 2442*, pages 143–161, 2002.

[8] Dario Catalano, Mario Di Raimondo, Dario Fiore, Rosario Gennaro, and Orazio Puglisi. Fully Non-interactive Onion Routing with Forward Secrecy. *International Journal of Information Security*, 12(1):33–47, Feb 2013.

[9] Dario Catalano, Dario Fiore, and Rosario Gennaro. Certificateless Onion Routing. In *16th ACM conference on Computer and Communications Security*, pages 151–160. ACM, 2009.

[10] Chen Chen, Daniele Enrico Asoni, David Barrera, George Danezis, and Adrian Perrig. HORNET: High-speed Onion Routing at the Network Layer. In *ACM Conference on Computer and Communications Security*, 2015.

[11] George Danezis and Ian Goldberg. Sphinx: A Compact and Provably Secure Mix Format. *30th IEEE Symposium on Security and Privacy*, pages 269–282, 2009.

[12] George Danezis and Paul Syverson. Bridging and Fingerprinting: Epistemic Attacks on Route Selection. In *Privacy Enhancing Technologies*, pages 151–166, 2008.

[13] Roger Dingledine and Nick Mathewson. Tor Protocol Specification. https://gitweb.torproject.org/torspec.git/tree/tor-spec.txt, 2019.

[14] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The Second-Generation Onion Router. In *USENIX Security Symposium*, 2004.

[15] Tariq Elahi, Kevin Bauer, Mashael AlSabah, Roger Dingledine, and Ian Goldberg. Changing of the Guards: A Framework for Understanding and Improving Entry Guard Selection in Tor. In *Proceedings of the 2012 ACM Workshop on Privacy in the Electronic Society*, WPES '12, pages 43–54, 2012.

[16] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In *26th Symposium on Operating Systems Principles*, SOSP '17, pages 51–68, 2017.

[17] Ian Goldberg, Douglas Stebila, and Berkant Ustaoglu. Anonymity and one-way authentication in key exchange protocols. *Designs, Codes and Cryptography*, 67, 2012.

[18] Sharon Goldberg, Leonid Reyzin, Dimitrios Papadopoulos, and Jan Vcelak. Verifiable Random Functions (VRFs). https://tools.ietf.org/html/draft-irtf-cfrg-vrf-05, August 2019.

[19] I2P Project. I2P Threat Model. https://geti2p.net/en/docs/how/threat-model.

[20] Aniket Kate and Ian Goldberg. Using Sphinx to Improve Onion Routing Circuit Construction. In *Financial Cryptography and Data Security*, pages 359–366, 2010.

[21] Aniket Kate, Greg Zaverucha, and Ian Goldberg. Pairing-Based Onion Routing. In *Privacy Enhancing Technologies*, pages 95–112, 2007.

[22] Akshaya Mani, T. Wilson-Brown, Rob Jansen, Aaron Johnson, and Micah Sherr. Understanding Tor Usage with Privacy-Preserving Measurement. In *Internet Measurement Conference*, IMC '18, pages 175–187, 2018.

[23] Nick Mathewson. Proposal 206: Preconfigured directory sources for bootstrapping. https://gitweb.torproject.org/torspec.git/tree/proposals/206-directory-sources.txt.

[24] Nick Mathewson. Proposal 300: Walking Onions: Scaling and Saving Bandwidth. https://gitweb.torproject.org/torspec.git/tree/proposals/300-walking-onions.txt.

[25] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple Schnorr multi-signatures with applications to Bitcoin. *Designs, Codes and Cryptography*, Feb 2019.

[26] Jon McLachlan, Andrew Tran, Nicholas Hopper, and Yongdae Kim. Scalable Onion Routing with Torsk. In *16th ACM Conference on Computer and Communications Security*, CCS '09, pages 590–599, 2009.

[27] Ralph C. Merkle. A Digital Signature Based on a Conventional Encryption Function. In *A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology*, CRYPTO '87, pages 369–378, 1988.

[28] Silvio Micali, Salil Vadhan, and Michael Rabin. Verifiable Random Functions. In *40th Annual Symposium on Foundations of Computer Science*, FOCS '99, pages 120–, 1999.

[29] Prateek Mittal and Nikita Borisov. ShadowWalker: Peer-to-peer Anonymous Communication Using Redundant Structured Topologies. In *16th ACM Conference on Computer and Communications Security*, CCS '09, pages 161–172, 2009.

[30] Prateek Mittal, Femi Olumofin, Carmela Troncoso, Nikita Borisov, and Ian Goldberg. PIR-Tor: Scalable Anonymous Communication Using Private Information Retrieval. In *20th USENIX Security Symposium*, pages 475–490, 2011.

[31] Lasse Øverlier and Paul Syverson. Improving Efficiency and Simplicity of Tor Circuit Establishment and Hidden Services. In *Privacy Enhancing Technologies*, pages 134–152, 2007.

[32] Trevor Perrin and Moxie Marlinspike. The Double Ratchet Algorithm. https://signal.org/docs/specifications/doubleratchet/.

[33] Michael Reiter and Aviel D. Rubin. Crowds: Anonymity for Web Transactions. *ACM Transactions on Information and System Security (TISSEC)*, 1:66–92, 1997.

[34] Sajin Sasy and Ian Goldberg. ConsenSGX: Scaling Anonymous Communications Networks with Trusted Execution Environments. *PoPETs*, 2019(3):331–349, 2019.

[35] Max Schuchard, Alexander W. Dean, Victor Heorhiadi, Nicholas Hopper, and Yongdae Kim. Balancing the Shadows. In *9th Annual Workshop on Privacy in the Electronic Society*, pages 1–10, 2010.

[36] Fatemeh Shirazi, Milivoj Simeonovski, Muhammad Rizwan Asghar, Michael Backes, and Claudia Diaz. A Survey on Routing in Anonymous Communication Protocols. *ACM Comput. Surv.*, 51(3):1–39, 2018.

[37] The Tor Project. Mozilla Research Call: Tune up Tor for Integration and Scale. https://blog.torproject.org/mozilla-research-call-tune-tor-integration-and-scale.

[38] The Tor Project. Tor Directory Protocol Specification. https://gitweb.torproject.org/torspec.git/tree/dir-spec.txt.

[39] The Tor Project. Tor Metrics—Number of Bytes spent on answering directory requests. https://metrics.torproject.org/dirbytes.html?start=2019-11-08&end=2020-02-06.

[40] The Tor Project. Tor Metrics—Relays. https://metrics.torproject.org/networksize.html.

[41] The Tor Project. Tor Metrics—Total Relay Bandwidth. https://metrics.torproject.org/bandwidth.html?start=2019-11-08&end=2020-02-06.

[42] The Tor Project. Tor Metrics—Users. https://metrics.torproject.org/userstats-relay-country.html.

[43] The Tor Project. Tor Guard Specification. https://gitweb.torproject.org/torspec.git/tree/guard-spec.txt, 2019. last accessed 2019-09-16.

[44] The Tor Project. Walking Onions Specification. https://spec.torproject.org/walking-onions, 2020. last accessed 2020-05-15.

[45] Nik Unger and Ian Goldberg. Deniable Key Exchanges for Secure Messaging. In *22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 1211–1223, 2015.

[46] Qiyan Wang, Prateek Mittal, and Nikita Borisov. In Search of an Anonymous and Secure Lookup: Attacks on Structured Peer-to-peer Anonymous Communication Systems. In *17th ACM Conference on Computer and Communications Security*, CCS '10, pages 308–318, 2010.

[47] Bassam Zantout and Ramzi Ahmed Haraty. I2P Data Communication System. In *ICON 2011*, 2011.

A Applying Walking Onions to Other Anonymity Network Designs

While we use Tor as a case study, in Section 2 we say that Walking Onions can be used by other anonymity networks. We now present one such application.

HORNET [10] presents an onion-routing protocol optimized for performance and is stateless for intermediate nodes in a path. Messages sent through the network are encrypted to each hop such that intermediate hops need only to persist a symmetric key for decrypting packets. Once decrypted, packet headers include all information the node requires for processing the onion-encrypted packet. The protocol requires a one-time setup phase in which the circuit is established in a single pass using a variant of Sphinx [11]. However, HORNET makes tradeoffs in security and assumptions of additional infrastructure that may prove undesirable in practice. We will now discuss how the use of Walking Onions in HORNET addresses two such cases.

First, HORNET assumes the existence of a safe mechanism for distributing path information to the client, such that paths are fully formed and *short* to improve performance over free-routed networks. However, in practice, some anonymity networks may seek to achieve the best of both worlds, to leverage the efficient packet structure and packet transmission techniques presented in HORNET while allowing clients to enjoy as much anonymity as that of a free-routed network such as Tor. As such, Walking Onions can be incorporated into networks utilizing HORNET to achieve efficient distribution of network information and path selection.

Second, to establish a circuit in a single pass, HORNET currently requires the client to use the long-lived public key for each node in a path to encrypt data in the setup phase. While a variant of HORNET allows the node to use an ephemeral key to establish the secret to encrypt client communication, HORNET is not forward-secure for the selection of nodes in a path in either variant. Applying Single-Pass Walking Onions to HORNET improves security of path selection to be eventually forward-secure after a window of time, while retaining the efficiency of establishing a circuit in a single pass.

B An Example ENDIVE

In Section 4.2 we introduced ENDIVEs and SNIPs, and in Section 6 we added the notion of per-property index ranges in SNIPs. Here, we work through an example of how an ENDIVE might be generated for a simple network.

Suppose that we have a network with four relays, R_1 through R_4. These relays have different bandwidths, and different properties:

ID	Bandwidth	Exit?	Entry?
R_1	128	yes	yes
R_2	256	yes	no
R_3	512	no	no
R_4	128	yes	yes

For Vanilla Onion Routing, all of these values are included in a single signed directory document, as in:

Timestamp, Signature

ID	Bandwidth	Exit?	Entry?	Keys, etc
R_1	128	yes	yes	...
R_2	256	yes	no	...
R_3	512	no	no	...
R_4	128	yes	yes	...

For Walking Onions, we would place them in an ENDIVE of independent SNIPs, such that every SNIP has a set of index ranges for each property that the client might want to select. We assume $\alpha = 1024$ for the sake of simplicity, and use bandwidths as weights for the probability distributions.

Timestamp, Signature

ID	Bandwidth	Exit?	Entry?	Keys, etc	Time-stamp	Sig-nature
R_1	0–127	0–255	0–511	...	...	...
R_2	128–383	256–767	∅	...	...	...
R_3	384–895	∅	∅	...	...	...
R_4	896–1023	768–1023	512–1023	...	...	...

Here we have an ENDIVE with four SNIPs. Each SNIP has three index ranges: one default range for selecting general-purpose relays, and two per-property ranges for selecting exit or entry relays, respectively.

Suppose that a client is extending a circuit to general-purpose relay. It picks at random $i = 527$, so the relay extends to R_3 and sends back the corresponding SNIP. The client can verify that i falls in the range 383–895, that the timestamp is live, and that the signature is correct. Thus, the client can be sure that the key information in the SNIP correctly identifies the relay that it chose (obliviously) with its random i.

C Vanilla Onion Routing protocol

In Section 5 we discuss how Walking Onions performs relay selection and circuit construction in comparison to a generic onion-routing protocol which we call *Vanilla Onion Routing*. Furthermore, in Section 7, we evaluate the performance of Walking Onions relative to Vanilla Onion Routing. We now describe the step-by-step behaviour of Vanilla Onion Routing.

K denotes a key exchange operation, and P denotes a path extension operation. Path extension and circuit establishment are distinct operations but performed jointly. Let (b_n, g^{b_n}) denote the long-term key for relay R_n.
When the client extends an existing circuit:

1. [P] Select a random next relay R_{n+1}.
2. [K] Generate an ephemeral keypair $(x, g^x) \leftarrow KeyGen_{Auth}(1^\lambda)$.
3. [P,K] Send (R_{n+1}, g^x) to R_n over the existing circuit.

When R_n receives a circuit extension request (i, g^x):

4. [P] Ensure a connection exists to R_{n+1}.
5. [K] Send the client's g^x to R_{n+1}
6. [P, K] Wait for a response from R_{n+1}, and send it to the client.

When R_{n+1} receives the circuit extension request g^x:

7. [K] Generate an ephemeral keypair $(y, g^y) \leftarrow KeyGen_{Auth}()$; compute the shared secret and authentication value $(S, A) \leftarrow ComputeSecretAndAuth(g^x, y, b_{n+1})$.
8. [K] Reply with (g^y, A); derive circuit keys from S.

When the client receives a reply indicating the circuit was extended:

9. [P] Look up the long-term key g^b for R_{n+1} locally.
10. [K] Complete the handshake: Compute $(S, V) \leftarrow ComputeSecretAndValidate(x, g^y, g^{b_{n+1}}, A)$. If $V = 0$, abort; otherwise, derive circuit keys from S.