Walking Onions: Scaling Distribution of Information Safely in Anonymity Networks

by

Chelsea Holland Komlo

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Masters of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2020

Author's Declaration

Statement of Contributions

Walking Onions first originated as a Tor proposal authored by Nick Mathewson [Mat20c], who is a co-author of this work. In this original proposal, Nick proposed the concept of ENDIVEs and SNIPs, as well as the concept of one protocol variant presented in this work which we call Telescoping Walking Onions. Additionally this proposal introduced the concept of bootstrapping using a circuit through a directory cache.

Our collaboration on the published version of Walking Onions has since been accepted to the USENIX Security Symposium [KMG20].

In his capacity as co-author of our joint work, Nick created the framework for our bandwidth and CPU performance analysis, and used this framework to assess the performance of Telescoping Walking Onions relative to Idealized Tor. Nick also wrote the python script for our real numbers analysis, which we use to compare the performance of Walking Onions in actualized estimates against a simplified Tor-like network. Nick additionally first wrote the example ENDIVE which we present as an example of separate index ranges. Nick also wrote the original protocol description for Telescoping Walking Onions and contributed readability, clarify, and informative edits throughout this work.

Ian Goldberg, my advisor, helped extend this python script to additionally assess the performance analysis of Single-Pass Walking Onions, the second protocol variant presented in our work. Ian additionally helped re-structure our table for the real numbers analysis for clarity of outcomes, and performed stylistic and clarification edits throughout the paper which we submitted for publication.

My unique contribution to this thesis can be summarized as follows. My work includes the Single-Pass protocol variant of Walking Onions, as well as the descriptions and algorithms for managing complex path requirements for both protocol variants. I performed the performance analysis for Single-Pass Walking Onions, and collaborated to create the real numbers table and scenarios presented therein. I performed the CPU analysis for Single-Pass Walking Onions, as well as the analysis for the CPU cost to generate and validate ENDIVEs and SNIPs. I wrote the main takeaways for our performance findings. Further, I performed the analysis comparing the performance in bandwidth of Walking Onions to PIR designs that aim to solve similar scalability issues, specifically considering ConsenSGX, and C-PIR and IT-PIR variants of PIR-Tor. Finally, I contributed clarity and stylistic edits throughout the entire body of our work submitted for publication.

Finally, for this thesis specifically, I am the sole author of the extended background and related work chapters, reviewing these topics in significantly more depth than presented in our publication.

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 evaluate Walking Onions against a generalized onion-routing anonymity network and discuss tradeoffs among the approaches.

Acknowledgements

I would like to thank Dr. Ian Goldberg and Nick Mathewson as my co-authors in the publication of this work. The outcome from this work is exciting and I can't wait to see it put into practical use.

I would also like to thank Dr. Douglas Stebila and Dr. Sergey Gorbunov as readers of this thesis and for their helpful comments to improve and clarify this work.

Dedication

To everyone in the CrySP lab, for being great collaborators and friends. To Matt, for your support and patience. And to my family who always unwaveringly believed in me.

Table of Contents

List of Figures

List of Tables

Chapter 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 [DS08] or intercept users' traffic [WMB10], allowing the attacker to gain sensitive information about users' activities, locations, or even identities. Anonymity networks in practice [DM20] have tried to stop these attacks by requiring all participants to share a *global, 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 severely limits scalability. As discussed by McLachlan et al. [MTHK09], the cost of global knowledge scales quadratically: as the number of clients increases, the number of relays must increase to provide more bandwidth, and all parties must download information about all relays. Clearly, requiring global knowledge is an obstacle to the goal of growing anonymity networks to reach the scale of modern-day browsers [Gab19].

In this work, we present 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 to quadratically. Notably, one protocol we present does not change the security model of Tor, while our second protocol relaxes forward secrecy for path selection (but not for content) from immediate to windowed in order to reduce client latency.

Our design uses a novel path-selection and circuit extension protocol called *Walking Onions*[1]

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

in which clients obliviously choose random paths through the network without prior knowledge about the network's membership or topology, yet can verify the correctness of their paths after they are constructed. In our first variant of Walking Onions, clients perform this oblivious relay selection by randomly sampling an index from a distribution, and receive proof after the circuit has been extended to verify that the new relay in their path corresponds to their random selection. Furthermore, the Walking Onions design can improve efficiency by using a technique from Sphinx [DG09] to create circuits with only a single round trip between the client and all relays on the path. This second variant of Walking Onions utilizes Verifiable Random Functions (VRFs) to perform this random sampling of indices at each intermediate relay, so that even though the client does not perform this random sampling, they receive proof that the choice of relays was in fact random and not influenced by intermediaries. This improvement reduces the latency of circuit creation, as each circuit takes only a linear number of messages to construct, as opposed to quadratic in onion routing networks today.

We verify that Walking Onions does in fact improve the scalability of Tor by simulating relay and client bandwidth overhead, assessing the bandwidth costs of the Walking Onions protocols against Tor's existing onion routing protocol in a network setting comparable to Tor's network today. Our simulations demonstrate that even in a network the size of Tor today, relays in a Walking Onions setting save 4–6 times the bandwidth over Tor today, and in a network 10 times the size of Tor, relays require 25–40 times less bandwidth. The savings for Tor clients in a Walking Onions setting is even larger; clients in Walking Onions save 10–15 times the bandwidth over clients in Tor today, and in a network 10 times the size of Tor, clients in Walking Onions require 90–150 times less bandwidth. Further, we demonstrate the CPU overhead of both Walking Onions variants is either equal or only a slight increase to the CPU requirements of clients and relays in Tor today.

We present several case studies demonstrating the applicability of Walking Onions to both Tor [DMS04] as well as several other proposed anonymity network designs with similar threat models and scalability concerns.

As such, this work aims to prove the following thesis statement:

It is possible to design an anonymity network whose clients maintain a constant amount of network information even as the network grows, and can build circuits through the network that require a linear number of messages relative to the length of the circuit. Furthermore, it is possible that an anonymity network with such a design can remain secure against epistemic and path-based attacks that previously have only been ameliorated by designs that require all participants to retain a globally consistent view of the network.

1.1 Contributions

We present a novel set of efficient path-selection and circuit-extension protocols for anonymity networks using onion routing. 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 (we describe these terms further in Chapter 2).
- Primitives to efficiently and verifiably transmit relay information, and a comparison of several 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.
- Evaluation of these protocols' bandwidth and CPU consumption in comparison to a generalized anonymity network modeled after Tor.
- An assessment of how these protocols perform on the Tor network today and at scale.

1.2 Organization

The remainder of this work is organized as follows. In Chapter 2, we review important concepts that are useful to build understanding that is foundational to our work. In Chapter 3, we review prior work in the literature and in practice that aims to address similar scalability and security goals for anonymity networks as Walking Onions, and we discuss existing gaps in prior work that Walking Onions should address. In Chapter 4, we discuss a high-level overview of Walking Onions, including our assumed threat model, as well as the assumption of a network authority. Furthermore, we review both scalability and security goals that Walking Onions is designed to achieve. Furthermore, in this chapter, we discuss key insights that are important to the construction and design of Walking Onions.

Next in this work, we present Walking Onions protocols in greater depth. In Chapter 5, we introduce notation and terminology, and a mechanism for encoding and distributing network information throughout an anonymity network implementing Walking Onions. We additionally present a brief discussion on mechanisms to authenticate network directory documents. In Chapter 6, we introduce two protocol variants in Walking Onions for path selection and circuit construction, and discuss the tradeoffs of each when compared to the current Tor protocol. In

Chapter 7, we present several mechanisms for path selection to facilitate more complex relay selection requirements.

The remainder of this work includes analysis of the effectiveness and generality of Walking Onions. In Chapter 8, we present a performance analysis of Walking Onions as compared to a Tor-like network, including evaluations for incurred bandwidth, CPU, and latency for both clients and relays. In Chapter 9 we discuss alternative applications of Walking Onions to anonymity network designs beyond Tor. Finally, we conclude and summarize main takeaways from this work in Chapter 10.

Chapter 2

Background

In this chapter, we review concepts that will be useful throughout this work as background knowledge to understand the set of Walking Onions protocols and algorithms we present in this work. We begin by reviewing the use of onion routing for transmitting messages across anonymity networks, and provide an overview of the Tor network and protocols [DM20], one of the largest and best-known onion-routing networks. We then discuss notable path-based attacks against anonymity networks, which we later demonstrate Walking Onions to safeguard against. Finally, we review cryptographic primitives and protocols which are used in Walking Onions. Specifically, we discuss Verifiable Random Functions (VRFs) and their use in cryptographic sortition, an operation that allows for members of a set to be verifiably sampled at random. Finally, as Walking Onions requires distributing verifiable network information, the use of multi-party signature schemes is critical in some applications of Walking Onions to ensure distribution of trust in anonymity networks. As such, we conclude with a thorough review of existing multi-party joint signature schemes in the literature and discuss the benefits and tradeoffs among their designs.

2.1 Onion Routing

The concept of Onion Routing was introduced by Goldschlag et al. [GRS96, RSG98] as an iterative method for clients to send information across a network anonymously while ensuring that no intermediary relaying this information or network adversary observing the network can either learn the content of the messages or link the sender and receiver of the messages purely by bitwise analysis of the network messages. Such a use case is important in many real-world scenarios, such as when users wish to hide their web-browsing patterns from either a local internet service provider or—as facilitated by the design of Tor—from the destination web server.

Onion routing uses a set of servers participating in the network (often called *relays*) to exchange messages wrapped in multiple layers of encryption. Appropriately, this message wrapped with multiple layers of encryption is the *onion*, as each layer of encryption can be decrypted only by a specific relay in the path through the network. Specifically, the message is encrypted multiple times, once per each relay in the client's path through the anonymity network. As each relay receives this message, they *strip off* the layer of encryption that is encrypted to a key only they and the client hold, and then forward the message to the next hop (the reverse process is true in the other direction for messages sent from the server to the client, as each hop will iteratively *add* a layer of encryption). In order to establish these encryption keys with each relay in the path, clients negotiate session keys with each relay using public-key key establishment protocols, resulting in the creation of a multi-layered encrypted communication tunnel called a *circuit* used to route traffic through the network.

In order to improve scalability of anonymity networks used in practice such as Tor, Walking Onions must build upon commonly used onion routing protocols, and specifically must limit the amount of relay information required by the client to establish circuits with network relays. However, Walking Onions must simultaneously maintain the capability for clients to be assured that the integrity and confidentiality of their messages are preserved while traversing the network through intermediate untrusted relays.

2.2 Tor

Tor [DMS04] is a low-latency onion-routing anonymity network with 2.5 to 11 million unique users [Tor19b, MWBJ$^+$18] and roughly 6,000 volunteer-run and independently operated relays [Tor19a]. Tor ensures user anonymity by routing traffic along a three-hop circuit through the network, thereby protecting against traffic analysis by adversaries such as a local internet service provider (ISP) or intermediaries that observe internet traffic passing through central internet exchanges. Furthermore, using multi-hop circuits allows Tor to hide the IP addresses of Tor users from destination servers hosting the requested content.

In this work, we will use Tor as a case study to demonstrate the applicability of the Walking Onions protocols. While Tor is the largest of anonymity networks facilitating general internet access today, in order to allow Tor to scale sufficiently to provide service to any internet-connected user, Tor must be able to service an influx of users larger than an order of magnitude, and to add a sufficient number of network relays to provide network bandwidth. For example, as modern web browsers such as Brave and Firefox explore adding Tor support for their users, the Tor network must be able to support these levels of user numbers [Gab19]. As we describe below, the current design of Tor cannot provide service at such a scale while both retaining the same threat model and good user experience in terms of bandwidth and latency. Walking Onions will need to provide a design to achieve just that—the ability to ensure network scalability while protecting users within the current threat model of Tor and while providing a good user experience.

Existing Tor Protocol. Tor itself is composed of multiple protocols, defining how relay information is securely distributed to all participants, how clients build circuits through the network, how the network authorities vote for which relay information is published for the current network epoch, and other network services requiring cooperation. We will now discuss several Tor protocols that are related to our work. Specifically, we will now review Tor's directory protocol, which defines how network information is distributed, and Tor's path selection and circuit extension protocols, which define how clients select relays for, and establish a secure path through, the Tor network in order to exchange data.

Walking Onions will need to build upon these protocols in order to preserve their current functionality and purpose while performing these operations in a more efficient manner in terms of bandwidth and latency experienced by clients.

2.2.1 Tor's directory system

Tor's protocol requires clients and relays to keep up-to-date information about every relay in the network. As such, Tor requires a consistent global view of relay information for every client and relay participating in the network. Up-to-date relay information is provided by *directory authorities*, a trusted set of servers administered by core members of the Tor community [Tor20a]. Every network epoch (in Tor, one hour), relays upload information about themselves to the authorities, who then vote on the status of these relays. The directory authorities combine information provided by the relays with information the authorities and other external parties collect about the relay directly. For example, relays provide information about themselves including their public identity keys and supported versions. Additional information measured by external parties about the relay includes relays' supported bandwidth. This external measurement is done in order to prevent relays arbitrarily inflating information about themselves that could be used to their advantage in the case of a malicious relay, such as the relay's bandwidth capacity [BMG+07]. As we will discuss in Section 2.2.2, relays are not selected by clients uniformly, and so certain characteristics about relays must be independently measured.

After gathering information about every relay in the network and voting on their status, the authorities compute a multisigned *consensus*, which is the agreed-upon network directory document representing the conclusions of the authorities about the state of the relays making up the Tor network. After receiving the consensus, clients and relays can validate the integrity, authenticity, and timeliness of the latest consensus by validating the signatures and timestamp included in the document. After directory authorities publish the consensus for the current epoch, network relays fetch this most-recent consensus directly from the directory authorities. Relays use the consensus directly, but also cache and serve the consensus to clients, in order to avoid overwhelming the directory authorities and distribute the load required to serve this document to every network participant.

To further conserve bandwidth, Tor uses a pair of optimizations to avoid multiple downloads of redundant consensus information. First, to avoid publishing and downloading of relay information which has not changed since the prior epoch, Tor publishes infrequently changing relay information in a set of *microdescriptors*. Each epoch, clients download only the updated microdescriptors which they do not already have locally. Second, clients and relays can provide the digest of an older consensus in their request for a newer consensus, and thereby receive only the changes between the two documents.

Even considering the above optimizations, Tor's existing directory protocol places an undesirable burden on clients in terms of memory and bandwidth as the number of relays in the network increases. Because clients persist information about each relay, every additional relay participating in the Tor network causes the size of the consensus to similarly increase.

Consequently, Walking Onions must balance the two requirements of directory protocols similar to Tor's. On one hand, Walking Onions must ensure clients use only timely and valid network relay information that has been approved by the network authority such as Tor's directory authorities. On the other hand, Walking Onions must provide clients with the ability to maintain less-than-linear growth relative to the number of relays in the network, to ensure clients with limited bandwidth capacity—such as mobile clients—can continue to participate.

2.2.2 Tor's path selection algorithm

Once a client has obtained the latest consensus, the client selects a list of relays to be used 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. A path through the Tor network consists of a *guard* relay, which is the relay occupying the first position on the path, and an *exit* node, which is the relay occupying the last position on the path. Any relays occupying positions between the guard and exit position are the *middle* nodes. As paths through the Tor network are typically three hops in length, most paths will only have one middle node [DM20].

While the earliest versions of Tor's onion routing protocol sampled relays for a path uniformly at random, today, Tor clients sample relays from a weighted distribution [Tor20d, Tor20b] depending on the relay role and position within the path. Furthermore, selection of relays is often determined by the extent to which a relay fulfills some property. For example, Tor directory authorities assign a *flag* label to relays to indicate that the relay meets sufficient requirements to be selected for a specific position within a client's path. When clients sample relays for the guard position in a path, this sampling is weighted by the relay's supported bandwidth, and *also* must have been assigned the Guard flag by the directory authorities (the Guard flag is assigned so long as the relay has fulfilled the required uptime and bandwidth capacity required by Tor for relays in the guard position). Middle nodes are sampled similarly as weighted by bandwidth, but require no additional flags or capabilities. Exit nodes are sampled weighted by bandwidth, but additionally must advertise support in the node's *exit policy*, which describes the port numbers the relay will support forwarding traffic to. [Tor20b, Tor20d].

As such, Walking Onions similarly will need to support more complex path selection requirements than simply selecting relays uniformly at random, while minimizing the amount of information clients are required to know about each relay. Simultaneously, Walking Onions will need to provide the capability for clients to verify that relays have been correctly selected according to their requirements.

2.2.3 Tor's existing circuit extension protocol

Tor uses a *telescoping* technique [DMS04] which establishes circuits 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 this telescoping mechanism allows for forward secrecy for the client communication, as the client negotiates a fresh shared session key with each hop. However, this telescoping technique similarly increases the latency experienced by the client between the start of circuit establishment until it completes. This latency can be attributed to the fact that the client must perform n consecutive request/response round trips to construct an n-hop circuit (one round trip between the client and each hop in the path). Tor attempts to limit this latency by preemptively building circuits in advance for when the client may need them, but in the case that no such preemptive circuits exist (such as when the client is first bootstrapping), the client is forced to experience the full latency required to establish a fresh circuit [Tor20d].

Negotiation of this shared symmetric key between the client and each hop in the path is important to protect against future key compromise. Because each party provides a fresh ephemeral key for each handshake, compromise of one party's private key will not impact either the client's future or past communication, nor the communication between other clients that communicate with a compromised relay. Key establishment in Tor is performed using using a variant of the ntor protocol [GSU12, Mat20b], which allows for a half-authenticated handshake, in which the client can authenticate the relay using the corresponding identity public key for this relay published in the latest consensus. The resulting nested encryption layers (as further described in Section 2.1) for messages sent through circuits in the Tor network provide additional layers of protection against compromise of intermediate relays on the path.

To improve the user experience for clients, Walking Onions should provide a mechanism to *decrease* the experienced latency for clients when establishing a circuit, while preserving the existing security properties for onion-routed networks using telescoping techniques for circuit establishment. Specifically, among other security properties such as confidentiality, integrity, and authenticity of client communication relayed through circuits, Walking Onions should also preserve forward secrecy for client communication in the case of key compromise.

2.3 Attacks Against Anonymity Networks

As discussed in Section 2.2.1, Tor currently follows the approach of requiring a globally consistent view for network directory information, meaning that all clients and all relays must maintain an up-to-date consensus. This approach is required by Tor to protect against a class of path-based attacks, but has clear scalability limitations. While other anonymity network designs have attempted to provide improved scalability, these designs have had undesirable consequences on the security guarantees of the system, which we will further review in Chapter 3. Our work provides a "best of both worlds" approach by ensuring improved scalability without significant changes to Tor's threat model, which we further discuss in in Chapter 4.

We now describe several relevant attacks against anonymity networks, and later in Chapter 3 we will describe several demonstrated examples of other anonymity network designs that do not protect against such attacks. Specifically, we will examine epistemic and route capture attacks and discuss how such attacks result in compromised security and privacy to clients.

We emphasize these attacks specifically because these attacks are protected against in Tor's current design, and it is critical that Walking Onions successfully protects these as well, even while seeking to improve the scalability and efficiency of the network.

2.3.1 Epistemic Attacks

One attack to compromise the privacy of users of an anonymity network is an *epistemic attack*, where an adversary can use information leaked to network observers, such as the frequency of connections to a certain relay or the client's choice of which relays to build paths through [DC06, DS08]. As such, in anonymity network designs where each client has a different, and partial, view of the relays making up the network, which nodes a client selects for their paths can be used as a unique fingerprint. This fingerprint arises from the fact that the views of the relays making up the network differ between each client, and the selection of one relay over another allows an adversary recording network traffic to deduce information about the client's access patterns. Such an attack could result in the adversary eventually piecing together an end-to-end correlation between the client and their end destination.

Tor protects against epistemic attacks by requiring all users and relays to hold a globally consistent view [Tor20a], as all users and relays maintain the identical consensus document. However, as further elaborated in Section 2.2.1, this approach has significant scalability problems. Walking Onions will need to protect against epistemic attacks while improving the scalability of anonymity networks such as Tor.

2.3.2 Route Capture Attacks

Route capture attacks are one variant of path-based attacks, and result in an attacker compromising the integrity of a client's path through the network by replacing or influencing the selection of relays for the path [WMB10]. These attacks can pass undetected if clients' views of the network are not consistent and authenticated. Furthermore, the capability for an adversary to perform route capture attacks depends on the amount of influence intermediate nodes are allowed over the client's path selection when constructing circuits through the network. For example, a system in which each node on a path is responsible for selecting the next node is trivially vulnerable to a route capture attack: a malicious node can always only select other collaborating malicious nodes, yielding an entirely malicious path.

Tor currently protects against route capture attacks using two approaches [Tor20a]. First, the network directory document that distributes relay information to the network is authenticated by the directory authorities, who serve as the root of trust for the network. Second, all clients receive exactly the same relay information, ensuring that a malicious intermediary cannot influence which relay information a client receives or does not receive. If the network directory document were unauthenticated, an intermediary could trivially replace valid relay information with arbitrarily chosen information. If clients could receive different views of the network, without a mechanism to validate their view is trustworthy, an intermediary could arbitrarily inject or withhold relay information from specific clients.

Walking Onions will need to distribute authenticated relay information in an efficient manner that simultaneously protects against path-based attacks. Furthermore, Walking Onions will need to ensure that such an authentication mechanism both ensures that received information is authenticated by the root of trust for the anonymity network, and that their selection has not been influenced by a malicious intermediary.

2.4 Cryptographic Sortition

In Walking Onions, our approach to improving user experience by decreasing the latency clients experience during circuit build time while remaining secure against attacks discussed in Section 2.3 requires that we employ specialized cryptographic primitives. As we will further discuss in our protocol specification in Chapter 6, Walking Onions will build upon the concept of cryptographic sortition—verifiably randomly selecting participants from a global set—to determine which relays are selected for a client's path through an anonymity network.

Cryptographic Sortition typically is instantiated using Verifiable Random Functions (VRFs) [MVR99] as the underlying cryptographic primitive to generate a random output, which we will now describe.

2.4.1 Verifiable Random Functions

Given an input string and a public/private keypair, VRFs produce an output and a proof, of which the output is simultaneously deterministic but unpredictable to anyone without knowledge of the private key used to process the input. However, anyone can use the corresponding public key to verify the correctness of the output using the proof. In short, only the holder of the private key can compute the output, but anyone with the public key can verify this output.

Notably, VRFs define three functions: a function to generate the public/private keypair which we refer to as *KeyGen*, a function that accepts an input string and generates an output and a proof using the secret key which we refer to as *Prove*, and a function which we refer to as *Verify* which determines the correctness of the output given the public key, input string, and proof. We introduce the definition of these functions more formally and their use in Walking Onions protocols in Section 6.1.3.

The desired properties from a VRF are as follows, and are a summarization from the properties as presented in the IETF draft specifying VRFs [GRPV19]:

Uniqueness. When querying an input to *Prove*, each input for any public/private keypair produces a unique and deterministic output.

Collision resistance. Finding two inputs to *Prove* that produce the identical output for a single public key must be cryptographically hard for an adversary.

Pseudorandomness. To an adversary with knowledge of only the public key and the ability to query inputs and observe outputs, every output must be unpredictable to the adversary.

VRFs can be instantiated using different instantiations of public-key cryptographic primitives, such as RSA or elliptic curves, as described in the VRF standardization draft [GRPV19]. The choice of which instantiation to use will depend on the implementation. However, for our definition of Single-Pass Walking Onions, we use an instantiation of a VRF whose public keys come from a prime-order group over an elliptic curve.

2.4.2 Applications of Cryptographic Sortition

Sortition has a number of applications in distributed networks; for example, the Algorand Byzantine agreement protocol uses sortition for leader election. Specifically, leader election in the Algorand Byzantine agreement protocol verifiably selects nodes at random for leader election and to participate in a consensus protocol for the current network epoch [GHM+17]. Using a VRF for the cryptographic sortition of selected users for a given epoch enables users to present proofs publicly to demonstrate their inclusion in the given round.

2.5 Multi-Party Signature Schemes

As mentioned in Section 2.3, Walking Onions requires the ability to authenticate network directory documents in order to ensure clients can validate the integrity of the path they build through the network. However, this authentication mechanism can vary depending on the anonymity network design. Here, we examine several cryptographic primitives that can be used to issue an authentication value as the result of multiple parties cooperating to produce the value, as is required by anonymity networks such as Tor.

First, we look at *aggregate signatures*, which allow n signatures issued by n different parties to be aggregated into a single signature over each document. Next, we look at *multisignatures*, which are a special case of aggregated signatures, as the resulting signature is valid over only a single document that has been independently signed by each participating signer. Finally, we examine *threshold signatures*, which allow t out of n authorized signers to issue a signature over a single document.

2.5.1 Aggregate Signatures

Definition and Instantiation

Aggregate signatures provide an n-out-of-n scheme in which the resulting signature represents the aggregation of n signatures all issued by a different signer [BGLS03b]. Notably, signature aggregation can be performed for signatures over different messages. More formally, an aggregated signature scheme accepts tuples of the form $(Y_i, \sigma_1, m_1), \ldots, (Y_n, \sigma_n, m_n)$, where Y_i is the public key for the signing party, σ_i is the signature over message m_i, and produces a single joint signature Σ over all of the messages $(m_1, \ldots, m_n)$.

Aggregate signature schemes implement four algorithms: *KeyGen*, *Sign*, *Combine*, and *Verify*. *KeyGen*, *Sign*, and *Verify* are standard for any signature scheme. *Combine* accepts a set of tuples, where each tuple contains a message, a public key, and a signature, and outputs a single signature over all of the messages in the set [Bon11]. The design of aggregated signature schemes may allow for random order when performing the aggregation, or may require strict ordering [BGLS03a]. Finally, because the *Combine* algorithm does not require privileged information or access, anyone can perform this operation, not just the participants.

Constructions

Aggregate signatures were first presented by Boneh et al. [BGLS03b] and are among the better-known schemes in the literature, and builds upon the Boneh-Lynn-Shacham (BLS) signature scheme [BLS04]. However, to prevent against rogue-key attacks, all n messages are required to be distinct. One possible mechanism to ensure message uniqueness is for participants to prepend their public key to each message. However, such a simple solution is not viable in a setting where a participant may produce a signature over the same message multiple times.

While the above scheme allows aggregation in random order, other aggregate signature schemes schemes have also been proposed requiring sequential aggregation during the signing stage but random order during the validation stage. Lysyanskaya et al. [LMRS04] present a sequential aggregate signature scheme based on RSA signatures. Bellare et al. [BNN07] address the limitations by Boneh et al. [BGLS03b] and present slight modifications to a construction by Lysyanskaya et al. [LMRS04], ensuring the resulting scheme is unrestricted such that signatures over identical messages remain secure against the rogue-key attack. Lu et al. [LOS$^+$06] present a sequential aggregated signature scheme based on Diffie-Hellman operations and security assumptions, and prove the security of the resulting scheme without requiring the assumption of random oracles.

One important point to note is that to validate signatures resulting from the above schemes, n public keys (one public key from each signer) are required to perform the validation, imposing linear growth of public key material relative to n. Boneh, Drijvers, and Neven [BDN18] present a multi-signature scheme with public-key aggregation, to ensure constant growth of key material as the number of signers increases. Furthermore, this scheme protects against forged messages via the rogue-key attack without requiring unique messages or providing proof of knowledge for the corresponding private key.

2.5.2 Multisignatures

Definition and Instantiation

Multisignatures are a subclass of aggregated signatures, as all parties sign the identical document but each produces a separate signature over this document, which are later aggregated into a single joint signature. As such, multisignatures are a subclass of aggregate signatures. Similar to the more general case, validation of multisignatures requires either all n public keys corresponding to each signer, or a single aggregate public key, if supported by the underlying scheme.

We will now review several schemes more recently proposed in the literature that could be good candidates for anonymity networks with a distributed root of trust, such as directory authorities in the Tor network.

Constructions

MuSig [MPSW19] presents a Schnorr-based multisignature scheme which allows for key aggregation. MuSig uses ECDSA as its underlying signature scheme in order to be fully compatible with Bitcoin. MuSig operates in the “plain public key model” while protecting against rogue-key attacks, meaning that validation of the signature requires only the corresponding public keys. MuSig requires three rounds of interactivity between signers to produce a signature. Although the authors have published a 2-round variant in prior work, the security of this variant is not yet proven under standard assumptions.

Boneh et al. [BDN18] present a modification to MuSig that uses BLS signatures [BLS04] as the underlying signature primitive. Doing so allows for public-key aggregation and fast verification while relying on the plain public-key model and provable security. Furthermore, signing operations can be performed in a single round. Finally, the authors present an accountable variant, allowing for verifiability of the identities of the signers.

2.5.3 Threshold Signatures

Definition and Instantiation

Threshold signatures allow for signing messages under a t-out-of-n trust model, in which n separate entities are pre-authorized to perform signing operations, but generating a signature requires t of these entities to coordinate to produce a single joint signature. The secret key used to produce the signature is distributed over all participants, but only a single public key is used to validate the signature. As such, the use of a threshold signature scheme to generate a signature is transparent to the verifier of the signature, as this signature can be validated using operations standard for validating a signature produced by a single entity.

To limit trust in any one party, threshold signature schemes often make use of a Distributed Key Generation (DKG) scheme to generate the long-lived secret key shares, allowing all participants to contribute equally in the key generation stage.

The algorithms implemented by a threshold signature Scheme are as follows. *KeyGen* generates a single public key and n secret shares (one for each participant). *Sign* requires t participants to coordinate to produce a joint signature. *Verify* allows a single party to verify the correctness of the signature, where the operations followed to perform validation is identical to the underlying signature scheme in a setting with only one participant.

Accountability. Note that one attribute of threshold signatures is that such schemes are non-attributable, meaning that the resulting signature does not disclose which parties participated to create the signature. We discuss the case when both accountability and redundancy are desirable for joint signature schemes in Section 2.5.4.

Complexity of Underlying Signature Schemes. As threshold signatures employ signature schemes as an underlying primitive (such as RSA, Schnorr, or ECDSA), the implementation of the threshold signature scheme will depend on which signature primitive is used. Consequently, the choice of signature scheme in turn impacts the complexity of the threshold scheme itself. We discuss the complexity of these schemes when presenting constructions below.

Constructions

We now provide a brief review of Schnorr and ECDSA-based threshold signature schemes; these schemes could be used by a distributed network authority in Walking Onions to produce an authenticated network directory document in which the signing process requires a threshold number of network authorities.

Schnorr Signatures. Using Schnorr signatures [Sch91] as the underlying signature primitive is by far the simplest and most straightforward for threshold signature designs. As Schnorr signatures utilize a random value for each signing operation, no single party should choose or hold full knowledge of this value, as doing so has security consequences and could lead to one party learning the secret shares for other participants. However, because Schnorr requires only additive operations to this random value during the signing operation, generating Schnorr signatures in a threshold setting consists merely of an iterative summation of partial signatures contributed by t signing parties.

Stinson and Strobl [SS01] introduce a Schnorr-based threshold signature scheme in which key generation and signing are both multi-round interactive protocols. Distributed Key Generation is performed using a scheme by Gennaro et al. [GJKR07] to generate a random shared secret in both the key generation and signing stages. The applicability of this threshold scheme is presented as a use case in a distributed certificate scheme requiring a threshold number of certificate authorities to issue a certificate.

ECDSA Signatures. ECDSA signatures require finding the inverse of a randomly sampled value k and then multiplying the resulting value to the secret key held by the signing party. However, performing these operations jointly across multiple participants without revealing k to any party—while maintaining equal contributions and levels of trust by all parties—adds significant complexity to these schemes as compared to simple variants such as those based on Schnorr signatures.

Gennaro et al. [GGN16] and Gennaro and Goldfeder [GG18] present ECDSA-based threshold schemes instantiated with Paillier [Pai99], an additively homomorphic encryption scheme, and require each player to perform multiplicative-to-additive share conversion subprotocols. Gennaro and Goldfeder [GG18] presents an improvement, moving from requiring $2t + 1$ cooperating signers to only t participants. However, this approach similarly requires converting shares from multiplicative to additive, which in turn requires first a pairwise protocol round between each participant to all other participants, followed by a second protocol round to coordinate generating the joint signature. In total, the signing round requires a five-phase protocol, with three broadcast phases among all players.

Distributed Key Generation Constructions

When performing threshold signing without requiring a central trusted authority (as is in the case of Shamir's secret sharing), using a Distributed Key Generation (DKG) protocol facilitates the generation of a shared secret among all participants, such that all participants know and share the public value, but each participant maintains a unique share of the resulting secret.

The first DKG proposed was by Pedersen, as part of a trustless threshold cryptosystem that does not require a central trusted dealer as is required in secret-sharing schemes such as Shamir's secret sharing [Sha79]. In Pedersen's construction, all parties collectively generate the random shared secret. This operation is performed by each party locally selecting a random value, and then secret sharing this to all other parties, keeping one share for themselves. Each party's share of the collectively shared secret is calculated by the (local) summation of each party's received shares. To prevent against misbehaving participants, verifying the correctness of shares is important in this context, as shares are no longer computed by a central trusted dealer. As the underlying secret-sharing primitive, Pedersen's scheme uses Feldman's Verifiable Secret Sharing (VSS) scheme [Fel87], but with every participant performing the actions of the dealer separately.

However, Gennaro et al. [GJKR07] demonstrate that Feldman's VSS does not completely protect against an adversary adaptively manipulating the output distribution of the secret after observing the inputs from other parties. To avoid this attack, the authors adapt Pedersen's DKG [Ped91] to use an additional *commitment phase* between participants at the start of the protocol. In this commitment phase, every party uses Pedersen's VSS [Ped92] to commit to their random choice before distributing shares to every party, thereby avoiding the adversary from adaptively selecting a more favourable input.

In a separate work, Gennaro et al. [GJKR03] demonstrate that certain threshold cryptosystems remain secure when initialized using Pedersen's DKG [Ped91] even though the distribution of the shared secret may be biased by a misbehaving adversary. The authors prove that the shared secret is *sufficiently* random for secure initialization of certain threshold schemes, and demonstrate one such case using a threshold Schnorr-based signature. However, using Pedersen's DKG has a tradeoff between efficiency and underlying security. For a q-order subgroup of field $\mathbb{F}_p$, achieving the same security level as the scheme presented in Gennaro et al. [GJKR07] requires that q be twice as long and p to be 2^3 times longer. Consequently, the growth of these parameters results in the growth of the cost of computation by a factor of 10. However, as Pedersen's DKG [Ped91] requires only one round of interactivity (where each participant acts as the dealer), as opposed to two as presented in Gennaro et al. [GJKR07], the tradeoff between computational cost and interactivity is one that depends on the use case in practice.

Kate and Goldberg [KG09] present an asynchronous distributed key generation protocol that considers a setting in which participant nodes are separated by a network partition and can crash (and later recover) or the network can fail. Furthermore, the threat model includes a t-limited Byzantine adversary that can perform arbitrary actions outside of the protocol. The protocol definition includes a leader-based agreement scheme and allows for group modification, such as node addition and removal, as well as threshold modification.

2.5.4 Accountable Subgroup Signatures

In an accountable subgroup scheme, t out of v authorized signers can produce a valid signature, and the verifier of this signature can verify which t signers cooperated to produce the signature. Recall that in Section 2.5.3 we discussed how threshold signatures provide this first capability, but not the latter. Consequently, such schemes muse use alternative constructions. Maintaining both accountability and the ability to generate a valid signature with the cooperation of only a subset of signers is desirable in many settings, such as in the case of certificate authorities that cooperate to produce a joint signature over a single certificate.

Micali et al. [MOR01] present the first formalization and construction of accountable subgroup signatures. In accountable subgroup signature schemes, the signature comprises an aggregate signature that is generated by t authorized signers, as well as a set of identifiers for each signer. This scheme ensures accountability as the verifier can determine which of the t (out of n) signers participated in the signing process.

Earlier constructions of accountable subgroup signature scheme represented the public key as the set of tuples of (identifier, public key) for the n valid participants. However, such an approach results in the linear growth of the public key relative to the number of participants. in schemes such as that by Boneh, Drijvers, and Neven [BDN18], the public key is aggregated, thereby limiting the growth of the size of the public key relative to the number of signers.

2.6 Proofs of Inclusion

While signatures provide one mechanism to validate the integrity of a document, proofs of inclusion provide another mechanism, so long as the proof itself includes an authentication tag issued by a trusted authority. To that end, we demonstrate later how Walking Onions can leverage proofs of inclusion to reliably demonstrate information about a specific relay is part of a set of documents published by a trusted authority.

Merkle trees are one common cryptographic primitive that implement such proofs of inclusion, which we now describe in greater detail.

2.6.1 Merkle Trees

Merkle trees [Mer88] are a common solution to provide an integrity value over a set of documents. Merkle trees provide proofs of inclusion termed *Merkle proofs*, which demonstrate that a particular document is in fact part of a larger collection of documents. For example, Merkle trees are used in the Certificate Transparency protocol [LLK13] to demonstrate inclusion of a specific certificate in a publicly auditable log, to protect against certificate authorities minting rogue certificates without oversight.

Merkle trees utilize a binary tree structure; the leaves are a collection of values such that individual values can later be validated to be within the entire collection. Each non-leaf node of the tree is generated by computing a hash of its children. Merkle trees implement the following functionality: *Add*, *Prove*, and *Validate*. The function *Add* accepts an input to add to the Merkle tree, and adds this input as a new leaf to the tree. Doing so requires re-deriving the root hash as well as all hashes on the path from the leaf to the root. Thus, the output of *Add* is a new Merkle tree with the inclusion of the new input. *Prove* accepts an input, and outputs a Merkle proof demonstrating the inclusion of this input in the tree. The Merkle proof demonstrates that this input is a valid leaf in the current Merkle tree, and consists of the path in the Merkle tree starting with the leaf (the input to prove) and demonstrating a path from this leaf through each level in the tree to the root hash. Concretely, the Merkle proof is the set of *sibling* elements to the elements making up the direct path from the leaf to the root node. *Validate* accepts as input the Merkle proof that is generated by *Prove*, as well as a valid root hash for the current Merkle tree. To validate the Merkle proof, each element of the proof is iteratively hashed together, eventually deriving a separate root hash. The Merkle proof is correct if the derived root hash matches the valid Merkle tree root hash.

2.7 Summary

Walking Onions builds upon onion routing to provide a scalable design for path-selection and circuit-establishment protocols for anonymity networks used in practice, such as Tor. However, Walking Onions must ensure that the network can both scale efficiently and maintain safety against known attacks, such as epistemic and route capture attacks. To demonstrate the applicability of our work, we use the Tor network as a case study.

To accomplish these goals, Walking Onions makes use of cryptographic primitives presented in this chapter. First, cryptographic sortition via Verifiable Random Functions (VRFs) provides verifiability of the randomness of another peer's output, given a specific input and a public/private keypair. Such assurances are useful to improving the efficiency for path selection through an anonymity network, as we will see later in the design of Walking Onions. Second, multiparty signature schemes allow multiple parties to cooperate to generate signatures over a document or set of documents. In the setting for Walking Onions, multiparty signature schemes are useful when the trusted authority for a anonymity network is divided among a trusted set of entities. Finally, proofs of inclusion such as Merkle trees provide an efficient mechanism to verify a specific element is included within a larger set. These primitives will be useful when we discuss how networks using Walking Onions can distribute network information more efficiently even as the network grows large, yet guarantee its authenticity and integrity to clients.

Chapter 3

Related Work

In this chapter, we review designs for managing the state of anonymity networks, specifically focusing on designs for distribution mechanisms of up-to-date network directory documents to network participants. We examine designs for completely decentralized anonymity networks in the literature and in practice, and also survey several scalability designs for the Tor network specifically. We summarize the differences in how anonymity network designs protect against known network attacks in a side-by-side comparison of each design against a range of attacks discussed in the literature. Next, we review designs for single-pass circuit creation protocols and discuss tradeoffs between design choices with a specific focus on how these choices impact the security guarantees of the protocol. As Walking Onions should both limit the amount of network directory information required by clients as well as present a circuit creation protocol more efficient in terms of latency, understanding designs that had a similar outcome (linear number of messages relative to the length of the circuit) but undesirable security outcomes is important. We conclude by discussing how Walking Onions can learn from the state of the literature and in which aspects Walking Onions will need to improve in order to offer a compelling alternative to be used in practice by anonymity networks such as Tor.

3.1 Designs for Managing Network Information

In a survey by Shiraz et al. [SSA$^+$18], anonymity networks fall into two categories depending 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. The latter approach can allow for adversary-run intermediate nodes to use this position of influence over the client's path to their advantage, as we will further discuss in our review of such designs. However, source-routed networks, such as Tor, have scalability costs. Because the clients in source-routed networks hold complete control over which relays are chosen for the path, efficiency tradeoffs arise in terms of which information clients must persist locally in order to make these decisions, or latency and bandwidth tradeoffs if the client asks for relay information from a separate entity, such as a set of Private Information Retrieval (PIR) servers, which we further describe below. Walking Onions will need to maintain the best of both approaches, maintaining the security of source-routed designs while the efficiency of hop-by-hop designs.

We will now describe several anonymity network designs, focusing specifically on how network directory information is distributed to clients and how paths are built through the anonymity network. We first describe path-routing and circuit-establishment designs for *centralized* networks, which depend on a root of trust for the network to determine the network's current state. As part of this review, we look at several proposed designs for the Tor network with the intention to improve efficiency over the current Tor design, further described in Section 2.2. We then describe path-routing and circuit-establishment designs for *decentralized* networks, in which all network participants hold equal levels of trust and responsibility when determining the current state of the network.

3.1.1 Centralized Anonymity Networks

While Tor is one of the better-known anonymity networks that require a centralized approach to network information distribution, other anonymity networks also require centralization in order to protect against path-based attacks described in Section 2.3. Walking Onions similarly will need to protect against such attacks, and thus must integrate and work within the threat model of anonymity networks such as Tor.

Katzenpost [ADD$^+$17] is a mix network that implements the Loopix [PHE$^+$17] design, a mix net anonymity network design that provides improved latency guarantees than previous designs. Loopix presents a medium-latency network protocol that utilizes dummy messages for cover traffic and randomized message delays to protect against network timing attacks and a global adversary who can observe end-to-end network traffic. Furthermore, Loopix defines a layered topology, in which nodes of the network are stratified into separate sets and messages only pass through one node per layer. This layered design ensures that links between nodes in each layer are well covered and mix traffic well in a few steps, as traffic passes between nodes that are in separate layers [DSS05]. While Loopix defines the network topology and circuit-establishment protocol, Loopix does not define how network directory information should be collected, authenticated, and distributed to both clients and relays. However, Loopix requires clients to maintain up-to-date information about network relays, as routes through the Loopix network are source-routed; clients themselves select each relay that will become part of their path. As an instantiation of Loopix, Katzenpost requires a globally consistent network view, and utilizes a set of directory authorities for node registration and to distribute authenticated network directory information to all network participants [ADPS16].

We will now review alternative designs to the Tor network that have been proposed to address similar scalability issues that Walking Onions should address, and discuss the tradeoffs with each approach.

3.1.2 Scalability Designs for Tor

Torsk [MTHK09] is a design to improve the scalability of the Tor network by reducing the amount of relay information clients must locally maintain. The Torsk design distributes relay directory information over a Distributed Hash Table, or DHT. This design allows a client to extend a circuit without requiring up-front relay information in order to perform a circuit extension, as selection of the next relay in the circuit R_n is determined by the previous relay in the circuit R_{n-1} performing a DHT lookup. To avoid information leaks resulting from these DHT lookups, each node performs lookups using a secret "buddy nodes" as a one-hop proxy. However, analysis by Wang et al. [WMB10] demonstrates security weaknesses in the method by which buddy nodes are selected in Torsk, leading to possible route capture attacks, as an adversary could gain outsized advantage in the selection of a relay's buddy node.

PIR-based designs

PIR-Tor [MOT$^+$11] 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. Notably, while PIR-Tor allows clients to download less relay information than Tor's current approach of requiring a globally consistent network view, PIR-Tor works within Tor's current threat model and protects against epistemic attacks by hiding from directory authorities and network observers *which* relays the client is requesting. However, PIR-Tor imposes undesirable performance or security tradeoffs depending on which PIR design is used to instantiate the private lookups in PIR-Tor. Specifically, C-PIR designs are costly in CPU resources, and IT-PIR designs require an assumption of non-collusion between PIR servers in order to remain secure. ConsenSGX [SG19] improves upon the resource usage of PIR-based relay selection by using trusted execution environments like Intel's SGX. Both approaches provide control to the client while limiting the amount of data the client must maintain, but are limited by the performance and bandwidth costs of PIR or the security shortfalls of trusted execution environments.

3.1.3 Decentralized Anonymity Networks

While later in this work we use Tor specifically as a case study to demonstrate the applicability of Walking Onions, Walking Onions should also be applicable in the context of a decentralized network. How network directory documents are authenticated will depend on whether the network relies on a centralized or decentralized design. The design of the authentication mechanism itself is out of scope for this work, although later we will discuss ideas that decentralized designs could adopt to facilitate the use of Walking Onions in such a setting. However, we will now review several decentralized network designs in the literature and in practice. Similarly to our analysis of centralized networks, we will examine how network information is distributed and how clients establish paths and circuits through the network.

Crowds [RDR97] relies on a peer-to-peer protocol in which all clients are also nodes in the network. Paths through the network are determined using a "coin-flipping" random-walk technique, where each node forwards requests either to another intermediate node, or directly to the intended recipient, depending on a weighted coin flip, continuing until the message reaches the recipient. Responses are relayed back along the same path in the reverse order. 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.

MorphMix [RP02, RP04] is a peer-to-peer decentralized circuit-based anonymity mix network. Clients exchange messages in MorphMix using multi-layered encrypted tunnels, as in onion routing. Relays in the network have ony a partial view of the entire network and receive network information from their known peers. Client paths extend using a hop-by-hop approach; each intermediate hop selects at random several relays as options for the next hop. Although the client directly selects only the first hop in the path, the client also selects at random "witness nodes", which help in the selection of further relays proposed for the path. Although MorphMix includes a collusion detection system to prevent intermediate nodes from biasing relay selection, this detection system can be fooled by a malicious relay providing a set of relay options (for the next hop in the path) that were not selected uniformly at random, but instead specially crafted to circumvent the collusion detection system [TB06]. Thus, MorphMix is vulnerable to route capture attacks, as it cannot prevent against route selection bias from malicious intermediaries.

To solve the issues that arise from hop-by-hop designs that can lead to route capture attacks, Salsa [NW06] presents an anonymity network design that uses a DHT to publish network directory information. Furthermore, this DHT determines which nodes become the next hop in a client's path through the network. Relays are required to have knowledge of all other relays logically near them in the DHT, and limited knowledge of relays far from them. The requirement of using a DHT to facilitate relay lookups is intended to prevent malicious nodes from influencing the selection of relays for the client's path. To ensure a malicious node has not influenced path selection, each selection for the next hop is done by performing redundant lookups. More specifically, relays will look up information for r relays at each hop, as opposed to one single relay, and present this information to the client. The client then will select one relay from the r options to serve as the next hop. However, the approach of performing redundant recursive lookups for specific nodes results in an exploitable information leak [MB12], such that passive adversaries observing network traffic can infer the nodes in a specific circuit. As such, passive adversaries detecting redundant queries can perform epistemic attacks using observed network patterns.

ShadowWalker [MB09] uses a peer-to-peer random-walk protocol and a DHT to distribute relay information to other relays in the network. ShadowWalker requires circuits to be built using a hop-by-hop approach with additional verification by the client. 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 trivial routing attacks by requiring relays to commit to routes by distributing their local finger tables to deterministically chosen "shadow nodes", which the client then uses to verify the response for their selected index. Verification that a route is included in a node's routing table is done via Merkle proofs, in which the root hash is signed by the node and distributed to its shadow nodes to commit to a specific route. In spite of these techniques, ShadowWalker is still vulnerable to route capture attacks [SDH^{+}10] when an insufficient number of shadow peers are used, leading to an eclipse attack in which an attacker can compromise a "neighborhood" of nodes, resulting in malicious nodes becoming shadows for each other. Increasing the number of shadow peers protects against such eclipse attacks because at least one honest node is required to inform the client of the dishonest behaviour. However, increasing the number of shadow peers results in the strengthening of the ability for an attacker to launch a selective denial-of-service attack, which can result from a shadow node refusing to provide its signature for its known routes. Furthermore, the probability of epistemic attacks in ShadowWalker grows relative to the length of the path, as a network adversary can correlate each peer's view of the network with the path the route followed, increasing the probability of determining where the route originated from.

The Invisible Internet Project (I2P) is a peer-to-peer, fully decentralized anonymous overlay network [ZH11]. Like Tor, I2P implements source-based routing. However, instead of building long-lived circuits between peers through the network, I2P requires participants to build incoming and outgoing tunnels, which serve as entry points for sending and receiving messages. This design is similar to Tor's onion service protocol, which uses similarly designed introduction points for connecting clients and hidden services. I2P also uses *garlic routing* for message encryption between nodes in a path; unlike onion routing, in garlic routing, all messages are self contained and do not depend on relays persisting long-lived connection state. Furthermore, the nature of self-contained messages in garlic routing allows intermediate relays to bundle multiple messages within a single encrypted packet destined for their peer (similarly to how a garlic contains multiple cloves). Furthermore, unlike Tor, I2P does not rely on central authorities to distribute network directory information. Instead, untrusted "netfill" routers maintain a DHT that represents all network information, and gossip relay information updates to their known peers. Consequently, the integrity of gossip messages between nodes cannot be validated for its integrity or authenticity by clients or relays. As such, malicious relays can gossip false information to its peers or influence routes through the network, as the network directory information served by any netfill router cannot be validated [I2P10].

Cascade network designs require the anonymity network to be divided into stratified layers; routes through the cascade network subsequently contain one relay from each layer. Vuvuzela [vdHLZZ15]—a private messaging network that is designed for high scalability while protecting sensitive information such as message metadata—is one example of a cascade network. In Vuvuzela, users are pre-assigned fixed "dead drops", which are independently operated servers in the Vuvuzela network. Users exchange messages with other users by sending their messages to these dead drops, which the other user will then request their messages from. New dead drops are assigned to users upon each new "round", or network epoch. A trivial end-to-end linking attack by a network observer is prevented by the fact that the dead drop serves as this connection point for many sets of users. Consequently, users of a single dead drop have effectively as much anonymity as the number of other users assigned to this particular dead drop. Similar in design to Vuvuzela is the Java Anon Proxy (JAP) [Tea11], another example of a cascade mix network. However, in JAP, users themselves choose one of several publicly available sets of independently-run mix servers to use to exchange messages through. Consequently, users of cascade mix networks such as Vuvuzela and JAP enjoy as much privacy as provided within their specific partition of the network. In contrast, users of "free-routed" networks, where paths through the network can traverse through any network relay, make up a single anonymity set; to deanonymize these users, attackers must distinguish one user from all other users in the network as a whole, as opposed from all other users in a subset.

Dandelion [BVFV17] and Dandelion++ [FVB+18] present a lightweight gossip protocol for cryptocurrency networks to protect against *mass deanonymization* (as opposed to targeted attacks). Dandelion and Dandelion++ protect against a weak but highly connected adversary (such as a botnet) that can hold "supernode" status in the network by maintaining connections to the majority of all peers in the network. Supernodes of this kind can passively gather information about the network at large and correlate messages to the originating node that was used to publish the message (in the case of a cryptocurrency, each message is a transaction). While Dandelion only assumes adversaries who can corrupt up to some large fraction of the network but follow the protocol honestly, Dandelion++ considers a stronger adversarial model, such that adversaries are allowed the power to maintain arbitrarily many connections to other nodes; thereby acting outside the limited number of connections specified in the Bitcoin protocol. Both protocols follow a randomized design to determine if intermediate nodes should either forward traffic to a single neighbor (making up the "stem") or to broadcast the transaction to the entire network (creating the "fluff") . While this approach demonstrates privacy against adversaries seeking to perform mass deanonymization, neither protocol considers adversaries seeking to perform targeted attacks against specific nodes, transactions, or users. Furthermore, the protocol is performed without encrypting messages between peers, so does not protect against colluding nodes in the same stem. Finally, further evaluation is needed to assess how a highly asymmetric network impacts the effectiveness of the design, such as the case if a handful of nodes are responsible for generating an outsized number of transactions.

3.1.4 Comparison of Security Properties

Because Walking Onions should simultaneously provide security against the attacks described in Section 2.3, while improving the scalability of the network such that the knowledge the client requires remains constant even as the network grows, it is important to understand how each network design impacts its security.

We will now provide a high-level comparison of the anonymity network designs presented in the previous section and how these designs compare against the presented attacks. We summarize our findings in Table 3.1.

Description of classes of attacks

Before comparing specific attacks against the anonymity networks we reviewed in in Section 2.3, we will first present a summary of these attacks as background knowledge.

- *Epistemic* attacks, as further described in Section 2.3.1, occur when an attacker observing the network can gain information about a specific client due to relays the client selects to build paths with.
- *Route capture* attacks, as further described in Section 2.3.2, occur when one node can influence a client's path through the network without detection.

- *Limited anonymity set* does not treat all users of an anonymity network as a single set, and instead divides users using some partitioning scheme into smaller subgroups. Ideally, users are within the largest anonymity set as possible, as an attacker's advantage to distinguish a single user is only so large as their anonymity set. As such, an attacker must be able to distinguish one user's pattern of behaviour among all other users (as opposed to only users within their respective set).

- *Timing fingerprinting* attacks occur when an attacker observing the network can distinguish identifiable information about the user by characterizing timing of user-influenced behaviour. For example, certain protocols have a unique fingerprint for the timing of network packets exchanged between parties; an attacker observing timing of such network packets exchanged between relays could perform an end-to-end correlation using timing delay as a fingerprint.

- *Usage fingerprinting* attacks occur when a user's behaviour on the network can be distinguished from other users, such the pattern of information retrieval. For example, in a network where users can hold differing views of the network, a user's query for network information updates can leak a unique fingerprint.

- *Denial of Service* attacks refer to the ability for a network adversary to prevent a user from using the anonymity network. For example, if a user must relay their traffic through a single pre-determined node, and this node refuses to forward the user's traffic, this attack constitutes a denial-of-service attack.

Analysis

We now provide a summarized assessment of the security of the anonymity network designs described in Section 3.1.1, Section 3.1.2, and Section 3.1.3 against the attacks just described.

Crowds. Because of its completely decentralized nature and hop-by-hop routing, Crowds provides security to users only in the form of providing a single anonymity set and protection against a malicious node refusing to relay a user's traffic. Other attacks are trivially possible because of the amount of influence relays have over a user's path and the lack of protection against an adversary observing messages passing through the network.

Salsa. Similarly to Crowds, Salsa is completely decentralized and follows a hop-by-hop routing design. Salsa requires recursive lookups from intermediate nodes into a specific position on a DHT to provide options to clients for the next hop. However, a malicious intermediate node can still introduce bias to a client's path by positioning nodes strategically in the DHT, leading

Table 3.1: Comparison of security properties of anonymity networks and protocols
●=secure; ○=not secure;

		Path Based Attacks		Network Anonymity			Usability
		Route Capture	**Epistemic**	**Limited Anon. Set**	**Timing Fingerprint**	**Usage Fingerprint**	**Denial of Service**
Random-Walk Networks	Crowds	○	○	○	○	○	●
	Salsa	○	○	○	○	○	●
	ShadowWalker	○	○	○	○	○	●
	Dandelion++	○	○	○	○	○	○
Tor-based protocols	Current Tor	●	●	○	○	●	●
	Torsk	●	●	○	○	○	●
	PIR-Tor	●	●	○	○	●	●
	ConsenSGX	●	●	○	○	●	●
Mix networks	MorphMix	○	○	○	●	○	●
	Vuvuzela	●	●	●	●	●	○
	Katzenpost	●	●	○	●	●	●
Other	I2P	○	○	○	○	●	●
Walking Onions	Telescoping	●	●	○	○	●	●
	Single-Pass	●	●	○	○	●	●

to an eclipse attack. Furthermore, these recursive queries provide a fingerprintable information leak to network observers.

ShadowWalker. Improving upon the Salsa design, ShadowWalker also follows a decentralized and hop-by-hop design and uses a DHT for node lookups. While nodes gossip their routing tables to other nodes, and the client can verify their path using lookups to other randomly chosen relays, eclipse attacks remain possible by strategic positioning on behalf of the adversary. Increasing the number of nodes to verify routes increases the capability for a denial of service attack. Finally, as nodes have only a partial view of the network, an adversary observing the network can distinguish routes to eventually perform end-to-end correlation by observing which relays have been selected for a path.

Dandelion++. Offering a security model that encompasses only passive adversaries, Dandelion++ protects against peer-to-peer gossip protocols in which one node can eventually connect to all other nodes in the network and observe message flow to perform end-to-end correlation attacks. Dandelion++ does not protect against any attacks stronger than this model, and thus any form of node collusion or injecting packet delay or watermarking can easily result in route capture and fingerprinting attacks. Because a message is distributed along a single "stem" and no verification mechanism exists to ensure that a message has been distributed to the rest of the network, a relay in the stem can drop the message without detection.

Current Tor. Because of its reliance on a globally consistent network view that is authenticated by trusted authorities and source-routed path selection design, Current Tor protects against epistemic and route capture attacks. However, because Tor is a low-latency network, an adversary that can observe two ends of a client's circuit can perform timing-based fingerprinting attacks [BMG$^+$07]; due to the difficulty of preventing such attacks while maintaining the low-latency properties of the network, Tor explicitly does not protect against such attacks [Din09]. Because Tor is a free-routed network, if a relay fails to forward a user's traffic, the user can simply build a new circuit using a set of different relays.

Torsk. To improve the scalability of Tor, the design of Torsk moves away from a centralized authority approach to one where network information is distributed by a DHT. However, each circuit extension operation requires multiple DHT lookups. Furthermore, similar to other DHT-based designs, malicious nodes can strategically position themselves in the DHT and thus perform route capture attacks.

PIR-Tor. Using a PIR-based design, PIR-Tor seeks to improve the scalability of Tor by requiring clients to look up one subset of relays via a PIR query for each circuit built. PIR-Tor maintains Tor's existing threat model but introduces complexity for the IT-PIR design and computational overhead for the C-PIR design.

ConsenSGX. Using a PIR design but with trusted hardware to ensure protection against an honest-but-curious server, ConsenSGX remains within Tor's existing threat model assuming the trusted execution environment can be trusted. However, this assumption is undesirable in practice, as trusted execution environments have been shown to have a number of vulnerabilities [VBMW$^+$18,RMR$^+$21,MOG$^+$20] and would require upgrading many Tor relays to hardware that supports them.

MorphMix. Utilizing both a mix network and onion-routing design, MorphMix is a peer-to-peer network in which circuits are built hop-by-hop. Malicious nodes can circumvent the collusion detection system and thus can perform route capture attacks. Furthermore, as nodes only have a partial view of the network, network adversaries can perform epistemic attacks by passively observing network traffic.

Vuvuzela. Following a cascade mix network design, Vuvuzela provides higher scalability while limiting the anonymity set of its users to a subset of of the complete userbase. The smaller the network, or the larger the number of subsets, the smaller a user's anonymity set becomes. Because each user is assigned one dead drop per network epoch, a user could experience a denial of service when their assigned dead drop fails to perform forwarding services; in this case, the user would have to wait until they are assigned a new dead drop in the next epoch.

Katzenpost. Implementing the Loopix mix net design, Katzenpost presents a centralized mix network design, relying on a set of network authorities to authenticate the network directory document for distribution to all other network participants. By both providing verifiability of relay information and timing and cover traffic to protect message patterns en route, Katzenpost provides a stronger security model than low-latency networks. Because users can make connections on demand with any relay in the network, if a node refuses to forward user traffic, a user can simply attempt to send their traffic through another node. Further, as the network is globally consistent, the user's view is the same as all others' view.

I2P. Because I2P is completely decentralized and does not require a single global view among all of its users, users do not have strong assurance about the integrity of relay information that they receive. As such, malicious relays can influence a user's view of the network and inject malicious relay information into the network. However, because the network is free-routed, if one relay refuses to service a user's request to forward traffic or send network information, the user can simply query other network relays (assuming the user's chosen introduction point into the network is reliable).

Walking Onions. While we describe further how Telescoping and Single-Pass Walking Onions compares to current Tor in Section 6.4, we include Walking Onions here to demonstrate how it compares to other anonymity network designs in the literature. Notably, Walking Onions maintains the same security properties as current Tor. What is not shown in Table 3.1 is how Single-Pass moves to "windowed" forward secrecy for the choice of relays in a user's path; we further expand on this tradeoff in Section 6.4.

3.2 Designs For Single-Pass Circuit Creation

As discussed in Section 2.2.3, Tor currently uses a telescoping design for circuit extension; each circuit extension requires the client to perform one round trip key exchange with the relay in the last position of the circuit. For a client with a slow connection to the first hop in the path, the telescoping design to extend a circuit results in additional latency even before the client can begin downloading data from the intended destination.

The original Onion Routing construction proposed a single-pass circuit construction technique [RSG98]. However, in this design, the client uses the relay's long-lived public key to derive their shared secret after generating an ephemeral key pair for their half. The client then sends the relay their ephemeral public key to finalize the handshake, and does the same for each hop on the path. However, while this design presents a single-pass construction, this approach lacks *forward secrecy*; any adversary that compromises the relay's long-lived secret key can decrypt past communication with this relay, so long as the adversary has also recorded incoming and outgoing network traffic with this relay. For this reason, the Tor protocol today utilizes a telescoping approach to circuit establishment [DM20] which requires relays to provide a fresh ephemeral key for each circuit establishment, ensuring forward secrecy for client communication at the expense of latency for the client (as further described in Section 2.2.3).

While more recent work has presented single-pass designs for anonymity networks such as Tor, these designs have undesirable tradeoffs in either security or scalability factors. Below we highlight past work in onion-routing designs that present a single-pass circuit establishment design. We furthermore assess the security properties of each design and their tradeoffs between efficiency, scalability, and security. We maintain these considerations when designing Walking Onions and how our designs preserve scalability and security while enabling a single-pass circuit construction protocol.

3.2.1 Early Single-Pass Designs

Øverlier and Syverson [ØS07] 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, and only provides eventual forward secrecy for client communication.

3.2.2 Single-Pass Designs Via Identity-Based Encryption

Kate et al. [KZG07] propose a pairing-based single-pass onion routing scheme using identity-based encryption, in which the session key between a client and a relay is negotiated using an ephemeral key selected by the client and the relay's identity, resulting in a non-interactive key agreement protocol. However, the drawback to this protocol is twofold. First, a trusted central authority is required to distribute identity keys to all relays for each epoch, resulting in a single point of trust that, if compromised, would result in a total compromise of the network. Second, the protocol does not ensure forward secrecy; compromising a node's private key leads to the compromise of all communication with this node.

Catalano et al. [CFG09] present a hybrid design utilizing both public key and identity-based cryptography to limit the power of the central authority. However, both approaches require a central trusted authority for key distribution, which is both a performance bottleneck and a further centralization of authority. Furthermore, these schemes also lack forward secrecy for client communication, as the node's identity key is used to negotiate the shared session key with the client.

Catalano et al. [CDRF$^+$13] present a non-interactive single-pass identity-based variant, in which the private identity keys of relays are ratcheted forward without requiring public key rotation, thereby allowing forward secrecy without periodic interaction with a central authority. However, compromise of a secret identity key leads to the total compromise of all future private keys and consequently client traffic. Furthermore, while ongoing key rotation is non-interactive and does not require communication with a central PKI, nodes bootstrapping to the network must still request their initial credential from a trusted central credential distributor. However, such a design is undesirable for anonymity networks seeking to decentralize trust as well as limit performance bottlenecks.

3.2.3 Single-Pass Designs Via Sphinx

In addition to the aforementioned tradeoffs, the above designs also do not defend against adversaries performing bitwise linking attacks using observed public key material exposed in transit. Any adversary observing the network could correlate incoming and outgoing packets simply by observing the client's ephemeral public key and each relay's long-lived public key during key negotiation.

Sphinx [DG09] presents a packet format for anonymity networks, providing bitwise unlinkability for public key material that is exposed in incoming and outgoing network packets during the key-establishment stage for circuit construction. The session key for each node on a circuit constructed using the Sphinx packet format is computed from the server's private key and a blinded element initially provided by the client and re-blinded at each hop, serving as the public key for the next hop and thus ensuring unlinkability of public key material between hops. As each intermediate relay contributes to this blinding factor before sending it to the next hop, *only* the client and this relay can derive the corresponding session key. Notably, protocols using the Sphinx packet format do not require intermediaries to interact with the client in order to derive blinding keys for the next hop in the path. As such, onion routing protocols can capitalize on this capability to build single-pass variants.

In its original work, the design of Sphinx included a proof of secure routing using a proof strategy first developed by Camenisch and Lysyanskaya [CL05], by proving the Sphinx design as indistinguishable to an adversary from an idealized onion-routing protocol. However, a more recent work by Kuhn, Beck, and Strufe [KBS20] demonstrate flaws in this proof strategy, thereby demonstrating that the idealized definition of onion routing [CL05] lacks or incorrectly defines important security properties. As such, the authors propose additional security properties and prove that a simplified variant of Sphinx satisfies their proposed security notions.

Kate and Goldberg [KG10] assess the security and efficiency of a range of onion-routing schemes presented in the literature when Sphinx is used as the underlying packet format. The schemes evaluated in this assessment include Tor-preDH, the half-certified DH agreement using a relay's long-lived public key [ØS07], pairing-based onion routing [KZG07], and Certificateless Onion Routing [CFG09]. Unfortunately, undesired tradeoffs remain with each construction. For example, each scheme at best provides for eventual forward secrecy for client communication. In the case of Tor-preDH, each key rotation epoch requires every relay on the network to send a fresh keypair to the directory authorities, creating a heavy burden on the network for each key rotation period. Furthermore, the use of Sphinx as the underlying packet format does not remove the requirement for a centralized system to generate and assign private keys—as is required by identity-based encryption schemes—and consequently retains undesirable tradeoffs.

While Sphinx provides a promising building block to constructing a single-pass circuit construction protocol for an onion-routed network, the lack of forward secrecy for client communication in prior designs presents an undesirable tradeoff between efficiency and security. We will construct Walking Onions to improve upon past designs to achieve the best of both worlds, allowing circuits to be built with lower latency than in telescoping designs while while preserving forward secrecy of client communication.

3.3 Summary

In seeking to provide a more efficient mechanism for users to select paths through an anonymity network, Walking Onions will also need to protect against path-routing and epistemic attacks by ensuring users' selection of relays is indistinguishable to a network adversary, as well as ensuring that users can verify the integrity of their selected paths. Furthermore, Walking Onions will need to maintain forward secrecy of client communication during circuit establishment even in a single-pass setting. Moreover, the Walking Onions design should not introduce additional centralization into the network, as additional network centralization poses both a security and performance risk.

Even while preserving the above-mentioned security properties, Walking Onions will need to do so while improving the scalability of the network to minimize the amount of network information clients are required to maintain even as new relays join the network. The benefit to improving the scalability of anonymity networks is twofold: users of these networks both enjoy a larger anonymity set, while also providing the opportunity for more users to make use of these networks and consequently improve their privacy online. As such, scalability is a critical feature that should be preserved even while maintaining security against known network attacks.

Next, we will present the design of Walking Onions, discuss the threat model and goals of the protocol, and demonstrate how Walking Onions improves over prior related work in order to achieve a scalable design without degrading security for users of the anonymity network.

Chapter 4

Walking Onions Overview

To address the scalability of Tor and similar anonymity networks, we present Walking Onions, a set of novel protocols and algorithms to allow clients to obliviously select relays for paths through the network and establish secure circuits with these relays. Our design protects against route capture and epistemic attacks, and does not require clients to maintain information about the complete state of the network.

In this chapter, we discuss the intended threat model of Walking Onions, present scalability and security goals for the protocol, and review key insights to our work. In Section 4.1, we present the threat model for Tor and subsequently for our work as well, as well as discuss the concept of a network authority and how such an authority can be instantiated in practice. In Section 4.2, we discuss the scalability and security goals for Walking Onions, against which we evaluate our proposed protocols later in this work. Finally, we review key insights that influenced the design of Walking Onions in Section 4.3.

4.1 Threat Model

We assume the threat model of the Tor network [DMS04]; 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. As such, a malicious relay is not bound to operate under any certain protocol and can behave arbitrarily. Furthermore, the threat model includes adversaries who can observe only a *subset* of network traffic. Outside of this threat model are 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 [Tor20a]. 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.

4.2 Goals

4.2.1 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 show a relaxation of this requirement to allow for logarithmic growth for clients, in order to improve bandwidth for relays (see our performance evaluation in Chapter 8). 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.

4.2.2 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.

More 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 [DMS04] cannot gain useful information about the user from observing traffic. By a limited network view, we assume that the adversary cannot simultaneously observe connections for a single user *both* entering and exiting the network, as otherwise the adversary could perform end-to-end traffic correlation. 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.

4.3 Key Insights

Walking Onions uses the following key 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 [GSU12]—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.

4.4 Summary

In this chapter, we present an overview of factors important to understanding the design and intent of Walking Onions. We review the threat model of Walking Onions, which corresponds to Tor's current threat model as well. Further, we discuss the assumption that the anonymity network implementing Walking Onions maintains an authority acting as a root of trust for the network. We present both scalability goals and security goals that we later use to evaluate Walking Onions path-selection and circuit establishment protocols. Finally, we present key insights that are important to the overall design of Walking Onions. Notably, these insights tell us that so long as clients can verify the identity of relays selected for their circuit even after the circuit has been built—and be assured of the randomness of their relay selection—then the resulting circuits preserve similar security properties as existing circuit-establishment designs.

Chapter 5

Distributing Network Information in Walking Onions

In this chapter, we present efficient encoding and distribution mechanisms for network directory information in anonymity networks implementing Walking Onions.

We begin by outlining notation and terminology used to define our work in Section 5.1. We then introduce how network directory documents are encoded and distributed in Walking Onions in Section 5.2. We conclude by introducing in Section 5.3 several mechanisms for efficiently authenticating network directory information by the network authority.

5.1 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*.[1] 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.

[1]For comparison, the length of an epoch in Tor is one hour.

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.

5.2 Encoding Network Directory Documents

In the Walking Onions design, once every epoch, the network 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.

5.2.1 ENDIVEs

Importantly, in Walking Onions, *only relays* need to download ENDIVEs. If the anonymity network requires all participants to download a constant-sized network parameters document, clients are required to download only this document once per network epoch in order to build circuits. Relays are required to fetch the entire ENDIVE at bootstrap; afterwards, relays fetch only updated SNIPs once per epoch.

Each ENDIVE contains the set of all valid SNIPs for the epoch and an authentication tag over this set. We describe options to generate this authentication tag in Section 5.3.

Alternative topologies. For simplicity, in this work we assume a full clique network topology; that is, that any 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 [ADD$^+$17], which relies upon a stratified topology, in which relays are partitioned into distinct layers and client paths contain one relay per layer.

5.2.2 SNIPs

In anonymity networks, a *relay entry* represents 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 properties such as supported features or versions. As described in Section 2.2, networks such as Tor distribute a network directory document that 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 extra fields in addition to the fields included in a relay entry.

First, each SNIP includes an *index range*: a range of integer values whose size is proportional to the probability of selecting this relay. When generating the ENDIVE for the next network epoch, the network authority computes and assigns index ranges for each SNIP included in the ENDIVE. As we describe further in Chapter 6, 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 network authority over *only* the content in the SNIP (we describe several options to perform this authentication in Section 5.3). Third, each SNIP includes two timestamps indicating the epochs when the SNIP was created and when the SNIP expires.

Because SNIPs are individually authenticated and include these additional fields, clients can validate SNIPs independently without downloading the entire ENDIVE. We describe how clients build upon this capability to security perform path selection and circuit extension in Chapter 6.

Weighted Relay Selection

Often, selection of relays occurs over a distribution weighted by some relay property, such as how much bandwidth the relay provides to the network. We address this and a range of additional complex path selection cases in Chapter 7, but note here one mechanism to select relays based on their fulfillment of a single property, such as the relay's available bandwidth, is to assign each relay's index range using a *weighted probability distribution*. In doing so, the probability that the relay will be chosen by the client depends on its respective weight.

We next describe an example algorithm in more detail that the network authority can perform in order to assign index ranges to each SNIP to be included in an ENDIVE.

Computing Index Ranges

Assume a set of relays R and a desired probability distribution P over R. First, discard all relays $R_i \in R$ for which $P(\cdot) = 0$. Let N denote the remaining relays where $P(R_i) \neq 0$. Next, calculate the cumulative distribution function C over N such that $C_0 = 0$ and $C_i = \sum_1^i P(R_i)$.

C_i can be transformed into a discrete representation by computing ${C'}_i = \lfloor C_i \cdot \alpha \rfloor$. Each relay R_i is assigned the index range $[C'_{i-1}, C'_i - 1]$; the endpoints of this range are placed in relay R_i's SNIP.

5.3 Authenticating ENDIVEs and SNIPs

To facilitate the ability for clients to verify SNIPs during circuit construction without requiring the complete ENDIVE, each SNIP has an authentication tag produced by the network authority over the information *only* within the SNIP. We now survey several authentication mechanisms first introduced in Sections 2.5 and 2.6, and evaluate the performance of each in Section 8.4.2.

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 [BGLS03b, BDN18, MPSW19] provide an n-out-of-n scheme in which all voters must participate to produce a joint signature, while threshold signatures [BLS04] 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 Proofs. Another authentication approach includes Merkle proofs [Mer88] to ensure a specific SNIP is within the ENDIVE signed by the authority. A client can download the Merkle root hash for the most-recent ENDIVE in the (constant-sized) network parameters document that itself is authenticated by the authority. During circuit construction, the client receives a Merkle 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 proofs 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 proof, 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 proofs, 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 8.2.1.

Authenticating ENDIVEs. Because ENDIVEs are simply the set of SNIPs valid for the current epoch, a relay can validate the integrity of an ENDIVE by verifying the authentication tag for each SNIP in the ENDIVE, and ensuring that every SNIP has been included in the ENDIVE. However, a more efficient mechanism is to include an additional authentication tag over the ENDIVE as a whole, for relays to validate 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.

5.4 Summary

In this chapter, we begin by introducing important notation and terminology used in our work. Following, we present encoding techniques for network directory documents to enable Walking Onions path-selection and circuit extension protocols. After presenting the concept of ENDIVEs and SNIPs, we discuss how to generate and distribute ENDIVEs such that each relay maintains a copy of the most up-to-date ENDIVE. Furthermore, we discuss how Walking Onions can be used in alternative network topologies other than a fully connected network. We conclude with a review of several authentication mechanisms for ENDIVEs and SNIPs using the network authority as the root of trust, such that each SNIP can be validated independently of the ENDIVE as a whole.

Next, we examine how Walking Onions can use ENDIVEs and SNIPs to construct efficient path-selection and circuit-establishment protocols through an onion-routed anonymity network.

Chapter 6

Walking Onions Path Selection and Circuit Extension

In this chapter, we introduce additional preliminaries to our work, and then present Walking Onions path-selection and circuit-establishment protocols. We conclude by assessing the performance and security tradeoffs of these protocols against Vanilla Onion Routing, a generalized onion-routing protocol that we use for comparison and evaluation purposes.

We begin by introducing notation and terminology in Section 6.1. We then present two protocol variants of Walking Onions, which provide clients with the ability to obliviously yet verifiably select relays and extend circuits through anonymity networks. We call the first variant *Telescoping Walking Onions*, and present this work in Section 6.2. We call the second variant *Single-Pass Walking Onions*, and present it in Section 6.3. We discuss the performance and security tradeoffs of each protocol relative to Vanilla Onion Routing in Section 6.4. Finally, we present a hybrid protocol between Telescoping and Single-Pass Walking Onions in Section 6.5 and mechanisms to bootstrap the first connection in any of the Walking Onions protocols in Section 6.6.

6.1 Preliminaries

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

6.1.1 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 secure mechanisms to establish the first hop in a circuit in Section 6.6.

6.1.2 Authenticated key exchange

We assume the existence of a one-way-authenticated two-party key exchange with post-specified peer, in which the initiator authenticates the responder after both supply ephemeral keys. We follow a similar approach to Canetti and Krawczyk [CK02] 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.

$ComputeSecretAndValidate(x, g^y, g^b, A) \rightarrow (S, \{0, 1\})$: Performed by the client during a client/relay handshake during circuit establishment. 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.

6.1.3 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 [GRPV19]. We require the following VRF operations:

KeyGen$_{VRF}(1^{\lambda}) \rightarrow (c, g^c)$: 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.

6.1.4 Vanilla Onion Routing

We describe our protocols with reference to a generic Tor-like onion-routing protocol. We call it *Vanilla Onion Routing*, and include a detailed protocol definition in Figure 6.1.4.

We use Vanilla Onion Routing as a generalized onion routing protocol upon which to build or modify for Walking Onions path selection and circuit extension protocols. In doing so, we can more easily identify where Walking Onions remains the same or departs from general onion routing protocols.

As in our next sections, we identify stages in the protocol as referring to either key exchange or path extension operations. K denotes a key exchange operation, and P denotes a path extension operation. Path extension and circuit establishment are distinct operations but performed jointly (e.g, in the same protocol stage).

6.2 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 5.2. 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 unqiue 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 5.3). 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.

Let (b_n, g^{b_n}) denote the long-term key for relay R_n.

When the client extends an existing circuit:

1. 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 (R_{n+1}, 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 $(y, g^y) \leftarrow KeyGen_{Auth}(1^\lambda)$; 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.

Figure 6.1: Vanilla Onion Routing 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 Section 6.1.4) 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 $(y, g^y) \leftarrow KeyGen_{Auth}(1^\lambda)$; 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 4.2 as clients do not need to maintain the complete network directory document. It also maintains the same latency overhead for clients during circuit construction as Vanilla Onion Routing, as the network traffic pattern is the same.

6.2.1 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 2.3, 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 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 [BDMT07], this behaviour is no worse than existing onion-routed networks, and the client can simply build another circuit using freshly selected relays.

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}.

6.3 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 (a generalized protocol presented in Section 6.1.4). We now present *Single-Pass Walking Onions*, a path-selection and circuit establishment protocol that enables the same scalability benefits as Telescoping Walking Onions, but requires only a linear number of total messages relative to the path length. Further, the client only sends one and receives one message when building a new circuit, which is of particular benefit for clients with high-latency connections. The key insight to Single-Pass Walking Onions is this: if the client can be assured that i was selected uniformly at random, 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 selectively influenced by any intermediary.

Building upon this insight, we will now 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 aforementioned, 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 calculates $(i, \pi) = Prove(c_j, D^{b_j})$—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 [DG09] 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 further assess security properties of Single-Pass Walking Onions in Section 6.3.1, 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 [DG09], 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.

Scalability Goals. As with Telescoping Walking Onions, Single-Pass Walking Onions fulfills the scalability goals described in Section 4.2 as clients do not require the complete network directory document. Furthermore, in Single-Pass Walking Onions, clients experience only a single round trip, which can be significant if the client has a high-latency connection.

Description of protocol. We first describe additional several 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 favourable (for example, colluding) relays, we require each relay to publish their path-selection public key in their SNIP, consequently binding the relay to this key for as long as the SNIP is valid. If a relay is compromised and an adversary learns its private path-selection key, we ensure eventual forward secrecy for clients' path selection by requiring path-selection keys to be rotated periodically. Because fresh key are generated for each key rotation, compromise of a past path-selection key will *not* impact future keys. Such a "windowing" approach to forward secrecy is well established for privacy protocols in practice [PM16, UG15], 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 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 adversary has less advantage from productively using this information as the network size and number of participating clients grows.

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 6.6).

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 jth 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 $(d, g^d) \leftarrow KeyGen_{Auth}(1^\lambda)$. 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 SNIP's authentication tag 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}.

6.3.1 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^x 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, while a malicious relay can perform a denial-of-service attack by refusing to select the next relay in the path [BDMT07], this is no worse than in existing onion routing schemes where on-path relays refuse client connections; in these cases, the client can simply build a new circuit with other relays.

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).

6.3.2 Performance Optimizations for Single-Pass

Key Binding Using Signatures

While Definition 2 uses a VRF to bind a relay's path selection key g^{c_j} to its long-term key g^{b_j}, other techniques exist that can achieve the same security goal. We now describe an alternative approach and assess the performance tradeoffs.

Most straightforwardly, at the time each relay uploads its information to the network authority to be included in the ENDIVE for the next network epoch, this relay can also include signatures for *both* its long-lived key and path-selection keys. Doing so proves to the network authority that the relay who has knowledge of the private key for the long-lived key also has knowledge of the private key for the path-selection key, avoiding key-misbinding attacks in the future. Depending on the desired level of trust in the network authority, these signatures can be included in the relay's SNIP for the client to verify during circuit-construction time, or simply verified by the network authority before publishing the ENDIVE (but not directly included). During circuit establishment time, expensive VRF computations can thus be avoided, as the index into the ENDIVE can be derived by calculating a shared Diffie-Hellman secret using the client's ephemeral path-selection key d, g^d and the relay's path-selection key c_j, g^{c_j}.

This shared key (derived via the client's ephemeral path-selection key and the relay's semi-ephemeral path-selection key) could also be used as a key for a Pseudo-Random function (PRF). However, the input into the PRF would need to be deterministic so that both the client and the relay can derive the same output, and also not influenceable by the relay. One option is using a hash of the client's message as input into the PRF, for example.

The tradeoff between using VRFs at circuit-build time or signatures over the path-selection key during compilation of the ENDIVe for the next network epoch is between trust in the authority and bandwidth and computational cost. The least costly option for overall performance is the option that also requires the most trust in the network authority, by requiring the authority verify signatures without publishing these signatures in the relay's SNIP. However, a misbehaving authority could fail to correctly perform this verification step without detection. Using VRFs for key binding during circuit-build time requires additional computation for generating and verifying proofs, while including signatures in the SNIP (to be verified by clients at circuit build time) requires additional bandwidth when transmitting the SNIP.

Table 6.1: 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	◐	○	◐

Batching Mechanisms for Path-Selection Keys

Note that the frequency at which a relay updates information in its SNIP impacts how often the rest of the network is required to sync these updates; more frequent updates result in higher bandwidth usage. As such, it is important to limit relay information churn where possible.

With this in mind, we note an optimization for Single-Pass Walking Onions to slow how often relays update their path-selection keys in their SNIP while still rotating these keys once per epoch. Instead of naively updating each relay's SNIP every epoch with a new path-selection key, relays can publish n path-selection keys to use for the next n epochs. Consequently, this batching mechanism ensures the relay need to only update its SNIP every n network epochs to include new key material, though it must still locally securely delete outdated private key material from each prior epoch.

6.4 Tradeoffs Between Protocols

We now discuss the performance and security tradeoffs between Telescoping and Single-Pass Walking Onions relative to the path-selection and circuit-construction protocols used by Tor (further described in Section 2.2). We summarize these tradeoffs in Table 6.1.

6.4.1 Performance Tradeoffs

We provide an in-depth analysis of the performance of the Walking Onions protocols in Chapter 8, 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.

6.4.2 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 Chapter 7 for more information on accommodating complex path requirements). 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 past 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 [KZG07, CFG09, ØS07] (as further described in Section 3.2) 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.

6.5 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 Chapter 7 for more details on selecting relays weighted by some specific property in Walking Onions). 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-Path Walking Onions only to 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.

6.6 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 relays for this first position for up to four months at a time to prevent enumeration attacks [Tor20b, EBA+12].

6.6.1 Building from trusted relays

Many anonymity networks, such as Tor, preconfigure a list of stable relays in the client software as a bootstrapping mechanism [Mat20a,DM20]. These relays can be used to bootstrap in Walking Onions as well. To bootstrap, clients can build a circuit using one of these pre-configured relays as the first hop. After the circuit is complete, clients can extend this circuit to an additional relay that is suitable to serve as the first hop for future circuits (we discuss how clients can ensure the suitability of relays based on certain criteria in Chapter 7). The client can then throw away the circuit, building fresh circuits through this new first hop. This mechanism relies on the same security assumptions as when sending traffic through multi-hop circuits; that is, the cost to perform an end-to-end correlation between the client and the end destination is sufficiently high to an attacker.

6.6.2 Private Information Retrieval

Using PIR for client bootstrapping in anonymity networks is well established in the literature (as further described in Section 3.1.2), such as using a set of PIR servers to serve a subset of relay information to Tor clients [MOT+11, SG19]. 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.

6.7 Summary

In this chapter, we introduce two Walking Onions path-selection and circuit construction protocols. *Telescoping Walking Onions* gives clients full control over selection of a random index that the client uses to indicate the relay that their circuit should extend to, thereby building circuits iteratively. *Single-Pass Walking Onions* allows intermediate relays to randomly select the next hop in the path in such a way that the client can verify the randomness of this selection, and consequently builds the circuit in a single pass. We compare the tradeoffs of these protocols to a generalized onion-routing protocol and discuss how each protocol compares in terms of forward secrecy, client control, and efficiency. We then introduce a hybrid variant between Telescoping and Single-Pass Walking Onions and discuss settings when this variant may be a good option to provide a measure of both client control and improved efficiency. We present several bootstrapping mechanisms for the client to establish a secure connection to the first relay in their path, as Walking Onions assumes a separate bootstrapping mechanism to learn information for the first relay in their path.

Next, we examine how to build upon these protocols to allow clients more control over selecting relays for their paths that require relays with more particular characteristics.

Chapter 7

Complex Path Requirements

Until this point, we have presented path-selection protocols that have not specified how to generate the distribution over which the client selects relays from the most recent ENDIVE. However, clients often have more complex path requirements. For example, networks such as Tor weight client's selection of relays by bandwidth [Tor20d]. Further, many networks restrict which relays can be selected for specific positions in a client's path, such as Loopix-based networks, which use a stratified topology. We now present several mechanisms to accommodate client path requirements in Walking Onions, and discuss settings in which each mechanism is more or less suitable.

In Section 7.1, we discuss the simplest approach that is appropriate when most relays are expected to fulfill some property. In Section 7.2, we discuss how to encode multiple indices into a SNIP to facilitate client selection over multiple properties. In Section 7.3, we introduce a simple mechanism to group properties to limit the total number of selection choices. Finally, in Section 7.4, we present an alternative encoding approach in which a relay's support of a set of properties can be encoded into a Merkle tree, and in Section 7.5, we introduce a simple protocol such that one relay can verifiably recommend a better-suited relay to a client's path requirements.

7.1 Optimistic Attempt and Discard

A simple approach that is acceptable when most relays support a given property is the *attempt-and-discard* approach: the client builds a circuit randomly, and discards it if it is not suitable. For an example, if a client wants a circuit whose final relay supports a common property, such as forwarding traffic to port 80 (http), they can simply construct circuits optimistically, discarding those ending with relays that do not fulfill this property. However, if a client wants a less well-supported port, such as port 25 (smtp), attempt-and-discard would be costly.

Note that this strategy could in theory be fingerprintable by an adversary watching the network, as the number of attempts could leak the (weighted) prevalence of the client's path requirement. However, in practice, the advantage for an adversary to deduce this information can be bounded by ensuring that this strategy is used only for well-supported properties.

7.2 Separate index ranges

As discussed in Section 5.2, every SNIP is assigned a default index range by the network authority. However, SNIPs may contain multiple index ranges, with each index range corresponding to a separate (non-uniform) probability distribution that represents the relay's fulfilment of some property, such as a relay's available bandwidth in the case of Tor. Such a weighted ranking can be encoded into the index range assigned to each relay by the authority when generating the ENDIVE for the next epoch. When the client samples an index at random, the relays with a higher weight will subsequently be more likely to be selected. If a relay does not fulfill a property, its corresponding index range is empty. Note that representing properties by separate index ranges requires the client to not only send an index i during circuit construction, but additionally an identifier indicating *which* index range to use for relay selection.

One application of this approach includes the case of Tor, where relays can be segmented into different "roles", or positions on the path. With Walking Onions, properties can be represented as a separate index range within a SNIP. We demonstrate this mechanism in an instantiation of an example ENDIVE in Figure 7.1; each relay role is assigned a separate index range depending on whether the relay fulfills a particular role. Specifically, in Tor, relays can be assigned roles to serve as entry positions in a circuit, or volunteer to serve in the exit position; Figure 7.1 demonstrates these roles encoded into a SNIP. Further, the relay has a separate index range specifically to determine this relay's available bandwidth relative to the rest of the network.

To perform this selection, the client sends (ψ, i) to the last relay in the circuit R_n, where ψ represents the client's desired property and consequently indicating the index range that R_n should use when selecting the next relay in the circuit R_{n+1}.

This example ENDIVE has separate index ranges to support default indexing as well as per-property ranges for exit or entry roles. For this example, $\alpha = 1024$.

The ENDIVE data structure is boxed to represent how the timestamp and signature are properties of the ENDIVE, even though each SNIP in the ENDIVE also has its own timestamp and signature.

Timestamp, Signature

ID	Bandwidth	Exit?	Entry?	Keys, etc	Timestamp	Signature
R_1	0–127	0–255	0–511	...	...	...
R_2	128–383	256–767	$\varnothing$	...	...	...
R_3	384–895	$\varnothing$	$\varnothing$	...	...	...
R_4	896–1024	768–1024	512–1024	...	...	...

Figure 7.1: Example ENDIVE

7.3 Grouping by Class

When the number of properties grows, however, representing each property as its own index range results 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, a clearly inefficient approach. In this case, we can reduce the number of properties by grouping exit ports into *port classes*. Two ports are in the same class if every relay in the ENDIVE either allows both ports or denies both ports. 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.

7.4 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 proof, logarithmic in the number of properties.

7.5 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}. While this requirement could be supported by building and discarding circuits until the user's path requirements are met, such an approach is costly, especially when a required property is not widely supported. We now discuss how to handle this case using a technique we call *Delegated Verifiable Selection*.

Delegated Verifiable Selection uses an optimistic technique where a client attempts to build a circuit to a relay R_n in the hopes that R_n fulfills some property ψ. However, if R_n does not fulfill this desired property, R_n can *recommend* another relay that does fulfill ψ in such a way that the client to remain oblivious to the selection process but can later verify that the recommended relays does support ψ and that the selection process was not biased.

After extending a circuit to relay R_n, if the client requires R_n to fulfill a specific property ψ, the client sends ψ to R_n through the circuit (thereby ensuring no intermediate relay can learn ψ). If R_n supports ψ, the circuit extension to R_n is considered complete and usable. Otherwise, R_n computes an index i^* (taken $\text{mod}\alpha$) derived from the hash of the client's messages to R_n so far in the protocol. Relay R_n then selects the SNIP Σ whose index range for property ψ contains

i^*. R_n 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^{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 requires either Telescoping or Hybrid Walking Onions to give the client complete control over specifying the extension to the relay corresponding to their desired index i'.

7.6 Summary

In this chapter, we present several techniques and protocols to facilitate Walking Onions path selection and circuit extension protocols for more complex complex path requirements. We first discuss the Attempt and Discard approach as a simple technique which—in the case that the probability of selecting appropriate relays is high—requires clients to simply discard circuits that do not meet the client's path requirements. However, in settings where the probability of selecting an appropriate relay is not as high, we discuss the technique of maintaining separate index ranges in each SNIP. However, as this approach results in linear growth in the size of the SNIP, we conclude this chapter by describing a technique using Merkle trees to encode a relay's support for a set of properties, as well as a protocol which one relay can verifiably recommend to the client another relay better suited to the client's path requirements.

Next, we evaluate the performance of Walking Onions protocols.

Chapter 8

Performance Results

We now compare the performance of Walking Onions protocols to that of an optimized version of Tor that uses Vanilla Onion Routing as described in Section 6.1.4; we refer to this variant of Tor as *Idealized Tor*. We compare Walking Onions and Idealized Tor on their bandwidth, CPU, and latency cost to both relays and clients.

We summarize the notation used for our evaluation in Table 8.1. In Section 8.1, we present Idealized Tor and its corresponding parameter choices. In Section 8.2, we compare the bandwidth cost between Idealized Tor and Telescoping and Single-Pass Walking Onions. In Section 8.3, we assess the latency cost of each protocol, and in Section 8.4 we derive CPU costs of each protocol for both ENDIVE generation and validation as well as circuit construction overhead. In Section 8.5 we compare Walking Onions designs to PIR-based designs with similar scalability goals for anonymity networks.

All of the code used to validate the derived performance results in this thesis is available at https://git.uwaterloo.ca/ckomlo/walking-onions-measurements.git.

8.1 Idealized Tor

Idealized Tor is a generalized and *strictly* more efficient design modeled after the real Tor network; we leverage a generalized model to provide a useful frame of reference for a larger class of anonymity networks that utilize onion routing protocols. By constructing this deliberately underestimated model, we ensure that any protocol that performs better than Idealized Tor will perform better than onion routing networks in practice.

Idealized Tor requires all participants download relay information once per epoch as such information changes, and bootstrapping requires all clients and relays to download information about all relays in the network. When relay information changes, Idealized Tor assumes that clients and relays update only the information that has changed. However, Idealized Tor ignores overhead incurred in practice for data requests such as packet headers or padding. Further, we assume the bare minimum of relay information, and do not include fields required in practice in Tor such as exit policies.

We provide more detail on the estimated numbers for Idealized Tor in Table 8.1.

How Idealized is Idealized Tor? We use the same network size and rate of relay information for Idealized Tor as today's Tor network (as further described below in Section 8.2.4). However, because our models for Idealized Tor do not account for overhead of transmitting data in practice, such as document headers, and only account for a minimum subset of relay information, the cost to transmit network directory documents is *underestimated* for Idealized Tor in comparison to the real Tor network. Our measurements show that a Tor client requires approximately 2.49 megabytes of data to bootstrap and 48 kilobytes to sync updated directory information each epoch. For the same network size, Tor relays require 10.3 megabytes of data to bootstrap and 608 kilobytes of updated network directory information to sync every network epoch. In comparison, our model for Idealized Tor requires both clients and relays to download 832 kilobytes of directory information upon bootstrapping to the network, and 158 kilobytes to sync updated directory information for each subsequent epoch.

8.2 Bandwidth Analysis

We now compare the cost in bandwidth required by Idealized Tor and Telescoping and Single-Pass Walking Onions. We assess the derived costs for distributing network directory documents in Section 8.2.1 and the derived cost of circuit construction in Section 8.2.2. We review the growth rates of each protocol in Section 8.2.3 and finally perform a comparison in real numbers and assess the total bandwidth costs in Section 8.2.4.

8.2.1 Distributing Network Directory Documents

We now evaluate the cost of maintaining network state for Idealized Tor and Telescoping and Single-Pass Walking Onions; we summarize our findings in Table 8.2.

We represent the number of relays in the anonymity network by N_R. In Idealized Tor, the information distributed to the network about each relay—termed a *relay entry*—is B_R in size;

Table 8.1: Notation and Estimated Values in a Tor-like network
Notation used in our performance assessment, and their estimated values based on a simplification of Tor ignoring implementation details such as legacy protocols and padding.

Notation	Description	Estimated value
N_R	Number of relays on the network	–
N_C	Number of clients on the network	–
N_H	Number of hops in a circuit path	3
N_V	Number of trusted voters constituting the network authority	–
B_R	Size of a relay entry	128 bytes
B_O	Additional fields needed for a SNIP	32 bytes
B_{AUTH} (threshold signatures)	Authentication tag size per SNIP	76 bytes
B_{AUTH} (Merkle proofs)	Authentication tag size per SNIP	$32\lceil \lg N_R \rceil$ bytes
B_{SNIP} (threshold signatures)	Total size of a SNIP $= B_R + B_O + B_{AUTH}$	236 bytes
B_{SNIP} (Merkle proofs)	Total size of a SNIP $= B_R + B_O + B_{AUTH}$	$160 + 32\lceil \lg N_R \rceil$ bytes
B_{ID}	Size of node identifier (used in Vanilla Onion Routing)	66 bytes
B_{IDX}	Size of ENDIVE index (used in Walking Onions)	8 bytes
B_T	Size of a TTL field (used in Single-Pass)	1 byte
B_V	Size of a VRF output and proof (used in Single-Pass)	80 bytes
B_E	Size of authenticated encryption overhead	28 bytes
B_C	Size of circuit handshake request	32 bytes
B_S	Size of circuit handshake response	64 bytes
P_Δ	Fraction of relay information changed per epoch	1.9%
γ	Circuits per client per epoch	8.9
ϕ	0 if using Merkle proofs; 1 otherwise. Reflects if relays must download authentication information for each SNIP; see Section 5.3.)	–

Table 8.2: Bandwidth Costs

Total=Summation of each relay on the path, NDD=Network Directory Document

		Idealized Tor	Telescoping Walking Onions	Single-Pass Walking Onions
Clients	Bootstrap	$N_R B_R$	0	0
	Download a NDD (per epoch)	$N_R B_R P_\Delta$	0	0
	Circuit Build	$(N_H - 1)B_{ID} + N_H(B_C + B_S)$	$(N_H - 1)(B_{IDX} + B_{SNIP}) + N_H(B_C + B_S)$	$2B_C + B_T + N_H B_S + (N_H - 1)(B_{SNIP} + B_V + B_E)$
Relays	Bootstrap	$N_R B_R$	$N_R(B_R + B_O) + \phi N_R B_{AUTH}$	$N_R(B_R + B_O) + \phi N_R B_{AUTH}$
	Download, distribute a NDD (per epoch)	$\left(\frac{N_C}{N_R} + 1\right) N_R B_R P_\Delta$	$N_R(B_R + B_O)P_\Delta + \phi N_R B_{AUTH}$	$N_R(B_R + B_O)P_\Delta + \phi N_R B_{AUTH}$
	Circuit Build (Total)	$(N_H - 1)^2 B_{ID} + {N_H}^2(B_C + B_S)$	$(N_H - 1)^2(B_{IDX} + B_{SNIP}) + {N_H}^2(B_C + B_S)$	$(2N_H - 1)(2B_C + B_T) + N_H^2 B_S + (N_H - 1)^2(B_{SNIP} + B_V + B_E)$

this information changes in each epoch with probability P_Δ. The bandwidth cost for both clients and relays to bootstrap in Idealized Tor is $N_R B_R$, or the size of the network directory document. Both clients and relays keep their copy of the network directory document up to date by fetching updated relay information, which is equal to the size of the relay entries that change per epoch, or $N_R B_R P_\Delta$. We denote the number of clients as N_C, and we assume each relay serves network directory documents to an equal fraction of clients. Consequently, the cost in bandwidth to each relay in Idealized Tor to maintain and distribute an up-to-date network directory document is $\left(\frac{N_C}{N_R} + 1\right) N_R B_R P_\Delta$.

Now consider Walking Onions. We assume each SNIP contains the same amount B_R of relay information as a relay entry. Additional fields required by the SNIP, such as index ranges (as described in Section 5.2), are represented by B_O. The size of authentication information for each SNIP is represented by B_{AUTH}. The mechanism for authentication influences the size of B_{AUTH};

the size of threshold or one-per-voter signatures remains constant regardless of the size of the network N_R, but the size of each Merkle proof grows logarithmically relative to the number of relays $\lceil \lg N_R \rceil$, as further described in Section 5.3. We indicate the mechanism for authentication using a flag ϕ. The total size of the SNIP is denoted as $B_{SNIP} = B_R + B_O + B_{AUTH}$.

Bandwidth cost to relays. Relays in Walking Onions must maintain an up-to-date network directory document, although they do not serve these documents to clients. Each epoch, relays will download SNIPs that have changed. Note that the choice of authentication mechanism impacts the frequency of SNIP updates, as further discussed in Section 5.3. Consequently, relays download a total of $N_R(B_R + B_O)P_\Delta + \phi N_R B_{AUTH}$ bytes each epoch.

Bandwidth cost to clients. Because clients to not require maintaining information about all relays in the network, there is no bandwidth cost for clients in Walking Onions to fetch network directory documents. We do not here consider infrequent, out-of-band bootstrapping mechanisms for clients to learn a subset of the relays, as described in Section 6.6.

8.2.2 Circuit Construction

We now describe how Idealized Tor compares to Telescoping and Single-Pass Walking Onions in bandwidth cost for circuit establishment. We summarize these results in Table 8.2.

Bandwidth cost to clients. In Idealized Tor, clients send a node identifier of size B_{ID} to identify the next relay the circuit should extend to. The size of the client's circuit extension request is represented as B_C, and the size of the relay's response as B_S. To build an N_H-hop circuit in Idealized Tor, the client's total bandwidth is $(N_H - 1)B_{ID} + N_H(B_C + B_S)$. The coefficient $(N_H - 1)$ corresponds to the client not sending the node identifier when establishing the first hop in the circuit.[1]

For Telescoping Walking Onions, the client's total bandwidth for circuit construction is $(N_H - 1)(B_{IDX} + B_{SNIP}) + N_H(B_C + B_S)$. The client sends an index of size B_{IDX} to select the next hop, and receives a SNIP in return. For Single-Pass Walking Onions, the client's total bandwidth for circuit construction is similar, totalling $2B_C + B_T + N_H B_S + (N_H - 1)(B_{SNIP} + B_V + B_E)$. We estimate the size of the path-selection and circuit public keys to each be B_C bytes in size; hence a client's cost to initialize Single-Pass Walking Onions is $2B_C + B_T$, where B_T represents the TTL value the client sends to indicate the path length. Clients in Single-Pass Walking Onions receive $N_H B_S + (N_H - 1)(B_{SNIP} + B_V + B_E)$ bytes in response; B_V represents the size of the VRF proof, and B_E represents the authenticated encryption overhead for

[1]Our calculations ignore the effect of message padding for simplicity, although this is often required in a live anonymity network.

the response. Note that with Single-Pass Walking Onions, the size of additional fields in a SNIP B_{AUTH} also includes each relay's public path-selection keys, and hence will slightly larger than in Telescoping.

Bandwidth cost to relays. We measure the total bandwidth cost to construct a single circuit for all relays in the circuit as the number of bytes sent and received. Note that when $N_H > 1$, some handshake messages must be relayed by all intermediate hops on the circuit. With this in mind, we estimate the total bandwidth cost for all relays in a circuit in Idealized Tor as $\sum_{j=1}^{N_H-1}(2j-1)B_{ID} + \sum_{j=1}^{N_H}(2j-1)(B_C + B_S) = (N_H - 1)^2 B_{ID} + {N_H}^2(B_C + B_S)$. For Telescoping Walking Onions, the total bandwidth is ${N_H}^2(B_{IDX} + B_C + B_S + B_{SNIP})$, while for Single-Pass Walking Onions, the total bandwidth cost is $(2N_H - 1)(2B_C + B_T) + N_H^2 B_S + (N_H - 1)^2(B_{SNIP} + B_V + B_E)$.

8.2.3 Analysis of Growth Rates

We now discuss the growth rates and performance implications of each protocol as the network grows large.

In Telescoping Walking Onions, clients begin to save bandwidth over Idealized Tor when $\frac{N_R}{\gamma} > \sigma = \frac{N_H-1}{B_R P_\Delta}(B_{IDX} + B_{SNIP} - B_{ID})$ — in other words, when the number of relays is high relative to the number of circuits that each client creates in an epoch. Regardless of the authentication mechanism, the left side of the equation quickly overtakes the right as N_R grows. Because σ is a constant independent of the size of the network when threshold or one-per-voter signatures are employed, and grows at rate $\lceil \lg N_R \rceil$ when using Merkle proofs, the growth rate of the N_R term still outweighs the growth of B_{SNIP}.

For overall bandwidth usage (clients plus relays), Telescoping Walking Onions outperforms Idealized Tor in bandwidth cost for relays when $N_R\left[(MB_R - B_O)P_\Delta - \phi B_{AUTH}\right] > \gamma M(N_H - 1)^2(B_{IDX} + B_{SNIP} - B_{ID})$, assuming each relay handles $\gamma N_C \frac{N_H}{N_R}$ circuits per epoch, and $M = \frac{N_C}{N_R}$ is the ratio of clients to relays. Again in this equation, $[(MB_R - B_O)P_\Delta - \phi B_{AUTH}]$ is a constant term independent of network growth, and the term $\gamma M(N_H - 1)^2(B_{IDX} + B_{SNIP} - B_{ID})$ represents the amount of data relays sent during circuit creation, whose growth is at most logarithmic relative to N_R when Merkle proofs are used. Therefore, the left side will again dominate as N_R grows.

The calculation for Single-Pass Walking Onions is similar. Note that while Single-Pass Walking Onions incurs slightly higher bandwidth overhead for smaller network sizes, it converges to that of Telescoping Walking Onions for networks of larger sizes. Because Single-Pass Walking Onions requires slightly larger SNIPs due to the VRF path-selection keys, and transmits slightly more data during circuit construction due to the VRF proofs, this overhead is distinguishable for smaller networks but is quickly dominated by other factors for larger networks.

8.2.4 Comparing Total Bandwidth Costs

Does Walking Onions save bandwidth in comparison to Idealized Tor? We experiment how Walking Onions protocols compare to Idealized Tor using estimated values presented in Table 8.1 and our bandwidth formulas from Table 8.2. We start by describing our methodology for instantiating actual paramaeters for Idealized Tor, and then present our results.

Recall that these estimates for Tor's directory protocol are *significantly* more efficient than the real Tor network, as discussed in Section 8.1.

Parameters for Idealized Tor

We first give more detail about our methodology for determining parameters for Idealized Tor.

Network size and client behaviour. We obtain our estimates of the current number of relays $N_R = 6,500$ and clients $N_C = 2,500,000$ from Tor's metrics site [Tor19a, Tor19b] as of 28 July 2019, rounding to two significant digits. We maintain this ratio of clients to relays, but experiment with different scaling factors. We compute the number of circuits γ each client makes per epoch by noting that Tor nodes log the total number of circuits they help create every six hours. We gathered 24 hours of these logs from a Tor exit node, and divided by the node's exit probability to estimate that the Tor network as a whole creates 536 million circuits per day, or 22 million per epoch.[2] Dividing by N_C yields $\gamma = 8.9$ for today's Tor network. We also experiment with a future anonymity network with increased utilization by assessing performance when $\gamma = 89$.

Tor circuits are usually $N_H = 3$ hops long.

Rate of network change. We estimate the current rate of change in Tor network information by an experiment in which we instrumented a Tor client to bootstrap and maintain a fully up-to-date directory for 24 hours, and to record the (compressed) bandwidth it used in doing so. Comparing the average hourly download size to the amount of information downloaded at bootstrap, we obtained an estimate of $P_\Delta = 1.9\%$.

Cryptographic size assumptions. We assume the use of elliptic-curve public keys that can be represented in 32 bytes, such as Curve25519 [LHT16], and whose signatures are of size

[2]This computation only counts circuits with an exit node, and not, for example, onion service circuits, but onion services are only about 1% of Tor's traffic. [Tor20c]

64 bytes. Furthermore, we assume hashing operations produce a digest of size 32 bytes. Authenticated encryption (for example, AES-GCM) requires $B_E = 28$ bytes for the nonce and authentication tag in total. We assume the use of the IETF-proposed VRF [GRPV19], which has a $B_V = 80$ byte output, including the proof.

Directory information. We calculate the number of bytes B_R needed to represent a relay as follows: identity public key (32 bytes), onion public key (32 bytes), IPv4 address (4 bytes), IPv6 address (16 bytes), exit policy digest (32 bytes), supported bandwidth (4 bytes), and usage/protocol flags (8 bytes). We represent this sum as B_R, totaling 128 bytes. Note, as mentioned in Section 8, we model Idealized Tor as a *significantly more efficient* onion routing network than existing Tor, as an underestimation of onion-routing networks in practice.

Circuit Extension Calculation. The Vanilla Onion Routing handshake (based on Tor's ntor) requires that the client send an ephemeral public key; the relay replies with an ephemeral public key and an authentication tag. These values make the request and response costs (B_C and B_S) 32 and 64 bytes respectively.

To identify a particular relay, Tor clients send an IPv4 address and port (6 bytes), a 20-byte identity digest, 8 bytes of headers, and a 32-byte public key that they believe the relay has. This makes $B_{ID} = 66$ bytes. An index, on the other hand, would be represented with 4 bytes of index and 4 bytes of headers, for a total of $B_{IDX} = 8$ bytes. Time-to-live fields θ are $B_T = 1$ byte.

SNIP size. We assume that Tor will need four separate indices for its routing: one for each of the Guard, Middle, and Exit positions, and one for Onion Service directories. Each index range on a SNIP would take 8 bytes (4 bytes for the start of the range, and 4 bytes for the end). This gives us a total SNIP overhead of $B_O = 4 \cdot 8$ bytes. We can represent the SNIP's validity period with 12 bytes (8 for the starting time, and 4 for its duration in seconds), and use a further 64 bytes for a signature.

We present threshold signatures and Merkle proofs for SNIP authentication in Section 5.3. With threshold signatures we have timestamped signatures of $B_{AUTH} = 64 + 12$ bytes (64 bytes for signature, 8 for start time, 4 for duration); with Merkle proofs we have $B_{AUTH} = 32\lceil \lg N_R \rceil$ bytes. (A single authentication tag exists, and is fetched once per epoch as negligible overhead.)

Results

We present our complete results in Figures 8.1, 8.2, 8.3, and 8.4, but summarize our findings here.

Total cost to clients. As demonstrated in Figure 8.1, clients in Walking Onions benefit in reduced bandwidth costs over Idealized Tor at any network size larger than the current Tor

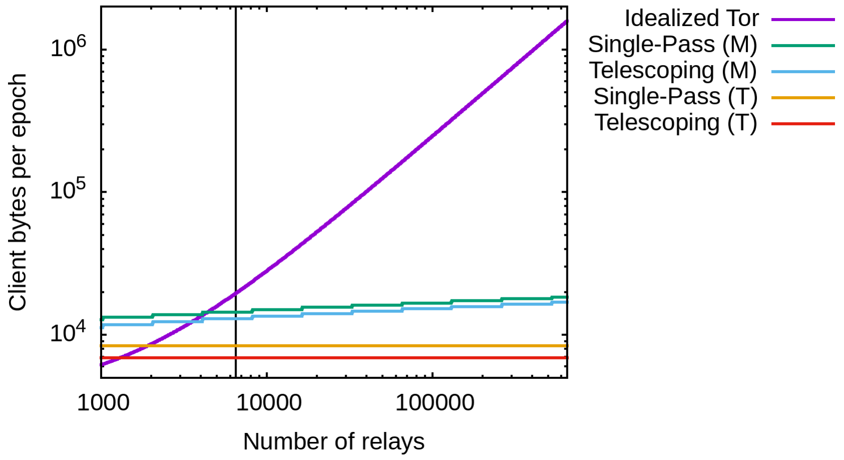

Figure 8.1: Client Bandwidth Utilization, for $\gamma = 8.9$

Crossover point (when $\gamma = 8.9$) where Walking Onions demonstrates client bandwidth savings as compared to Idealized Tor. The approximate number of relays in today's Tor network is indicated by the black vertical line. (M) denotes Merkle Signatures; (T) represents Threshold Signatures.

network (and even for somewhat smaller networks as well), when the average number of circuits γ created per client per epoch and the ratio of clients to relays are close to that of today's Tor. Further, while the cost in bandwidth to clients using Idealized Tor grows linearly relative to the number of relays, the cost to clients using Walking Onions instantiated with Merkle proofs grows logarithmically. Clients using Walking Onions instantiated with threshold signatures do not experience *any* growth in bandwidth as the number of relays increases.

Specifically, our measurements presented in Figure 8.1 indicate that Walking Onions outperforms Idealized Tor using Vanilla Onion Routing for clients when the network is of size between 1,300 and 4,800, depending on which protocol variant of Walking Onions is used and authentication method. The crossover of clients in Telescoping Walking Onions is at 1,300 relays when using threshold signatures and 3,500 relays when using Merkle proofs. For clients using Single-Pass Walking Onions, the crossover point is at 2,000 Relays when using threshold signatures and 4,700 when using Merkle proofs.

As such, all variants of Walking Onions provide bandwidth savings to clients in networks the size of today's Tor network, and this bandwidth savings increases as the network grows.

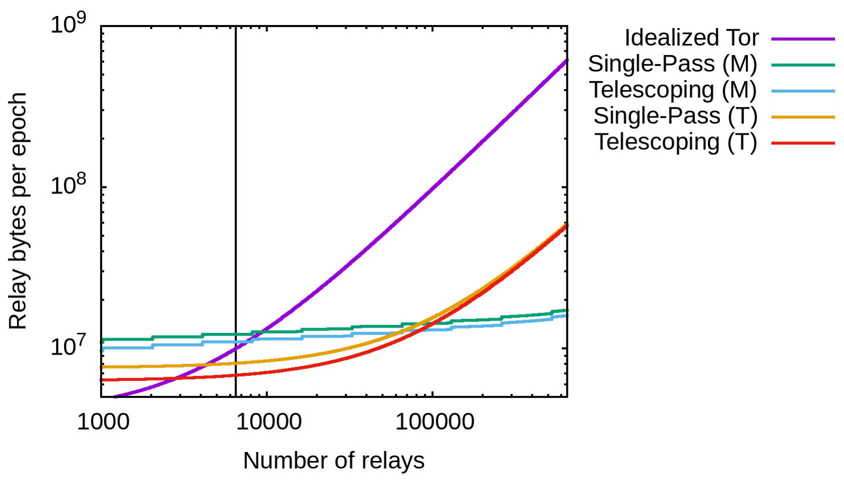

Figure 8.2: Relay Bandwidth Utilization, for $\gamma = 8.9$

Crossover point (when $\gamma = 8.9$) where Walking Onions demonstrates relay bandwidth savings as compared to Idealized Tor. The approximate number of relays in today's Tor network is indicated by the black vertical line. (M) denotes Merkle Signatures; (T) represents Threshold Signatures.

Total cost to relays. Estimating relay bandwidth savings follows a similar trend, as demonstrated in Figure 8.2. For relays, the bandwidth benefit of Walking Onions over Idealized Tor similarly increases as the size of the network increases. We find that for Tor's current γ, the crossover point for relays (the size of a network above which Walking Onions is more bandwidth efficient) is either below, or within a factor of 2, of the current Tor network size. While relays must use more bandwidth in circuit construction in Walking Onions than in Idealized Tor, as the numbers of clients and relays increase, relays save in bandwidth in Walking Onions due to the savings of not transmitting network directory documents to clients in every epoch. Consequently, while the rates of growth in bandwidth for relays remain linear for Walking Onions as in Idealized Tor, the bandwidth cost of Walking Onions to relays grows at a smaller factor as compared to Idealized Tor.

We expect Walking Onions to offer better performance for relays when the network size is between 2,900 and 10,000 relays, again depending on choice of protocol and authentication method.

When comparing Telescoping and Single-Pass Walking Onions, we find that Single-Pass has

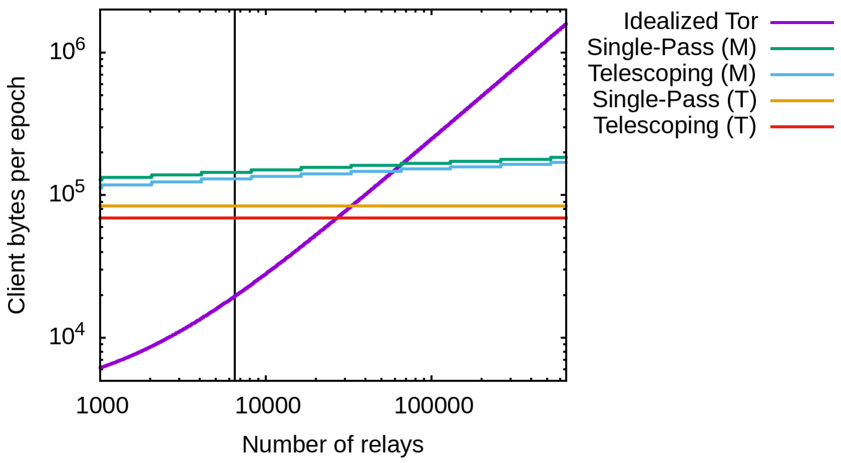

Figure 8.3: Client Bandwidth Utilization, for $\gamma = 89$

Crossover point (when $\gamma = 89$) where Walking Onions demonstrates client bandwidth savings as compared to Idealized Tor. The approximate number of relays in today's Tor network is indicated by the black vertical line. (M) denotes Merkle Signatures; (T) represents Threshold Signatures.

slightly higher bandwidth costs than Telescoping due to transmitting additional key material for the VRF. However, Single-Pass trades this bandwidth for improved latency, as we discuss in Section 8.3.

Performance under Higher Utilization. As seen in Figures 8.3 and 8.4, we also experiment with different network utilization levels by increasing the circuit construction rate γ. Assuming a higher γ, bandwidth savings for clients in Walking Onions outperforms Idealized Tor using Vanilla Onion routing when the network grows large. At a higher γ, bandwidth savings for relays in Walking Onions utilizing threshold signatures is more efficient for networks of smaller size. However, if the number of relays in the network is sufficiently large relative to γ, then Merkle proof authentication becomes a strong benefit for relays, at a slight cost to clients.

8.3 Latency Evaluation

The primary savings we expect for Single-Pass Walking Onions over Telescoping or Idealized Tor is in the reduced latency experienced by clients during circuit construction. The improve-

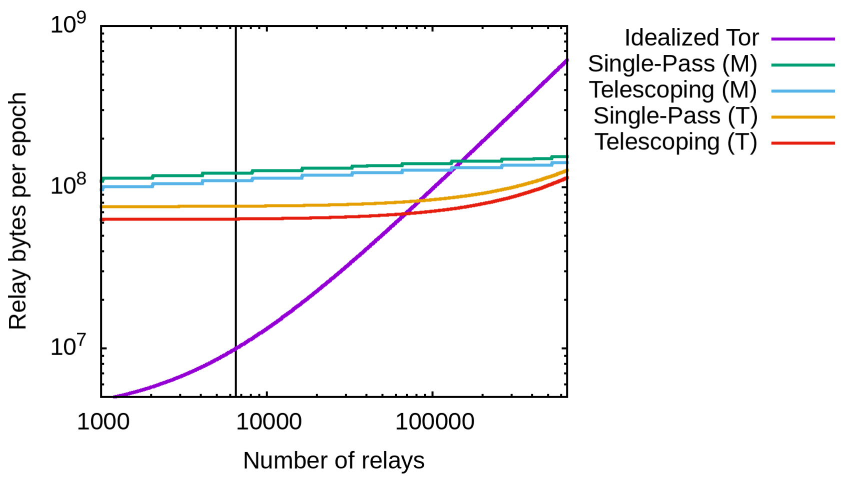

Figure 8.4: Relay Bandwidth Utilization, for $\gamma = 89$

Crossover point (when $\gamma = 89$) where Walking Onions demonstrates relay bandwidth savings as compared to Idealized Tor. The approximate number of relays in today's Tor network is indicated by the black vertical line. (M) denotes Merkle Signatures; (T) represents Threshold Signatures.

ment of Single-Pass Walking Onions is the same as that of other single-pass circuit construction proposals: whereas Idealized Tor and Telescoping both require a total of $N_H(N_H + 1)$ messages before the circuit can be constructed, Single-Pass Walking Onions requires 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 or Idealized Tor.

8.4 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

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

Table 8.3: Number of group operations to construct a circuit.
This assessment excludes SNIP validation

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

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 8.3 and ENDIVE generation and validation in Table 8.4.

8.4.1 CPU Cost to Extend a Circuit

For specificity, we instantiate the generic authenticated key exchange and VRF functions from Section 6 with ntor [GSU12], requiring two public key operations for both clients and relays, and the IETF-proposed VRF [GRPV19], 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 Idealized Tor. For Single-Pass Walking Onions, additional group operations are required for the blinding and VRF computations.

As described in Table 8.4, 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.

8.4.2 CPU Cost for ENDIVE Generation and Validation

In Section 5.3, we present several mechanisms as options for performing SNIP and ENDIVE authentication by the network authority; we now evaluate the performance in CPU of these options here.

Table 8.4: Costs of ENDIVE Generation, Signature Storage

CPU cost measured in public-key operations, signature size measured in bytes.

	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

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 proofs 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 proof 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.

8.5 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 [MOT$^+$11] instantiated with both Computational PIR (C-PIR) and Information-Theoretic

PIR (IT-PIR) designs, as well as ConsenSGX [SG19], which relies on trusted hardware. Recall that we expand on the design and security and efficiency tradeoffs of each of these designs in Section 3.1.2.

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 [MOT+11] 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 5.2) 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 scale sublinearly for client bandwidth usage as the size of the network grows, while the cost to clients in Walking Onions remains constant. Further, for IT-PIR and C-PIR schemes, server computation scales linearly relative to the network size. 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.

8.6 Summary

In this chapter, we present a performance evaluation for Walking Onions against an idealized version of path-selection and circuit-establishment protocols in Tor. We review high-level as well as derived bandwidth evaluations using a model of the Tor network as our case study. We determine that the crossover point in which Walking Onions demonstrates bandwidth savings to both clients and relays over Idealized Tor is at network sizes comparable to today's Tor network.

We present a comparison of the cost in CPU between Walking Onions and Idealized Tor, and discuss how different authentication mechanisms impact the resulting CPU and bandwidth efficiency of these protocols. Further, we compare the performance cost of Walking Onions to other designs with similar goals based on Private Information Retrieval, and discuss the security and complexity tradeoffs of these designs.

Chapter 9

Applications of Walking Onions

Throughout this work, we have used Tor as a case study to demonstrate how Walking Onions can be used by anonymity networks with similar threat models and scalability concerns, and whose designs build upon source-based onion routing constructions. However, demonstrating more widespread applicability of Walking Onions is not only important, but possible due to the generality of the protocols and techniques presented in this work.

In this chapter, we will discuss applications of Walking Onions to other anonymity network designs to demonstrate the generality of Walking Onions beyond Tor. In Section 9.1 we discuss how Walking Onions can be applied to HORNET, a high-performance onion-routing protocol proposed in the literature. In Section 9.2, we outline how Walking Onions techniques can be incorporated into Lightning, a set of privacy and performance-enhancing protocols for the Bitcoin network. Finally, in Section 9.3, we present several options for incorporating Walking Onions into anonymity networks implementing the Loopix mix network design.

9.1 Applying Walking Onions to HORNET

HORNET [CAB$^+$15] presents an onion-routing protocol that is optimized for performance and is stateless for intermediate nodes in a path. Messages sent through the network are encrypted to each hop in such a way 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, including routing information and the node's shared symmetric key with the client. The protocol requires a one-time setup phase in which the circuit is established in a single pass using a variant of Sphinx [DG09].

While HORNET is optimized for performance, 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, requiring that paths selected by clients are fully formed (i.e, clients select entire paths, not individual relays to make up a path) 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, thereby achieving this best-of-both-worlds approach.

Second, in order 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 in order to encrypt data to this hop 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. Consequently, an adversary who compromises the node's long-lived key can recover the routing information exposed to this node. 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. Because nodes in the Single-Pass Walking Onions periodically rotate their keys used to set up the circuit, path selection is eventually forward secure without requiring the client to perform an update of more-recently published network directory information.

9.2 Applying Walking Onions to Lightning

The Lightning Network protocol [Osu16, Lig19] is a proposed network upgrade to the Bitcoin network to provide improved scalability. However, in addition to proposing scalability improvements, the protocol also includes an onion-routing protocol to improve network privacy. The proposed protocol is source-routed, such that nodes select paths using their own local knowledge of the network (which may be inconsistent with the views of other nodes, similarly to any completely decentralized network), and messages are sent in a format modeled after the Sphinx message format.

The Lightning protocol currently is based on a single-pass design where the initiator creates a fresh ephemeral key for each circuit, and establishes a shared key using each relay's static public key. In contrast, Single-Pass Walking Onions offers a drop-in replacement option with improved security properties, as communication in Single-Pass remains forward secure.

Unlike in Tor, the Bitcoin network does not have a central PKI; currently, nodes use information that is gossiped to them about other relays without having a reliable mechanism to ensure the validity of this information. As such, designing an "authority" for a completely decentralized network such as the Bitcoin network must employ a different network authority strategy. One option for designing an appropriate authority for a completely decentralized network includes a heuristic-based approach based on the information exchanged in the underlying gossip protocol. For example, a node that has received gossip information from a node that only recently joined the network could be weighted as less reliable as opposed to information received from a longer-lived node.

An important point to note when considering the use of Walking Onions in a fully decentralized network is how each relay's network view impacts the client's path selection, as the selection of a relay for the next hop in the Walking Onions design is impacted by the size of the local routing table held by each relay. In a network where the view of a relay can be influenced by its peers (such as by propagating false gossip messages or refusing to relay gossip messages), this effect will also influence the security of path selection in Walking Onions.

9.3 Applying Walking Onions to Loopix-Based Mixnets

Loopix [PHE+17] is a mix network design that injects message delays and dummy messages to hide a message's path from source to destination against a global adversary. The frequency of these message delays and dummy packets are chosen from a Poisson distribution, instead of performing global coordination for mixing, as was common in earlier mix network designs [Cha81, KEB98, BFK01].

Notably, the Loopix design assumes a PKI, ensuring that clients can be assured of the identity and corresponding public keys for mix servers in the network. Further, Loopix requires the network to be *horizontally* stratified, such that a user building a path will select one server from each layer. This design approach entails the client's path is as long as the number of horizontal layers in the network. Loopix assumes all users build paths with equal probability through any mix server within a single layer. Consequently, because Loopix is also source-routed, the client selects the path for their message to take from the complete set of all mix servers. As such, Loopix assumes that clients maintain an up-to-date network directory document that is authenticated using some authority mechanism.

Currently, mix servers in Loopix send messages statelessly, using the Sphinx packet format such that private symmetric keys encrypting each layer over the message are encrypted to each node's public key, and then included in the packet itself. When each node receives a packet, it first uses its corresponding private key to decrypt its symmetric key, and then uses this symmetric key to unwrap one layer of encryption of the message. The node then processes the message headers which include instructions from the client such as the amount of delay to wait before forwarding the message and information for the next relay int he path. Consequently, this design is not forward secure for client communication due to the use of a relay's static keys to encrypt message keys for each packet during transmission.

Because Walking Onions requires a circuit setup phase between clients and relays, and sends a stream of messages between client and destination using this circuit, implementing Walking Onions in a Loopix-based mix network system requires trading the efficiency of stateless message-based architecture for improved scalability and forward secrecy of client communication. The resulting design would become a hybrid between mix network design and onion routing, by utilizing longer-lived fixed circuits for message transmission between two parties, but injecting message delays and dummy packets for frustrating a global adversary.

9.4 Summary

In this chapter, we introduce three additional case studies where Walking Onions can offer security and performance improvements to anonymity network designs other than Tor. We introduce HORNET and describe how Walking Onions can be incorporated into the design, so long as the design can support freely routed path selection. We then introduce the Lightning onion-routing protocol whose design is intended to improve network privacy in Bitcoin, and describe how the use of Single-Pass Walking Onions can improve the forward-secrecy of communication in this design. We note how the security tradeoffs of completely decentralized networks such as Bitcoin similarly impacts the security guarantees offered by Walking Onions. Finally, we review how Loopix-based networks can incorporate Walking Onions, so long as the network can support sending traffic through longer-lived circuits, as opposed to requiring relays to send traffic statelessly.

Chapter 10

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 aimed to demonstrate the positive conclusion for the following hypothesis:

It is possible to design an anonymity network whose clients maintain a constant amount of network information even as the network grows, and can build circuits through the network that require a linear number of messages relative to the length of the circuit. Furthermore, it is possible that an anonymity network with such a design can remain secure against epistemic and path-based attacks that previously have only been ameliorated by designs that require all participants to retain a globally consistent view of the network.

As demonstrated in this work, we can conclude with a positive affirmation of the above hypothesis. As evidence towards this conclusion, 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 global 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.

References

[ADD$^+$17] 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.

[ADPS16] Yawning Angel, Claudia Diaz, Ania Piotrowska, and David Stainton. Katzenpost Mix Network Public Key Infrastructure Specification. https://github.com/katzenpost/docs/blob/master/specs/pki.rst, 2016. last accessed 2019-09-25.

[BDMT07] 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, New York, NY, USA, 2007. ACM.

[BDN18] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact Multi-signatures for Smaller Blockchains. In *Advances in Cryptology – ASIACRYPT 2018*, pages 435–464, Cham, 2018. Springer International Publishing.

[BFK01] Oliver Berthold, Hannes Federrath, and Stefan Köpsell. *Web MIXes: A System for Anonymous and Unobservable Internet Access*, pages 115–129. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[BGLS03a] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. A Survey of Two Signature Aggregation Techniques. *CryptoBytes*, 6, 09 2003.

[BGLS03b] 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, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[BLS04] Dan Boneh, Ben Lynn, and Hovav Shacham. Short Signatures from the Weil Pairing. *Journal of Cryptology*, 17(4):297–319, Sep 2004.

[BMG$^+$07] Kevin Bauer, Damon McCoy, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. Low-resource Routing Attacks Against Tor. In *Proceedings of the 2007 ACM Workshop on Privacy in Electronic Society*, WPES '07, pages 11–20, New York, NY, USA, 2007. ACM.

[BNN07] Mihir Bellare, Chanathip Namprempre, and Gregory Neven. Unrestricted Aggregate Signatures. In *Automata, Languages and Programming*, pages 411–422, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[Bon11] Dan Boneh. Aggregate signatures. In *Encyclopedia of Cryptography and Security, 2nd Ed*, page 27. Springer, 2011.

[BVFV17] Shaileshh Bojja Venkatakrishnan, Giulia Fanti, and Pramod Viswanath. Dandelion: Redesigning the Bitcoin Network for Anonymity. *Proc. ACM Meas. Anal. Comput. Syst.*, 1(1):22:1–22:34, June 2017.

[CAB$^+$15] 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.

[CDRF$^+$13] 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.

[CFG09] Dario Catalano, Dario Fiore, and Rosario Gennaro. Certificateless Onion Routing. In *16th ACM conference on Computer and Communications Security*, pages 151–160. ACM, 2009.

[Cha81] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, February 1981.

[CK02] 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. Springer-Verlag, 2002.

[CL05] Jan Camenisch and Anna Lysyanskaya. A formal treatment of onion routing. In *Advances in Cryptology – CRYPTO 2005*, pages 169–187, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[DC06] George Danezis and Richard Clayton. Route Fingerprinting in Anonymous Communications. In *Sixth IEEE International Conference on Peer-to-Peer Computing (P2P'06)*, pages 69–72, Sep. 2006.

[DG09] George Danezis and Ian Goldberg. Sphinx: A Compact and Provably Secure Mix Format. *30th IEEE Symposium on Security and Privacy*, pages 269–282, 2009.

[Din09] Roger Dingledine. One cell is enough to break Tor's anonymity. https://blog.torproject.org/one-cell-enough-break-tors-anonymity, 2009. last accessed 2019-12-16.

[DM20] Roger Dingledine and Nick Mathewson. Tor Protocol Specification. https://gitweb.torproject.org/torspec.git/tree/tor-spec.txt, 2020. last accessed 2020-06-22.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The Second-Generation Onion Router. In *USENIX Security Symposium*, 2004.

[DS08] George Danezis and Paul Syverson. Bridging and Fingerprinting: Epistemic Attacks on Route Selection. In *Privacy Enhancing Technologies*, pages 151–166, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[DSS05] Roger Dingledine, Vitaly Shmatikov, and Paul Syverson. Synchronous Batching: From Cascades to Free Routes. In *Proceedings of the 4th International Workshop on Privacy Enhancing Technologies*, PET'04, pages 186–206, Berlin, Heidelberg, 2005. Springer-Verlag.

[EBA$^+$12] 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, New York, NY, USA, 2012. ACM.

[Fel87] Paul Feldman. A Practical Scheme for Non-interactive Verifiable Secret Sharing. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, SFCS '87, pages 427–438, Washington, DC, USA, 1987. IEEE Computer Society.

[FVB$^+$18] Giulia Fanti, Shaileshh Bojja Venkatakrishnan, Surya Bakshi, Bradley Denby, Shruti Bhargava, Andrew Miller, and Pramod Viswanath. Dandelion++: Lightweight Cryptocurrency Networking with Formal Anonymity Guarantees. *Proc. ACM Meas. Anal. Comput. Syst.*, 2(2):29:1–29:35, June 2018.

[Gab19] Gaba. Mozilla Research Call: Tune up Tor for Integration and Scale. https://blog.torproject.org/mozilla-research-call-tune-tor-integration-and-scale, 2019. last accessed 2019-08-13.

[GG18] Rosario Gennaro and Steven Goldfeder. Fast Multiparty Threshold ECDSA with Fast Trustless Setup. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 1179–1194, New York, NY, USA, 2018. ACM.

[GGN16] Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. Threshold-Optimal DSA/ECDSA Signatures and an Application to Bitcoin Wallet Security. In *Applied Cryptography and Network Security*, pages 156–174, Cham, 2016. Springer International Publishing.

[GHM$^+$17] 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, New York, NY, USA, 2017. ACM.

[GJKR03] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure Applications of Pedersen's Distributed Key Generation Protocol. In Marc Joye, editor, *Topics in Cryptology — CT-RSA 2003*, pages 373–390. Springer, 2003.

[GJKR07] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. *Journal of Cryptology*, 20:51–83, 05 2007.

[GRPV19] 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.

[GRS96] David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. Hiding Routing Information. In *Information Hiding: First International Workshop*, pages 137–150. Springer-Verlag, LNCS 1174, May 1996.

[GSU12] Ian Goldberg, Douglas Stebila, and Berkant Ustaoglu. Anonymity and one-way authentication in key exchange protocols. *Designs, Codes and Cryptography*, 67, 02 2012.

[I2P10] I2P Project. I2P Threat Model. https://geti2p.net/en/docs/how/threat-model, 2010. last accessed 2019-06-11.

[KBS20] Christiane Kuhn, Martin Beck, and Thorsten Strufe. Breaking and (Partially) Fixing Provably Secure Onion Routing. *41st IEEE Symposium on Security and Privacy*, pages 549–566, 2020.

[KEB98] Dogan Kesdogan, Jan Egner, and Roland Büschkes. Stop- and- go-mixes providing probabilistic anonymity in an open system. In *Information Hiding*, pages 83–98, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[KG09] Aniket Kate and Ian Goldberg. Distributed Key Generation for the Internet. In *29th IEEE International Conference on Distributed Computing Systems*, pages 119–128, June 2009.

[KG10] Aniket Kate and Ian Goldberg. Using Sphinx to Improve Onion Routing Circuit Construction. In *Financial Cryptography and Data Security*, pages 359–366, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[KMG20] Chelsea Komlo, Nick Mathewson, and Ian Goldberg. Walking Onions: Scaling Anonymity Networks while Protecting Users. In *29th USENIX Security Symposium*, 2020.

[KZG07] Aniket Kate, Greg Zaverucha, and Ian Goldberg. Pairing-Based Onion Routing. In *Privacy Enhancing Technologies*, pages 95–112, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[LHT16] Adam Langley, Mike Hamburg, and Sean Turner. IETF RFC 7748: Elliptic Curves for Security. RFC 7748, RFC Editor, January 2016.

[Lig19] The Lightning Network. BOLT Number 4: Onion Routing Protocol. https://github.com/lightningnetwork/lightning-rfc/blob/master/04-onion-routing.md, 2019. last accessed 2019-12-15.

[LLK13] Ben Laurie, Adam Langley, and Emilia Kasper. IETF RFC 6962: Certificate Transparency. RFC 6962, RFC Editor, June 2013.

[LMRS04] Anna Lysyanskaya, Silvio Micali, Leonid Reyzin, and Hovav Shacham. Sequential aggregate signatures from trapdoor permutations. In *Advances in Cryptology - EUROCRYPT 2004*, pages 74–90, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[LOS$^+$06] Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential Aggregate Signatures and Multisignatures Without Random Oracles. In

Advances in Cryptology - EUROCRYPT 2006, pages 465–485, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[Mat20a] Nick Mathewson. Proposal 206: Preconfigured directory sources for bootstrapping. https://gitweb.torproject.org/torspec.git/tree/proposals/206-directory-sources.txt, 2020. last accessed 2020-06-22.

[Mat20b] Nick Mathewson. Proposal 216: Improved circuit-creation key exchange. https://gitweb.torproject.org/torspec.git/tree/proposals/216-ntor-handshake.txt, 2020. last accessed 2020-06-22.

[Mat20c] Nick Mathewson. Walking Onions: Scaling and Saving Bandwidth. https://gitweb.torproject.org/torspec.git/tree/proposals/300-walking-onions.txt, 2020. last accessed 2020-06-22.

[MB09] 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, New York, NY, USA, 2009. ACM.

[MB12] Prateek Mittal and Nikita Borisov. Information Leaks in Structured Peer-to-Peer Anonymous Communication Systems. *ACM Trans. Inf. Syst. Secur.*, 15(1):5:1–5:28, March 2012.

[Mer88] 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, London, UK, UK, 1988. Springer-Verlag.

[MOG$^+$20] Kit Murdock, David Oswald, Flavio D. Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. Plundervolt: Software-based Fault Injection Attacks against Intel SGX. In *Proceedings of the 41st IEEE Symposium on Security and Privacy (S&P'20)*, 2020.

[MOR01] Silvio Micali, Kazuo Ohta, and Leonid Reyzin. Accountable-subgroup Multisignatures: Extended Abstract. In *Proceedings of the 8th ACM Conference on Computer and Communications Security*, CCS '01, pages 245–254, New York, NY, USA, 2001. ACM.

[MOT$^+$11] 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.

[MPSW19] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple Schnorr multi-signatures with applications to Bitcoin. *Designs, Codes and Cryptography*, Feb 2019.

[MTHK09] Jon McLachlan, Andrew Tran, Nicholas Hopper, and Yongdae Kim. Scalable Onion Routing with Torsk. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 590–599, New York, NY, USA, 2009. ACM.

[MVR99] Silvio Micali, Salil Vadhan, and Michael Rabin. Verifiable Random Functions. In *40th Annual Symposium on Foundations of Computer Science*, FOCS '99, pages 120–, Washington, DC, USA, 1999. IEEE Computer Society.

[MWBJ$^+$18] Akshaya Mani, T. Wilson-Brown, Rob Jansen, Aaron Johnson, and Micah Sherr. Understanding Tor Usage with Privacy-Preserving Measurement. In *Internet Measurement Conference 2018*, IMC '18, pages 175–187, New York, NY, USA, 2018. ACM.

[NW06] Arjun Nambiar and Matthew Wright. Salsa: A Structured Approach to Large-scale Anonymity. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, pages 17–26, New York, NY, USA, 2006. ACM.

[ØS07] Lasse Øverlier and Paul Syverson. Improving Efficiency and Simplicity of Tor Circuit Establishment and Hidden Services. In *Privacy Enhancing Technologies*, pages 134–152, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[Osu16] Olaoluwa Osuntokun. Privacy-Preserving Decentralized Micropayments: Onion Routing in Lightning. https://scalingbitcoin.org/milan2016/presentations/D1%20-%206%20-%20Olaoluwa%20Osuntokun.pdf, 2016. last accessed 2019-12-15.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology — EUROCRYPT '99*, pages 223–238, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[Ped91] Torben Pryds Pedersen. A Threshold Cryptosystem Without a Trusted Party. In *Proceedings of the 10th Annual International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'91, pages 522–526, Berlin, Heidelberg, 1991. Springer-Verlag.

[Ped92] Torben P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '91, pages 129–140, London, UK, UK, 1992. Springer-Verlag.

[PHE$^+$17] Ania M. Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. The Loopix Anonymity System. In *Proceedings of the 26th USENIX Conference on Security Symposium*, SEC'17, pages 1199–1216, Berkeley, CA, USA, 2017. USENIX Association.

[PM16] Trevor Perrin and Moxie Marlinspike. The Double Ratchet Algorithm. https://signal.org/docs/specifications/doubleratchet/, 2016. last accessed 2019-08-14.

[RDR97] Michael Reiter and Aviel D. Rubin. Crowds: Anonymity for Web Transactions. *ACM Transactions on Information and System Security (TISSEC)*, 1:66–92, 01 1997.

[RMR$^+$21] Hany Ragab, Alyssa Milburn, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. CrossTalk: Speculative Data Leaks Across Cores Are Real. In *Proceedings of the 42nd IEEE Symposium on Security and Privacy (S&P'21)*, 2021.

[RP02] Marc Rennhard and Bernhard Plattner. Introducing MorphMix: Peer-to-peer Based Anonymous Internet Usage with Collusion Detection. In *Proceedings of the 2002 ACM Workshop on Privacy in the Electronic Society*, WPES '02, pages 91–102, New York, NY, USA, 2002. ACM.

[RP04] Marc Rennhard and Bernhard Plattner. Practical Anonymity for the Masses with MorphMix. In *Financial Cryptography*, pages 233–250, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[RSG98] Michael G. Reed, Paul F. Syverson, and David M. Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communications*, 16(4):482–494, May 1998.

[Sch91] Claus P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, Jan 1991.

[SDH$^+$10] 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.

[SG19] Sajin Sasy and Ian Goldberg. ConsenSGX: Scaling Anonymous Communications Networks with Trusted Execution Environments. *PoPETs*, 2019(3):331–349, 2019.

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979.

[SS01] Douglas R. Stinson and Reto Strobl. Provably Secure Distributed Schnorr Signatures and a (T, N) Threshold Scheme for Implicit Certificates. In *Proceedings of the 6th Australasian Conference on Information Security and Privacy*, ACISP '01, pages 417–434, London, UK, UK, 2001. Springer-Verlag.

[SSA$^+$18] 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):51:1–51:39, June 2018.

[TB06] Parisa Tabriz and Nikita Borisov. Breaking the Collusion Detection Mechanism of Morphmix. In *Proceedings of the 6th International Conference on Privacy Enhancing Technologies*, PET'06, pages 368–383, Berlin, Heidelberg, 2006. Springer-Verlag.

[Tea11] JAP Team. JAP. https://anon.inf.tu-dresden.de/index_en.html, 2011. last accessed 2019-09-25.

[Tor19a] The Tor Project. Tor Metrics—Relays. https://metrics.torproject.org/networksize.html?start=2019-09-01&end=2019-12-02, 2019. last accessed 2020-06-22.

[Tor19b] The Tor Project. Tor Metrics—Users. https://metrics.torproject.org/userstats-relay-country.html?start=2019-09-01&end=2019-12-02&country=all&events=off, 2019. last accessed 2020-06-22.

[Tor20a] The Tor Project. Tor Directory Protocol Specification. https://gitweb.torproject.org/torspec.git/tree/dir-spec.txt, 2020. last accessed 2020-06-22.

[Tor20b] The Tor Project. Tor Guard Specification. https://gitweb.torproject.org/torspec.git/tree/guard-spec.txt, 2020. last accessed 2020-06-22.

[Tor20c] The Tor Project. Tor Metrics—Onion Services. https://metrics.torproject.org/hidserv-rend-relayed-cells.html, 2020. last accessed 2020-06-22.

[Tor20d] The Tor Project. Tor Path Specification. https://gitweb.torproject.org/torspec.git/tree/path-spec.txt, 2020. last accessed 2020-06-22.

[UG15] 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, New York, NY, USA, 2015. ACM.

[VBMW$^+$18] 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 *Proceedings of the 27th USENIX Security Symposium*. USENIX Association, August 2018.

[vdHLZZ15] Jelle van den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. Vuvuzela: Scalable Private Messaging Resistant to Traffic Analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, pages 137–152, New York, NY, USA, 2015. ACM.

[WMB10] 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, New York, NY, USA, 2010. ACM.

[ZH11] Bassam Zantout and Ramzi Ahmed Haraty. I2P Data Communication System. In *ICON 2011*, 2011.