Orchid: A Fully Distributed, Anonymous Proxy Network Incentivized Through Bandwidth Mining

David L. Salamon, Brian J. Fox, Jay Freeman, and Gustav Simonsson with Stephen F. Bell, and Dr. Steven Waterhouse, Ph.D.

October 25, 2017
Version 0.9.0

Abstract

As methods for censoring browsing and for discovering private browsing information have become more effective, the interest in anonymization methods has increased. Unfortunately, existing approaches to unrestricted, unsurveilled Internet access such as I2P and Tor suffer from a lack of widespread adoption. Indeed, only a few thousand unpaid volunteers host relays and exit nodes, allowing sophisticated attackers a tractable number of nodes to monitor or otherwise compromise. We present a market based, fully decentralized, and anonymous peer-to-peer system based on "bandwidth mining" which we believe addresses this lack of relay and exit nodes by directly incentivizing participants.

This paper is written to describe a system still under development. As such, it will undoubtably change and have new content added to address any implementation differences that arise; it is flexible in its use of library components and specific encryption algorithms. However, the essence of the system, its purpose and its goals will remain the same.

Contributions include:

- A blockchain-based stochastic payment mechanism with transaction costs on the order of a packet
- A commodity specification for the sale of bandwidth
- A method for distributed inductive proofs in peer-to-peer systems which make Eclipse attacks arbitrarily difficult
- An efficient security-hardened auction mechanism suited for the sale of bandwidth in circumstances where an attacker may alter their bid as part of an attack
- A fully distributed anonymous bandwidth market

Contents

1. Introduction

The Orchid Protocol organizes bandwidth sellers into a rigidly structured peer-to-peer (P2P) network termed the Orchid Market. Customers connect to the Orchid Market and pay multiple bandwidth sellers to form a proxy chain to a specific webserver. Proxy chains allow customers to send and receive data from the global Internet.

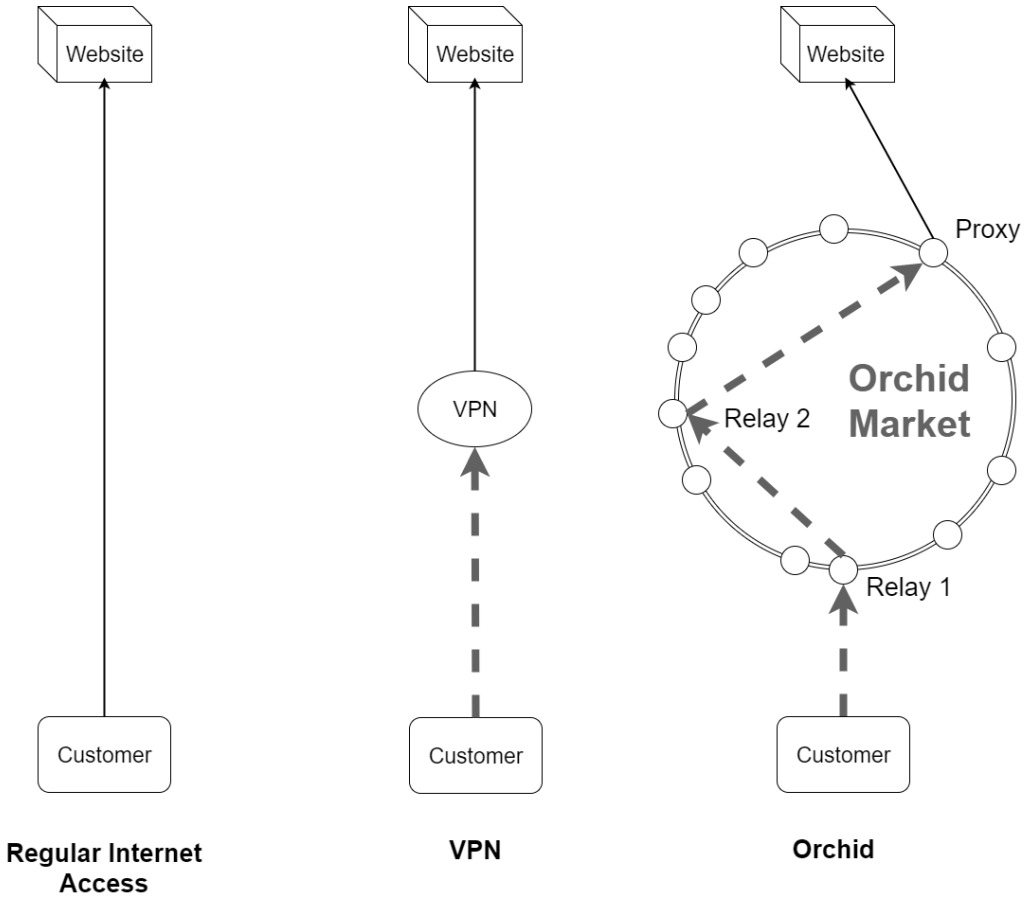

Figure 1: Direct connection, VPN connection, Orchid connection

Unlike more common methods for sending and receiving data from the global Internet, proxy chains in the Orchid Market naturally separate information about the source of data from information about its destination; no single relay or proxy holds both pieces of information, or knows the identity of someone who does. The rigid structure of the Orchid Market further supports this separation of information by providing strong resistance against *collusion attacks* – the ability of a group of bandwidth sellers to overcome this separation of knowledge.

Unlike less common methods for sending and receiving data from the global Internet, which do compartmentalize source and destination knowledge, the Orchid Market provides *fixed rate relaying* to prevent traffic analysis, and an incentive for participation not related to the hiding or discovery of information: payment in tokens.

Before we describe the details of the system, we will briefly review the core problems it solves, and the general solutions we have chosen for our system's foundation.

1.1. The *Traffic Analysis* Problem

Problem Statement: Imagine you are in a cafeteria full of mathematicians and wish to send a message to your friend across the room without anyone else knowing that fact. You have not already negotiated a message passing protocol, so all implementation details must be publicly stated to everyone the room. What can be done?

A particularly elegant solution to this problem, proposed by Chaum in 1981[57], is to have every person act as both a relay and a recipient. In this scheme, participants prepare encrypted messages which are the digital equivalent of "envelopes containing envelopes" – to send a message to Alice, you would compute

$$Enc(\text{“}ToBob''||Enc(\text{“}ToAlice''||Enc(message, Alice), Bob), Carol)$$

and send that message to Carol, who decrypts it and sends it to Bob, who decrypts it and sends it to Alice. To prevent traffic analysis everyone sends a fixed number of messages every cycle. To handle return addresses, we can have Bob and Carol remember a unique message identifier and send messages back along the chain.

Of particular importance to systems using the above method is the possibility of a *Collusion*. If Bob and Carol cooperate they can potentially determine who sent a given message, and to whom it was sent.

1.2. The *Sybil* Problem

The above cafeteria problem statement used physical bodies to prevent *Sybil Attacks* – situations in which one participant might pretend to be an arbitrarily large number of users. Unfortunately, in digital systems this approach cannot be used.

Problem Statement: How can we know that someone is "real" in a purely digital context?

A solution to this problem can be found in Hashcash[81]. If we require those claiming to be "real" to expend computational resources, we can put Sybil Attackers in a position where claims of being an incredible number of network participants requires actually possessing an incredible amount of computational resources.

1.3. The *Random Selection* Problem

The above cafeteria problem statement assumed an easy method for sending a message to every other user of the system (e.g., yelling across the cafeteria). To implement a Chaumian mix which is maximally resistant to *Collusion Attacks*, we need to be able to select randomly from those relays who are "real." Naively this requires being notified whenever someone joins or leaves the network. Unfortunately, in real-world P2P networks, having every user maintain such a list would result in an unacceptable amount of network traffic ($O(n^2)$ notifications.)

Problem Statement: How can we maintain a distributed list of all currently "real" relays which minimizes networking overhead and supports efficient random selection of peers?

A particularly elegant solution to this problem can be found in the Chord[81] Distributed Hash Table (DHT). In this scheme, peers are assigned unique addresses in a large space and then are connected in such a way that lookups can be performed in $O(log(n))$ time. Adding or removing a user only requires notifying $O(log(n))$ peers.

1.4. System Overview

The Orchid Protocol is, at its core, a combination of the above solutions. In our approach, peers are required to produce *Medallions* to demonstrate their "realness", and are then organized into a distributed

P2P network termed the *Orchid Market*. To keep the Orchid Market participants honest, every peer checks the correctness of its neighbor's behavior. Customers then use the Orchid Market to select random peers for Chaumian message forwarding. To incentivize participation, we have Customers pay Relays on a per-forwarded-byte basis.

This is a simple idea, but of course the devil is in the details. The system is to be fully decentralized, fully autonomous, fully anonymous, and is to handle payments. Much of this design document is therefore centered on preventing attacks on customer security, the system's performance, and the system's economic soundness. Although attack analysis is important, and will take up much of our time, it is ultimately no more than a necessary conceit to the context in which the market is to operate; if you find yourself "lost in the woods" we hope you will use the preceding exposition as your north star – the system's design details are all toward realizing a real-world solution to the above trio of problems.

2. Attacks

As much of this paper is on attack prevention, we begin by reviewing the literature on those attacks which are particularly common against P2P networks.

2.1. Attacker Goals

2.1.1. Information Gathering

The largest class of attacks against which the Orchid Protocol must defend against are those which reveal information about its users. Because Orchid is implemented as an overlay on the existing Internet, some information is *unavoidably* shared with some peers. In the list below, such information is marked with a "*". Any information which is not specifically listed as *unavoidably* shared in this document, but for which a method is discovered to uncover that information is termed an *informational attack* and is covered by Orchid's White Hat Bug Bounty. For more information on what is shared, see the protocol specification in Section 8 and discussion of collusion in Section 9.2, and our reference implementation of the network[1].

Types of data which are assumed to be of interest to an attacker (timeless):

- Real-World Identity Information. A user's given name, SSN, address, etc.
- Website Account Information. The user accounts at a specific website. Note this can be different from Real-World Identity Information.
- *IP Information. The IP address from which a user is accessing the Orchid Network. Note that in some usage scenarios this may be equivalent to learning Real-World Identity Information.
- *Ethereum Information. The keys associated with a user's wallet (*public or private). Note that in some usage scenarios this may be equivalent to learning Real-World Identity Information.
- *Orchid Network Information. The keys associated with a node's current business on the Orchid network (*public or private).

Types of Behavioral information which are assumed to be of interest to an attacker (time and Chain associated data):

- *Customer Identification. The attacker learns the IP address of a customer.
- *Relay Identification. The attacker learns the IP address of a relay.
- *Proxy Identification. The attacker learns the IP address of a proxy.
- *Link Identification. The attacker learns that two IP addresses were employed in a Chain.

- *Website Access. The attacker learns that an outbound connection was made from the Orchid network to a specific website.
- *Webserver Access. The attacker learns that an outbound connection was made from the Orchid network to a specific webserver (which may host multiple websites).
- *Ethereum Linking. The attacker learns that an Ethereum public key is held by a Orchid user.
- *Purchase Linking. The attacker learns that two transactions share a common payer.
- *Purchase Information. The attacker learns the quantity and timing of bandwidth sent over a Chain.

Although all of the above behavioral information is shared with other nodes on the Orchid Network during normal operation, as described below, in most contexts it is assumed that users will only be directly harmed by **Behavioral Information Gathering** if the attacker can learn several pieces of information at once. For example, to say that user X accessed website Y the attacker would need: buyer identification, website access information and several pieces of link identification. For this reason, peers following the reference specification do not store or share any of the above information except as required to provide the services a customer has purchased.

2.2. Economic Attacks

Unlike similar systems, Orchid Protocol must also concern itself with attacks on payment mechanisms. The taxonomy used in this paper is:

1. **Economic Exploits**. Profitable undesirable behavior such as a user providing "free sample" bandwidth allowing users to exclusively use free sample bandwidth.
2. **Economic Denial of Service (EDoS)**. Using payments to overwhelm another node on the Orchid Network with purchases, thereby taking them offline.

2.3. Quality of Service Attacks

Some adversaries may be satisfied by slowing down system performance of Orchid Network users generally, thereby potentially diminishing usage.

2.4. Man-in-the-Middle Attacks

Actions that can be performed only after inserting oneself between two interacting parties are collectively referred to as man-in-the-middle attacks. Encrypted information may be logged for analysis of metadata (Section 2.8), while non-encrypted data may additionally be changed to control behavior. If key exchange is not secured, the man-in-the-middle may also trick two parties into wrongly believing the attacker's key is the other party's key.

2.5. Sybil Attacks

Malicious actions, performed by pretending to be multiple users, are termed Sybil Attacks, after a patient suffering from multiple personality disorder. Applications of this type of attack include:

- Submitting multiple reviews to Yelp, Amazon, etc.
- Achieving faster downloads on BitTorrent by pretending to be multiple leachers[72].

2.6. Eclipse Attacks

In an Eclipse Attack, the attacker's goal is to hide part of a system from itself. The methods employed are generally the network equivalent of privilege escalation attacks: gain control of network positions which have more control of the network, then use that control to acquire more control.

- Segmenting the Bitcoin mining P2P network, allowing for so-called "51% attacks" when the attacker controls substantially less than 51% of the compute power[65].
- Removing a file from the BitTorrent DHT by taking over the address space associated with its magnet link[83].

2.7. Denial of Service Attacks

Attacks centered around taking a specific resource offline are termed Denial of Service Attacks (DoS). System behavior during "unexpected" circumstances is often poorly specified and tested. DoS attacks are useful for deanonymizing nodes in P2P networks. Notable examples:

- Targeted DoS attacks used in concert with Sybil Attack based monitoring to deanonymize Tor traffic[53].
- DoS off-lining for complete control of I2P's floodfill database, requiring only 20 Sybil nodes, thereby deanonyimizing all traffic on the network[64].

2.8. Inference Attacks

Deanonymization which stems from a statistical modeling of system behavior are termed Inference Attacks or Monitoring Attacks. These are often combined with "probing moves" such as carefully crafted or timed requests, or other attacks such as DoS-ing a specific peer off of the network and observing how traffic patterns respond.

- Inferring medical illnesses, family income, and investment choices of end-users from SSL encrypted web traffic[58].
- Deanonymizing Tor, I2P and Orchid traffic from global traffic logs[60].
- Learning the private key of an OpenSSL server through timing analysis[55].

2.9. Hacking

By converting historically trustworthy peers into attack vectors, motivated attackers might directly compromise nodes on the network. When bandwidth is deployed using Chains, iterative hacking may eventually allow an attacker to "backtrace" a connection. Such attacks have important security implications but are out of the scope of the Orchid Network. If the Orchid Network's design achieves its goals, this will be the main attack against users of the system.

3. Alternative Approaches

3.1. Unprotected Internet Access

Users who access the Internet without protection provide their complete browsing history and website use to their ISP, who can then share or sell that data.

3.2. Virtual Private Networking (VPN) Services

Virtual Private Networks (VPNs) use encryption to securely transport a VPN subscriber's traffic across a larger insecure network. Once the VPN has received the traffic, it is decrypted and retransmitted across a different large insecure network. The retransmission can assist users in circumventing access restrictions imposed by websites, and to a lesser extent, reduce the tracking of their website browsing habits. Encryption prevents the user's ISP from seeing their traffic, thereby preventing monitoring attacks. This is accomplished by making the VPN a new ISP for the user. Any attack an ISP could previously perform can easily be performed by the VPN provider.

VPN users should not assume their VPN provider is trustworthy. Although VPN service providers face more competition than ISPs, they ultimately draw talent from the same sources, and have similar bandwidth-for-cash-type business models. It is unlikely that VPN providers will not fall prey to the same incentives which led the user to not trust their ISP. Additionally, the re-use of IP addresses for relaying traffic in VPN setups enables relative ease in blocking their use by commercial websites[**13**].

3.3. Tor

Tor[61] is a free software project famous for introducing the idea of *Onion Routing* to a wider audience. In this system, users download a global list of relays and exit nodes, randomly select from that list, and form *onion routes* from their selection. Onion routes are an ordered list of relays; packets sent along an onion route are encrypted for each peer in turn, ensuring that each node must have received a packet enroute for it to be understood by the exit node. The result is that unless several nodes are compromised or run by the same user, no two relays know both who sent a packet and where it went.

4. External Libraries

Orchid's functionality is built on several important primitives. As some readers may not be familiar with these primitives, or be familiar with the specific properties used in the Orchid Network, we briefly summarize them here.

4.1. WebRTC

WebRTC[48] is a system originally designed to facilitate real-time communication between web browsers. It provides excellent implementations of NAT and firewall traversal methods, including STUN, ICE, TURN, and RTP-over-TCP. By selecting WebRTC as the basis for our networking protocol, rather than custom coded TCP and UDP networking code, we both get a world-class implementation of these technologies, and (to an extent) mask our user's traffic as general web traffic.

4.2. NACL

NaCL[49] (pronounced "salt") is a cryptography library by Daniel J. Bernstein et al., focused on building the core operations needed to build high-level cryptographic tools. It was chosen as the source for cryptographic primitives on this project due to both it and its author's sterling reputation. All cryptographic operations described below are implemented using NaCL, aside from Ethereum smart contract cryptographic code.

4.3. Ethereum

Ethereum[56] is a decentralized blockchain and platform that includes a native currency (ETH) and Turing-complete smart contracts. The smart contracts proved extremely useful for the design of Orchid, allowing us to offload a plethora of design concerns related to tracking payment balances and the verification and fairness of Orchid payment tickets.

5. Medallions

Fully decentralized, fully anonymous digital systems suffer from attacks in which a single malicious user pretends to be thousands of users (Sybil Attacks).

To combat this class of attack, the Orchid Protocol employs Medallions – data which demonstrates that a given public key was in possession of a sizable amount of computation at a given time. As computation is an expensive resource, the use of Medallions places budgetary limitations on a given attacker's ability to impersonate multiple users.

5.1. Medallion Specification

To produce a medallion, a peer takes a public key K, and the most recent Ethereum block hash E, then (iteratively or in parallel) locates a salt S such that $H(K, E, S) \geq N$, where N is some difficulty scaling factor.

Because it is key specific, it cannot be used to impersonate multiple public keys. Because it is tied to an Ethereum block hash, it cannot be precomputed.

5.2. Selection of Proof-Type

Readers who are familiar with other distributed market based networks will have recognized Medallions as being similar in premise to proof-of-work systems (bitcoin, etc), and may be inclined to ask: why not use proof-of-stake, proof-of-idle, or other less energetically wasteful methods for proving "realness"?

Proof-of-stake rests on the assumption that no attacker will ever control the majority of tokens. As our attack model includes oppressive governments, this can not be counted on. Even Bitcoin's astonishing market capitalization is far less than the GDP of a modestly sized country. Making matters more complicated, in the near future we intend to extend the system to support anonymous payments, which will make detection of such a "hostile takeover" much more difficult. In short: we did not use proof-of-stake because we did not want to engineer a system in which our users' right to privacy might be sold to the highest bidder.

Proof-of-space looks much more interesting. Although we are not sure that a suitable method will be located, we are exploring the possibility of using proof-of-space for an upcoming version of the Orchid Protocol. This would allow old smart phones, for example, to be installed by users in their homes as Relays and Proxies. For more on this idea, see Section 15.1.

Proof-of-idle rests on the additional assumption that periodic, synchronized proof-of-work is sufficient to demonstrate a User's share of the global computational power. Unfortunately, while the network is in its infancy ($\leq$ 10 million Peddlers), this leads to a situation where any company in control of a supercomputing center may, with only the sacrifice of 1% of their computational power, take control of the network. As we expect it to be quite a while before we have sufficient numbers of Peddlers for this attack to cease being devastating, we are not using proof-of-idle for this release.

6. Medallion Proof-of-Work

Medallions form the bridge between our core security assumptions and the network as a whole. Since our fundamental security goal is to limit a well-motivate attacker from gaining control of the Orchid Network, our choice of Medallion creation must meet the following conditions,

1. Medallion creation must be *easy* for a non-malicious node to create
2. Medallions must be *easy* to verify
3. Medallions must be *difficult* to create in bulk

With these conditions, we define *difficulty* to mean prohibitive scalability in time and money. In short, we want a proof-of-work system where it is *easy* for a normal node to obtain entry to the network but difficult for an attacker to scale entry into the network. We will discuss our choice of proof-of-work over other methods such as proof-of-stake and proof-of-space

Two primary methods currently exist that satisfy the requirements above: challenge-response protocols, and crypto-puzzles. Unfortunately, challenge-response protocols may not provide sufficient security within the Orchid model as an attacker may be able to precompute challenge and responses via collusion. This leaves crypto-puzzles of which there are many in existence today [51, 76] each with their own trade-offs. Again, in order to satisfy the requirements of Orchid only a subset of those crypto-puzzles are suitable. Namely, crypto-puzzles which can not easily be parallelized, made into an ASIC, or scaled trivially. Recently, researchers have discovered algorithms that produce easy-to-verify results that have tunable creation difficulty [51]. These collection of algorithms exploit the trend that memory and total silicon area is expensive to scale [46, 62]. These class of algorithms are called asymmetric memory-hard functions and we use them for medallion creation. There are several varieties of these functions [51, 73, 82] but we have chosen to use Equihash. Equihash is based on the k-XOR birthday problem and provides memory hardness via a time-space trade-off[1]. Since Equihash is tunable, simple, is based of an NP problem, and has gained acceptance in the cryptocurrency community, we believe that using such a function as our basis for proof-of-work provides an acceptable level of security and future-proofing.

To produce a medallion, a peer takes a public key K, and the previous Ethereum block hash E, then performs a series of computations in order to locate a salt S such that $F(K, E, S, ...) \geq N$, where N is some difficulty scaling factor. When a new Ethereum block is added to the chain, a new S must be calculated to keep the Medallion current.

7. Payments

7.1. Orchid Payment Requirements

Paying for bandwidth presents a rather unique set of challenges. In most other payment systems, the cost of an item is substantially greater than the cost of sending a packet, and so the networking cost may be safely ignored as just another transaction cost. In the Orchid Network however, the cost of a packet is the *price being paid*, and so even if the transaction costs for sending payment are as low as a single packet, they would be equal in cost to the purchased item.

We thus require transaction fees to be low enough that users can pay (automatically via the Orchid client) for arbitrarily amounts of relayed traffic, down to a single packet. Aside from low transaction fees, the payment mechanism must be granular enough that micro or even nano payments are possible. This creates a requirement not only on the efficiency of the payment mechanism but on the divisibility of the underlying item or verifiable record that is accepted as payment.

[1] Note that it is no coincidence that this time-space trade-off is reminiscent of time-memory trade-offs as first discovered by [67]

As the purpose of the Orchid Network is to get rid of Internet surveillance and censorship, other requirements of the payment mechanism include that it is uncensorable and anonymous and does not depend on trusted third parties. Even if the underlying network is resistant to surveillance and censorship, if the payment mechanism is not, then it becomes a failure point where users can be censored and tracked. Similarily, relying on trusted third parties would expose the Orchid Network to interference from state actors and other powerful entities whom can influence payment providers.

Thus, the requirements on Orchid Payments are:

1. $Unforgeability$, only the owner of the underlying item or verifiable record accepted as payment should be able to use it for payments.
2. *Availability*, meaning that no one can prevent a user from sending Orchid payments, and no one can prevent a recipient from receiving payments.
3. $Irreversibility$, it should be impossible, even for the sender of payments, to reverse past payments.
4. *Anonymity*, defined as unlinkability of senders and recipients, whether by account addresses, amounts or time. Ideally, anonymity should hold not only against malicious observers, but also if the sender or recipient is malicious.

In the following sections we will discuss potential solutions for payments, keeping these requirements in mind. We will argue that The Orchid Payments (section 7.12) fulfill all but the anonymity requirement. We continue with a discussion of extensions and improvements around payment anonymity, efficiency as well as making the scheme more non-interactive.

7.2. How Much Will A Packet Cost?

For the purposes of this discussion, let us assume that a packet is 1×10^3 bytes in length. To calculate an upper bound, we observe that one of the most expensive cloud services is Amazon Web Services's Singapore CloudFront, charging \$0.14 per 1×10^9 bytes. This yields a per-packet cost of 1.4×10^{-5} cents (\$0.00000014). Because bandwidth is a wasting good (any unsold bandwidth is lost forever), the actual price is likely to be significantly lower than this upper bound.

7.3. Traditional Payments

In current financial payment systems, transactions are settled through negotiations between two or more entities such as banks or payment service providers[2] using protocols such as ISO/IEC 7816[3] for payment cards and EBICS[4] for bank payments. Such protocols run on networks such as SWIFT[5] and NYCE[6] to support both national and international transactions. The entities forming these networks each maintain their own ledgers and continously update them from electronic payment receipts as well as manual reconcilitation[7].

Connecting to traditional payment networks typically requires special licenses in most jurisdictions as well as case-by-case business agreements between connecting entities. The resulting global financial network can be seen as an permissioned ad-hoc mesh of connecting businesses and a mix of protocols and networks. Each ledger represents a single point of failure, lacks cryptographic integrity and can be arbitrarily modified at the whims of the controlling business entity.

While classical payment protocols typically do not in of themselves define transaction fees, the entities running the protocols add fees on top. Per-transaction fees can range from a few cents for payment card transactions[8] up to \$75 for international wire transfers[9]. Many systems, instead or in addition, charge a percentage fee of the transacted amount, which can amount to as much as 13% for bank transfers[10] and 3.5% for payment cards[11].

As traditional payments depend on trusted parties, they are virtually impossible to use for the Orchid Network without sacrificing the properties we require. In particular, reversibility is present by design in the

form of reversal transactions[75]. Transactions are generally hard to forge, but credit card fraud is common and identity theft or hacking can lead to compromised user accounts. Moreover, these payment systems provide only partial availability, as they tend to malfunction at inconvenient times and suffer downtime on a regular basis. Anonymity is lacking as the trusted parties managing the payment typically have not only records of the sender, recipient, amount and time of payment but also often identity information about the sender. Finally, as we will see in the following sections, transaction fees on the order of traditional payments would be prohibitively expensive in the Orchid Network.

7.4. Blockchain Payments

Bitcoin revolutionized the status quo of traditional payment systems and continues to disrupt global markets for payments and international transfers. Bitcoin is a global network and protocol unaware of geographical boundaries. Applying public-key cryptography, transactions transfer bitcoin amounts between addresses generated by the users themselves, without the need for any trusted party. Users generate keypairs where a hash of the public key can be used as a payment address, requiring the private key to sign transfers from the address[12]. Bitcoin payments are unforgeable and irreversible[77] (within a reasonable time to account for block confirmations). The Bitcoin network has seen minimal downtime since its inception and other than unlikely active censorship by miners (discussed further in section 7.6) it can be seen as generally available. Bitcoin payments are pseudo-anonymous and the level of anonymity depends to a large extent on how the network is used[68].

In general, decentralized cryptocurrencies allow humans and computer systems alike, for the first time in history, to transact value without trusted third parties - a crucial requirement for incentivized, distributed overlay networks such as Orchid.

Transaction fees in Bitcoin are not determined by the transaction amount but rather by the size of the transaction data structure multiplied by a factor configured by the sender. Until 2017, average transaction fees remained well below $1, but in February of 2017 fees rapidly rose as the Bitcoin network reached maximum transaction capacity. Average fees rose[13] to as high as $8, rendering applications relying on low fees infeasible on the Bitcoin network.

The Ethereum network, also rooted in public-key cryptography and secured by proof-of-work like Bitcoin, derives the same properties of unforgeability, availability and (non-classic) irreversibility. Ethereum has a higher and dynamically adjustable transaction capacity, and the network has seen low fees since its launch in 2015. However, due to increased number of transactions as well as price growth of Ethereum's underlying native token, Ether, transaction fees (known as *gas*) have grown[14] to an average of $0.20 with peaks up to $1.00. Transactions executing smart contract code cost even more, in proportion to how much computation is performed.

The increase in transaction fees in popular, public blockchain networks inhibit their potential for handling micropayments directly, pushing micropayments to 2nd layer solutions such as payment channels.

7.5. Ethereum Transaction Costs

Ethereum smart contracts allow for the creation of sophisticated payment mechanisms, drawing on the power and flexibility of the Ethereum Virtual Machine[85] (EVM) which offers (within economic bounds) a Turing complete execution environment. Each instruction executed by Ethereum smart contracts add to the transaction fee of the originating transaction.

Each EVM instruction costs some amount of gas, and Ethereum transaction fees are defined as the total gas spent by the transaction multiplied by the gas price set by the sender. Miners select any valid transactions for inclusion in their mined blocks and can include transactions with any gas price, including zero. Selecting transactions with higher gas price may lead to more profit as each block has a limit on how many transactions can be included. Likewise, accepting a lower gas price may also lead to more profit as it can allow a miner to fill up their blocks if the network is not running at maximum capacity. This mechanism creates an

ever-changing yet stable game theoretic equilibrium which is tracked by sites such as the Ethereum Gas Station[15].

As of October, 2017, the cost of getting a transaction included with high probability within a few blocks is $0.026. For confirmation within 15 minutes, $0.006 suffices. These estimates are for the base cost of a transaction - 21,000 gas for a plain ether transfer without any smart contract code execution. If the transaction executes smart contract code, each EVM instruction adds an additional gas cost. For example, permanently storing a new 256 bit value in smart contract storage costs 20,000 gas and updating an existing value costs 5,000 gas.

As an Ethereum ERC20 ledger is simply a mapping of account addresses to balances, an ERC20 token transfer should cost on the order of 21,000 + 20,000 gas for new accounts, with subsequent transfers requiring 21,000 + 5,000 gas (as the recipient account then already has an entry in the token ledger). Observing live[16] ERC20 transactions we see the gas costs are a bit higher at approximately 52,000 and 37,000 gas for transfers to new and existing accounts, respectively. The difference accounts for smart contract code executing validations of invariants such as if the sender has sufficient balance as well as other implementation details such as the logging of payment receipts. 50,000 gas would require between $0.014 and $0.062 in transaction fee, depending on how fast we want the transaction confirmed.

7.6. The Orchid Token

The Orchid Network is using an Ethereum-based ERC20 token in order to satisfy the payment requirement of unforgeability, availability and irreversibility. The following sections discuss how we are able to lower transaction fees for ERC20 transfers to enable sending of arbitrarily small token amounts. Payment anonymity is discussed in section 7.17.

The Orchid Token (OCT) is used for payments within the Orchid Network. The Orchid Token is a new, Ethereum-based, ERC20-compatible, fixed-supply token. The supply is fixed at 1×10^8 tokens where each token has 1×10^{18} non-divisible subunits (same divisibility as Ether).

At first glance, the Orchid payment system detailed in the following sections can be configured to use Ether or any ERC20 token. In fact, using Ether would simplify the ticket contract, slightly reduce gas costs and improve usability as users would only need Ether rather than having to acquire both the Orchid Token and Ether (for transaction fees).

However, Ethereum is planning future protocol upgrades to allow transaction fees to be paid by arbitrary mechanisms, including ERC20 tokens [17] [18]. This will remove most of the drawbacks of using a new token; there will be no difference in gas cost and users only need to acquire a single token. It is also possible to set the gas price to zero and add an ERC20 token payment to the miner (using the EVM COINBASE[85] op code) in the contract execution [19]. This would require explicit support from miners as they would need to configure their mining strategy to accept zero gas price and validate that the transaction execution includes an ERC20 token transfer to the coinbase address.

However, the decision to introduce a new token instead of simply using Ether is for socioeconomical, not technical, reasons. By creating a new token and making it the only valid payment option in the Orchid Network, we engineer socioeconomic effects that we believe are significant enough to warrant the increased complexity.

7.6.1. Incentivization

Incentivization is a way to bootstrap new protocols and networks by giving people partial ownership of the network [20]. New decentralized networks such as Orchid suffer from the chicken and egg problem. The more proxy and relay nodes, the more utility the network provides for users. And the more users, the more valuable it becomes to run a proxy or relay node. By deploying a new network token, the network effect can be accelerated as all potential users are incentivized to use the network early on.

7.6.2. Decoupling

In decentralized systems built on other decentralized systems, new tokens decouple the market value of the new systems from the underlying system. For example, as of October, 2017, Ether has a market cap of approximately $30 Billion and daily, global trading volume on the order of $500 Million [21]. The price of Ether is affected by a variety of factors such as overall speculation of cryptocurrencies, hashing power of Ethereum miners and the success and failure of hundreds of projects built on Ethereum. However, the failure or success of a single project may not significantly affect the Ether price, but will have a dramatic impact on a token specific to the project in question. Decoupling market value using a new token creates a better indicator of the size and health of the project and system in question, and effectively creates a prediction market on the future of the system.

7.6.3. Liquid Market

A liquid market for a system-specific token can enable users heavily reliant on the system to hedge against the potential failure of the system by taking short positions. If this seem far fetched we should note that the original intention of financial derivatives was to allow businesses to hedge against unfortunate future events. With the advent of decentralized exchanges such as 0x[22] and etherdelta[23], and prediction markets such as Augur[24] and Gnosis[25], derivatives on Ethereum-based tokens and systems are not too far away. In fact, such derivatives can be even more effective[26] than traditional financial derivatives, as the former have no trusted party, are permissionless and potentially even anonymous.

7.6.4. New Tokens

New tokens also make it easier to engineer specific incentives for stakeholders; as the tokens exclusively derive their value from the new system, they act as powerful incentives for anyone working towards the success of the system. Ethereum smart contracts can implement autonomous locking of tokens to ensure that token holders can only access their tokens according to a defined schedule. This aligns incentives over time and puts the focus of token holders on the long-term success of the *system* rather than social structures such as specific teams or associated corporations. If the Orchid Network simply used Ether, and stakeholders received a lockup of Ether, they would actually be more incentivized to work towards the overall success of Ethereum rather than towards any specific system making use of Ethereum. It can be argued that such an outcome would not be an optimal incentive alignment for the Orchid Network and project.

7.7. Ethereum Censorship Resistance

Similar to most public blockchain networks, Ethereum transactions cannot be censored unless the validator (miners in the Ethereum network) chooses to not include them in their created blocks. As blocks are mined randomly among all miners, proportionally to hash power, it would require the vast majority of miners to actively censor Orchid payments to significantly disrupt the Orchid Network. For example, even if 90% of the hash power chooses to not include Orchid related transactions, the Orchid Network would still function with the only caveat that transactions would take, on average, ten times longer to confirm. A more severe form of censorship would be if a large group of miners, say 51%, chooses to censor Orchid related transactions by rejecting blocks including them [71]. This is valid according to the Ethereum protocol rules and effectively creates a soft-fork. However, organizing large-scale miner collusion to create such a soft-fork comes with great risk of loss of profit; if the soft-fork fails to achieve sufficient hashing power the colluding miners would miss out on their block rewards. Aside from the profit risk, we consider this possibility extremely unlikely given the decentralized nature of Ethereum miners and the lack of legal and regulatory limitations on blockchain mining strategies.

7.8. Building Micropayments from Macropayments

With transaction costs and choice of payment token now discussed, let us now look at viable payment methods. One fundamental challenge with blockchain-based micro-payments is how to avoid transaction fees. Imagine we want to send a single cent a large number of times, if we send each cent as a plain Ethereum ERC20 transaction, we would pay 1.4 cents - 140% in transaction fee for each payment! Effective micro-payments requires lowering transaction fees by several orders of magnitude.

One potentially interesting approach, which was employed in MojoNation[27], is to have a "balance of trade" between each pair of nodes. As bandwidth flows between them, they periodically settle up when the balance gets too far from zero. However, as we have seen, the transaction costs of settling payments using plain Ethereum transactions would result in at minimum a $0.014 transaction fee. We can see this price equals around 140 megabytes of bandwidth, based on the previously discussed upper bound. A secondary issue with this approach, is that peers nearing the reconciliation threshold would know that fact, and be tempted to disconnect and create a new identity rather than pay the fee.

7.9. Payment Channels

A popular technique in blockchain applications first seen on the Bitcoin network is payment channels [28]. Partially described by Satoshi Nakamoto[66] and later defined and implemented by Hearn and Spilman[29], payment channels were later studied by Poon and Dryja[30] for the Bitcoin lightning network. Payment channels allow a sender and recipient to send an arbitrary amount of transactions between each other and only pay transaction fees for two transactions - one to setup the payment channel and one to close it. This is accomplished by first having the sender post a transaction that locks up some amount of tokens that can only ever be sent to either the recipient or back to the sender. Typically, the tokens can only be sent back to the sender at some future time T. Meanwhile, the tokens can be (incrementally or in full) sent to the recipient. The sender continously signs transactions spending a larger and larger amount of the tokens to the recipient, and sends them directly to the recipient without posting on the blockchain. The recipient can at any time until T post their last received transaction to claim the aggregated amount sent to them.

Payment channels provide an efficient way for a sender to provide a recipient with cryptographic proof of continous payments. Since the intermediate payments do not incur any transaction fee, they can pay arbitrarily small amounts and be sent arbitrarily often. In practice the bottleneck becomes the computational overhead of verifying transactions as well as the bandwidth requirement of sending them.

While payment channels effectively provide constant complexity of transaction fees for arbitrary amounts of intermediate payments, they are not efficient enough for all use cases. In particular, in systems with large amounts of senders and recipients that often change with whom they interact, the constant creation of new payment channels may prove too expensive. Likewise, for very small or short lived services provided - such as a single HTTP request or 10 seconds of video streaming - the transaction fees of the required on-chain transactions can be too costly.

7.10. Probabilistic Payments

If we cannot challenge the assumption that payment settlement must happen on a blockchain with transaction fees, the theoretical minimum cost is the cost of a single transaction - as blockchains require at least one transaction to execute a state transition. To settle some amount of (micro) payments, we thus need at least one transaction.

What if we could do away with the setup transaction required by payment channels, and still be able to prove to a recipient that they are being paid?

Fortunately, there is a similar, solved problem, in the blockchain industry: mining pool shares[31]. As the proof-of-work difficulty increased in networks such as Bitcoin, miners began pooling their computational power to avoid high variance where it could take years for a single miner to find a block solution. Mining

pools award rewards in proportion to hashing power, and individual miners prove their hashing power by continously sending solutions[32] for the same underlying block hash but at a lower difficulty. This technique enables mining pools to cryptographically verify the hashing power of each pool member - regardless of whether that pool member finds a solution satisfying the actual proof-of-work target.

If we apply the same thinking to payment channels we can construct probablistic payment schemes where the sender continously proves to the recipient that they are being paid *on average*, regardless of whether an actual payment takes place. This allows us to create probabilistic micropayments where no setup transaction is needed, and the recipient only needs to pay a transaction fee when "cashing in".

Before we look at how we can construct such probablistic micropayments using Ethereum smart contracts, let's take a step back and observe that the original idea of probabilistic payments predate blockchain technology and was first published by David Wheeler[84] in 1996. Wheeler describes the core idea of probabilistic payments and how it can be applied to electronic protocols using random number commitments in such a way that neither the sender not the recipient (buyer and seller in the paper's terminology) can manipulate the outcome of the probablistic event, while still proving to each other what the probability and the winning amount is.

Several papers followed up on Wheeler's idea and in 1997 Ronald Rivest[79] published a paper describing how to apply probabilistic payments in electronic micropayments. In 2015 Pass and Shelat[78] described how to apply probablistic micropayments to decentralized currencies such as Bitcoin, noting that prior schemes all relied on trusted third parties. The following year Chiesa, Green, Liu, Miao, Miers and Mishra[59] extended this research to work with zero knowledge proofs, providing decentralized and anonymous micropayments applicable to cryptocurrency protocols.

Given the interest in and prevalence of payment channels in recent Ethereum-based systems, it can be valuable to view probablistic payments from the perspective of payment channels. In exchange for omitting the first setup transaction, we lose the ability to guarantee sending of exact amounts, achieving instead only a probablistic guarantee. However, we will show that by tuning the probability, winning amount and frequency of payments we can make probablistic micropayments so granular that they can replace payment channels for several classes of blockchain-based applications with no significant drawbacks.

Essentially, as we can do away with the initial setup transaction, we gain the ability to, from the same sender account, pay for arbitrarily small service sessions to an arbitrary number of recipients while still proving to each of them the exact probability of the payment amount. Assuming the service provider (a relay or proxy node in the Orchid Network) provides a sufficient amount of service, the variance of the probablistic payouts will quickly even out.

7.11. Blockchain-Based Probablistic Micropayments

To easier convey the core idea of how probablistic payments can be applied to blockchain protocols, we will here gloss over several details. A formal description of the MICROPAY1 scheme is available in the cited original paper, and the Orchid probablistic payment scheme is formalized in section 7.12

Pass and Shelat describes MICROPAY1[78], where digital signatures and a commitment scheme are combined to engineer release conditions including a random outcome of an exact probability. The sender first makes a "deposit" by transfering bitcoin to an escrow address of a newly generated key. Then, the recipient (merchant in MICROPAY1 terms) picks a random number and sends a commitment over this number to the sender. Alongside the commitment the recipient also provide a new Bitcoin address. The sender also picks a random number and signs the concatenation of this number (in plaintext), the commitment from the recipient and other payment data such as the payment destination address provided by the recipient.

Verification of the resulting ticket involves checking that the recipient commitment matches the number they reveal, as well as verifying the signature from the sender matches the address of the bitcoin deposit. If the last two digits of XOR of the random numbers from the sender and recipient are 00 then the ticket is a win, and can be spent by the recipient.

Intuitively, we can think of the "coin toss" in this scheme as unbiased unless the sender can break the binding property of the commitment (or forge a signature), or if the user can break the hiding property of the commitment.

Note that the sender can "double spend" their deposit by issuing tickets to multiple recipients in parallel or front-run the recipient by broadcasting a spend when a ticket claim is seen from the recipient. The authors of MICROPAY1 discuss how this can be resolved by a "penalty escrow", a second amount deposited by the sender that can be spent back to the sender at some future time and until then "slashed" or "burned" by anyone who can submit two valid tickets for the same payment escrow. This prevents the sender colluding with the recipient or acting as it's own recipient.

The authors of MICROPAY1 construct iterative improvements in MICROPAY2, and MICROPAY3 where a trusted party is introduced to perform some computational validation steps on the ticket and release a signature if the computations are correct.

7.12. Orchid Payment Scheme

Now that we have located a suitable abstraction for our payments, the question becomes: how should they be implemented?

Alongside the requirements discussed in section 7.1 we also want to satisfy:

- *Reusability*, the method for constructing each new ticket must not require new transaction fees or new on-chain transactions for each ticket, as otherwise transaction fees will once again be an issue.
- Double spending must be prevented, or failing that not profitable.
- The system must be sufficiently performant in terms of computational cost so as not to overwhelm the cost of a packet.

Of those requirements, the last element is perhaps the most troublesome. To the best of our knowledge, no method for constructing lottery tickets based on Ethereum tokens exists which do not require computation on the order of verifying an ECDSA signature. As detailed in this section, this follows from the requirement of the sender to cryptographically prove to the recipient not only the ticket amount and probability of winning, but also that the sender's Ethereum account has a sufficient amount of Orchid Tokens locked up for the purpose of sending tickets.

For this reason, although it was not sufficient for use alone, we are forced to employ a balance-of-trade approach similar to the one mentioned above. This in turn leads to a new requirement, namely "the balance of trade must be kept sufficiently small so as to not cause an incentive to disconnect during trade". As this is a mechanism design issue caused by an implementation reality, let us for now focus on implementation by assuming a solution exists, and defer further discussion until section 7.15.

The Orchid payment scheme is a pseudo-anonymous, probablistic micropayment scheme inspired by MICROPAY1 and related constructs. It mitigates front-running and parallel (including double) spending attacks without the need for a trusted party by leveraging Ethereum smart contracts and slashable penalty deposits. The pseudo-anonymity of Orchid payments is equivalent to what can be achieved in regular Ethereum transactions (although Orchid clients employ additional privacy techniques such as one-time addresses and key separation between node identities and payment addresses to achieve limited anonymity).

The trusted party introduced in MICROPAY2 and MICROPAY3 can effectively be replaced by Ethereum smart contract code. The EVM allows to implement arbitrary logic (within economic bounds on the computation) for validating micropayment tickets, and provides primitives[85] for the ECDSA[70] recovery operation as well as cryptographic hash functions.

7.12.1. Payment Ticket Definitions

Orchid payment tickets have the following fields:

H (function)	– cryptographic hash function (more details in section 7.12.2)
$timestamp$ (uint32)	– Unix time denoting when the ticket value begins decreasing exponentially
$rand$ (uint256)	– random integer chosen by $recipient$
$nonce$ (uint256)	– random integer chosen by the ticket sender
$faceValue$ (uint256)	– value of a winning ticket
$minValueMarket$ (uint256)	– expected value of a ticket based on the bandwidth market
$minValueAccepted$ (uint256)	– expected value of a ticket based on what the a recipient accepts
$winProb$ (uint256)	– probability that a particular ticket wins $faceValue$ from the sender
$recipient$ (uint160)	– 160-bit Ethereum account address of the ticket recipient
$randHash$ (uint256)	– digest of $H(rand)$
$ticketHash$ (uint256)	– digest of $H(randHash, recipient, faceValue, winProb, nonce)$
$(v1, r1, s1)$ (tuple)	– ECDSA signature elements of the ticket sender
$(v2, r2, s2)$ (tuple)	– ECDSA signature elements of $recipient$

7.12.2. Payment Ticket Cryptographic Choices

In order to reduce the cost of Orchid micropayments, we have chosen certain cryptographic functions over others due to their reduced Ethereum gas costs compared to other arbitrary functions.

H	– Keccak-256 – chosen over other hashes due to having the lowest gas cost (36 gas[85] for hashing 32 bytes) of all hash functions available in the EVM
$ECDSA$	– secp256k1 with Keccak-256, chosen over other curves due to the EVM support for ECDSA recovery of this curve as well as compatibility with existing blockchain software libraries and tools

7.12.3. Payment Ticket Generation

Let *Alice* be a recipient and *Bob* be a sender,

1. *Alice* picks a random 256-bit number, $rand$, calculates $randHash$, and sends the digest to *Bob*
2. *Bob* chooses values for $(nonce, faceValue, winProb, recipient)$ [2]
3. *Bob* calculates $ticketHash$
4. *Bob* calculates Sig(PrivKey, $ticketHash$)
5. The resulting ticket consists of:
   (a) $randHash$
   (b) $recipient$
   (c) $faceValue$
   (d) $winProb$

[2] Using information of this specification such as: general bandwidth market data and public capabilities signed by $recipient$

(e) *nonce*

(f) $ticketHash$

(g) *creator* (address of the sender key that signed $ticketHash$)

(h) *creatorSig* (the sender's signature over $ticketHash$)

Note that while this ticket is valid in the sense that the recipient can fully verify it, the recipient needs to sign it (see below) in order to be able to claim it in the Orchid Payment Ethereum smart contract.

7.12.4. Payment Ticket Verification

`Alice` (bandwidth seller) will then perform the following operations,

Verify:

(a) $randHash = H(rand)$

(b) $faceValue \geq minValueMarket$

(c) $winProb \geq minValueAccepted$

(d) $recipient$ = {the Ethereum account address published by the recipient}

(e) $creator$ = {the Ethereum account address published by the sender}

Validate:

(a) Validate: *creatorSig* is a signature by the private key who's public key is the creator address

Check:

(a) Validate: *creator* has sufficient Orchid Tokens locked in the Orchid Payment smart contract

Ticket is now proven to be valid, and may be a winning ticket

7.12.5. Claiming Payment from a Ticket

While the recipient can locally fully verify whether a ticket is valid and if it's a winning ticket, the actual payout of tokens in winning tickets is done by a Orchid Payment smart contract.

This smart contract exposes a Solidity API that takes as input:

1. *rand*
2. *nonce*
3. $faceValue$
4. $winProb$
5. *receipient*
6. *recipientSig* (*recipient* signature over $ticketHash$)
7. *creatorSig* (the sender's signature over $ticketHash$)

7.12.6. The Smart Contract Executes

Suppose Alice is a user who wishes to buy bandwidth. Alice must have an Ethereum account address, *addressAlice*, and Orchid tokens. Note that this address will have an associated public key, $PubKeyAlice$. Alice must also have Orchid tokens *locked up* in an Ethereum smart contract as defined in previous sections and locked with $PubKeyAlice$. In the previous section, Alice's address would be the Ethereum account address equaling the public key recovered from $creatorSig$ over $ticketHash$.

Let SLASH be a temporary boolean value which is set to FALSE and $PubKey$ be the public key recovered from $recipientSig$ over $ticketHash$,

Calculate:

(a) $ticketHash$

Verify:

(a) $randHash$; If not, abort execution[3].

(b) $PubKey = recipient$ address; If not, abort execution.

(c) *addressAlice* has Orchid Tokens locked up in the penalty escrow account. If not, abort execution.

(d) *addressAlice* has enough Orchid Tokens locked up in it's ticket account to pay for the ticket. If not, set SLASH to TRUE and continue execution.

(e) $H(ticketHash, rand)^4 \leq winProb$. If not, abort execution.

Determine:

(a) If SLASH = FALSE, then the ticket is paid out: $faceValue$ is transferred from the creator's ticket funds to *recipient*.

(b) If SLASH = TRUE, then creator is slashed.

Settlement:

(a) Send creator's ticket funds, if any, to *recipient* (this is from prior validations guaranteed to be less than $faceValue$).

(b) Set creator's penalty escrow account to zero (burns / slashes those tokens).

Note that while the slashing of the penalty escrow prevents double-spending by creating a disincentive for the ticket sender where they will lose more than they can gain from a potential double spend, there is still a danger of a ticket sender over-spending on a grand scale. To address this, the value of winning lottery tickets should begin to decrease exponentially at *timestamp*, thereby providing a strong incentive for winners to cash in immediately. This immediacy can be used by the recipient to calculate the "wasting rate" of the sender's Orchid token balance.

7.13. Orchid Gas Costs

We have measured a gas cost of approximately 87,000 from a solidity prototype implementation of the above scheme. This cost is for the full execution of the API for ticket claiming, when called with a winning ticket as input. The ticket claiming execution includes a sub call to the Orchid ERC20 ledger $transfer$ API. The solidity implementation of all Orchid smart contracts will be open sourced after cryptographic reviews and a minimum of external security auditing.

[3]Since the transaction was aborted, not Ethereum state transition occurs and no gas is spent

[4]interpreted as a uint256

7.14. Verifiable Random Functions

The payment tickets described in the prior section can be made less interactive by replacing the recipient's random number commitment with a verifiable random function (VRF). First published in 1999 by Micali, Rabin and Vadhan[80], a IETF draft for VRFs was recently proposed by Goldberg and Papadopoulos[33]. This draft specifies two VRF constructions, one using RSA and one using Elliptic Curves (EC-VRF).

Using a VRF, a sender of Orchid Payment tickets would be able to create tickets without the need of a per-ticket (or per-ticket until a winning ticket is encountered) commitment from the recipient. Rather, the sender only needs to know a public key of the recipient. The sender would replace the random number hash in the previously described ticket scheme with this public key. For efficiency, this could be the recipient public key for receiving funds that is already present in the ticket, but to adhere to the cryptographic principle of key separation, a second key may be required.

However, verifying an EC-VRF in the Orchid Payments smart contract would require explicit EVM acceleration of Elliptic Curve operations, as implementing them directly in solidity or EVM assembly would be prohibitively expensive in terms of gas cost.

Fortunately, in the Ethereum Byzantium[34] release, the Ethereum network added EVM support for Elliptic Curve scalar addition and multiplication[35] as well as pairing checks[36] for the alt_bn128 curve[50]. The EC-VRF construction is defined for any Elliptic Curve, and the IETF draft specifically defines EC-VRF-P256-SHA256 as the EC-VRF ciphersuite (where P256 is the NIST-P256 curve[54]). However, there appear to be no reason why the alt_bn128 curve could not be used instead while still achieving a sufficient security level. Also, SHA256 could be replaced with Keccak-256. This would allow VRF verification in an Ethereum smart contract and thus integration with the Orchid Payments smart contract.

However, while the alt_bn128 curve is used in zcash, it is a much more recent curve compared to P256, and not as well studied. Perhaps more significant is that the EC-VRF construction is an early draft pending review, and the EVM Byzantium upgrade is occuring at the time of writing this paper and have not yet been proven in live system handling significant value. Using an EC-VRF in the Orchid probablistic micropayments is thus not immediately feasible and the Orchid Project will aim to conduct further research as to the feasibility of using e.g. an EC-VRF-ALTBN128-KECCAK256 construction that can be verified by the EVM.

7.15. Balance of Trade

As mentioned earlier, the realities of symmetric encryption performance prevent us from sending payment with every packet, and so we need a good understanding of the risks inherent in employing a "balance of trade" approach. We do so here in a general setting: imagine Alice and Bob wish to transact in a fully anonymous manner. Bob is to perform some task for which he charges x, and Alice is to pay him once every y tasks. Unfortunately, the nature of anonymity is such that without prior transactions, Alice and Bob have no mechanism to trust one another. Can they cooperate?

If there is some setup cost to Alice and Bob's relationship (S_{Alice}, S_{Bob} s.t. $S_{Alice} > xy, S_{Bob} > xy$), the answer is yes: running away with the money or work ceases to be economically rational, unless (1) the total amount of work Alice was seeking was $\leq xy$ or (2) the total amount of work that Bob can perform is $\leq xy$. As we will see in our discussion of the Orchid Market (Section 11), setup costs exist on the Orchid Network which support trade imbalances in excess of 1×10^3 packets. Because sellers in the Orchid Market generally pay a higher setup cost than buyers, and because Customers asymmetrically know how much work they will require, the Orchid Network has Customers pre-pay.

7.16. Improvements

7.17. Anonymity

The Orchid payments discussed in prior sections are as pseudo-anonymous as regular Ethereum transactions are; all transactions are public including the amount and the sender and recipient accounts. The Orchid Client aims to improve on the default pseudo anonymity of public blockchain transactions by modern wallet techniques such as one-time addresses [37] and use of HD wallets [38] to provide unlinkability of payment addresses despite using a single root key.

With the Ethereum Byzantium release, it is now possible to implement linkable ring signatures with reasonable gas costs by leveraging the new EVM primitives for Elliptic Curve operations [39]. Combining Ethereum smart contracts with stealth addresses such as those provided by HD wallets and linkable ring signatures enables a class of mixing technologies such as the Möbius[74] mixing service. Möbius provides strong anonymity guarantees that are cryptographically proven using a game-based security model for mixing services. However, unlike prior mixing technologies, it provides anonymity against malicious observers and senders but not against malicious recipients. Combining services such as Möbius with Orchid probablistic micropayments brings us closer to our final requirement on payments - anonymity.

To achieve full anonymity guarantees against any malicious actor, whether observer, sender or recipient of payments, we have to look at zero knowledge technology.

zk-SNARK[47] technology as applied in the zcash network [40] can provide stronger anonymity guarantees compared to ring signatures. In zcash, transactions between shielded addresses provide full anonymity of the sender, recipient and amount.

7.18. Non-interactive

In section 7.14, we show that by replacing the random number commitment in the Orchid Payment Scheme with a VRF makes the scheme more non-interactive by removing the communication steps associated with the random number commitment. Instead of the recipient having to communicate the commitment to the sender before the sender can construct tickets, the sender would be able to immediately construct tickets from only public recipient information.

Each recipient would generate a new keypair specifically for the VRF and publish the public key alongside other public recipient information detailed in section 11.1. The sender would simply configure this public key in the ticket, and the recipient would sign received tickets with the corresponding private key. The ticket verification logic defined in section 7.12.4 would interpret the recipient VRF signature as the value that it compares to the winning probability threshold.

As discussed in section 7.14, while this would be a relatively simple modification to the payment scheme, the feasibility of VRF verification in the EVM requires further research.

7.19. Performance

While the Orchid Payments smart contracts are immutable they can effectively be upgraded by deploying new contracts and upgrading Orchid client software to point to them (while remaining backward compatible to old contracts if needed). Ethereum smart contracts support a multitide of optimizations to reduce gas costs and we anticipate future versions of the Orchid Payments smart contract to use e.g. inline solidity assembly [41] to optimize gas cost similar to how regular software systems often replace expensive subroutines with inline assembly.

However, the bottlenecks in verification of Orchid payment tickets are the cryptographic operations such as ECDSA recovery and the state updates of the sender and recipient entries in the Orchid token ledger. There, one improvement could be to dual use the recipient signature of the Ethereum transaction carrying the smart

contrat API call payload as the signature covering the ticket data structure. Currently, the Orchid scheme defines two signatures there for reasons of flexibility in the Orchid Client and to make it easier to specify and reason about the payment scheme without relying on Ethereum specifics. More simpler optimizations include tightly packing ticket fields and encode multiple internal variables in single 256 bit words to align with the EVM stack words and permanent contract storage slots, both being 256 bit in size.

On the other hand, to achieve greater anonymity, optional or even mandatory use of mixing techniques could, perhaps substantially, increase the gas cost of Orchid Payments. Using mixing services based on linkable ring signatures could easily lead to roughly an order of magnitude higher transaction fees[39]. However, users may find this worthwhile if it provides strong anonymity guarantees. As we can easily tune the probablistic variables of Orchid payments - ticket frequency, winning probability and winning amount - we can tune the average time between ticket claims to reduce transaction fees (especially for long-running nodes, who may only care to be paid on average once every few days).

Finally, an extremely interesting property of zero-knowledge technology such as zk-SNARKs is to dramatically reduce the computational overhead of arbitrary computation, such as Ethereum smart contract execution[42]. While generation of zk-SNARK proofs is expensive, the verification is cheaper - even compared to the original code. Since only the verification needs to be executed on-chain, a zero-knowledge proof of claiming an Orchid Ticket could be made cheaper to verify than the original verification code.

Going further, recursive SNARKs[52] have the potential to aggregate a set of SNARK proofs into a single proof. While they may be more applicable for blockchain consensus protocols[43], they may also be useful for Orchid to e.g. batch multiple ticket claims into a single smart contract transaction while avoiding linear gas cost complexity.

8. Bandwidth Mining

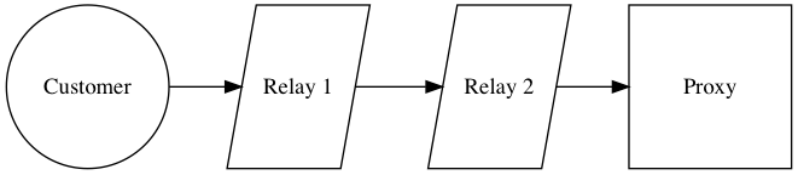

Figure 2: A three-Peddler Chain routing traffic for a Customer

In this section we will describe the specification for Relay and Proxy behavior, and discuss the "chaining together" of these nodes to support uncensorable, anonymous web browsing.

8.1. Specification for the Sale of Bandwidth

Relay nodes implement a relatively simple behavior pattern:

- Maintain one or more connections, each with their own encryption key.
- Check any tickets received, and cash-in winners.
- Monitor the balance of trade, and disconnect if it exceeds a predeclared amount.
- Receive data from any open connection, and perform decryption at message boundaries.
- Process decrypted messages as follows:
  - Forward any non-control segments to the connection(s) specified in the message.
  - Process any control segments:

* *Dummy Data.* Instructs the Relay to discard this segment.
* *Burn at Rate.* Instructs the Relay to send data over a connection at a fixed rate, queueing packets and generating data as necessary to maintain the rate.
* *Ratchet Ticket.* Instructs the Relay to pass a Ticket to the peer it received this packet from.
* *Initiate Connection.* Instructs the Relay to establish a new connection. Used during setup and to handle disconnection.
* *Initial Web Connection.* (Proxies Only.) Instructs the proxy to open an SSL connection to the specified host. To support whitelists, this cannot be a raw IP address.

An important consideration in the above behavior is that no proof-of-work is required of Relays on an ongoing basis. When combined with all our connections being WebRTC connections, this leaves the door open for websites potentially monetizing their visitors by running pure javascript relay code.

For discussion of possible extensions via application specific control segments, see Section 15.

8.2. Guard Nodes and "Bandwidth Burning"

The relay that a customer is connected to has a very important piece of information: the customer's IP address. We assume customers will want to keep this as private as possible, and so the default client expresses a preference for long-lived peers as the first hop.

Another concern for nodes at the first hop, which is discussed in depth in our discussion of informational attacks stemming from collusion (Section 9.2), is that they sit in an ideal position to perform timing attacks. To prevent these attacks, we recommend that privacy-conscious users employ a method called *Bandwidth Burning* – paying the second hop to send a fixed amount of bandwidth to the customer. As this approach results in data-usage which is completely uncorrelated to network usage, this approach prevents timing attacks performed by adversaries which cannot see the inbound traffic of relay three.

To provide assistance to users seeking evasion (Section 12), bandwidth burning will also support non-fixed rates determined by the statistical properties of popular non-Orchid WebRTC protocols.

8.3. Chaining

Customers interested in employing Relays for anonymous Internet access will use the above specification to create "chains" of relays.

9. Collusion Attacks on Chains

In this section we explore what kinds of information may be inferred or deduced by an attacker controlling or monitoring multiple relays and/or Internet Service Providers (ISPs). Using the assumption that Relays and Proxies are selected randomly (and consequently so were ISPs), we build a probability model of the chance that a given attack may be performed at different chain lengths.

9.1. Information Available to Individual Relays and Proxys

Due to the inherent structure of IP-based networking, and the Orchid protocol's use of Ethereum based payments, Relay and Proxy nodes *and their IPSs* gain access to the following information:

- The IP addresses of all computers they are connected to.
- The size, timing, and number of packets they forward.

- The public key which controls the tokens paying them.
- The contents of any control segments directed to them.

Additionally, Proxy nodes *and their ISPs* gain access to the following information:

- The hostname of the webserver, and the plantext portions of the SSL/TLS session negotiation.

9.2. Potential Parties to a Collusion

The following roles have access to customer information, and so might meaningfully collude or be monitored as part of an attack:

- The Internet Service Provider (ISP) of a customer, relay, proxy, or webserver. Untrustworthy with probability s.
- Website. The webserver the proxy is connected to. Untrustworthy with probability w.
- Relay_n. The nth relay in the chain. Untrustworthy with probability $\frac{r}{n}$.
- Proxy. The proxy relaying bandwidth to the webserver. Untrustworthy with probability $\frac{x}{n}$.

We have separated out r and x above because although an attacker cannot control the total amount of computation they have available for proof-of-work computations, they can control how that computation is allocated between relay and proxy nodes.

9.3. Types of Attack

The central goal of collusion attacks is the linking of a specific Orchid customer with a specific SSL connection. There are two ways this can be done:

- Relation. When this is possible, the attacker can deduce that a customer is talking to a given website because they can observe enough points along the route.
- Timing. When this is possible, the attacker can infer that a customer is talking to a given website by controlling and then observing the timing of packets.
- Unburning. When this is possible, the attacker can perform a timing attack in spite of bandwidth burning being employed by the customer.

9.4. "Regular" Internet Access: Zero Relays, Zero Proxies

Although the Orchid system is of course not used when a Customer directly connects to a website, we feel it is important to review what informational risk are present in this setup to ground the rest of our analysis.

ISP	Website	P(Relate)	P(Timing)	P(Unburn)
x		s		
	x	w		

In the above table, an "X" indicates participation in a collusion, and the values in P(Relate) and P(Timing) indicates the chance of this happening. Lines where attacks are not possible are omitted, as are lines with extraneous "X"s, and mention of more sophisticated attacks where simpler attacks are possible.

9.5. VPN: Zero Relays, Zero Proxies

For the purposes of grounding our analysis, we also present the collusion risk inherent to VPN access.

ISP	VPN	Website	P(Relate)	P(Timing)	P(Unburn)
	x		g		
x		x	sw		

Where g is the chance the VPN provider is being monitored, or is colluding with an adversary. Note that g may change over time in difficult to model ways, for example as a result of your VPN usage.

9.6. Zero Relays, One Proxy

ISP	Proxy	Website	P(Relate)	P(Timing)	P(Unburn)
	x		$\frac{x}{n}$		
x		x	sw		

It should come as no surprise that the risks in this case are quite similar to those of VPN usage. A Chain employing no relays is equivalent to a VPN in which a new VPN provider is selected at random before each browsing session, and no personal information is stored by the VPN provider.

9.7. One Relay, One Proxy

ISP	Relay_1	Proxy	Website	P(Relate)	P(Timing)	P(Unburn)
	x	x		$(\frac{rx}{n^2})$		
	x		x	$w(\frac{r}{n})$		
x		x		$s(\frac{x}{n})$		
x			x		sw	

If bandwidth burning is employed in this configuration, all timing attacks are mitigated. Observe that adding Relay_1 or the Proxy to the Timing case allows for a Relation.

9.8. Two Relays, One Proxy

ISP	Relay_1	Relay_2	Proxy	Website	P(Relate)	P(Timing)	P(Unburn)
	x		x		$(\frac{rx}{n^2})$		
x		x	x		$s(\frac{rx}{n^2})$		
	x	x		x	$w(\frac{r}{n})^2$		
x		x		x	$sw(\frac{r}{n})$		
	x			x		$s(\frac{r}{n})$	
x				x		sw	

If bandwidth burning is employed in this configuration, all timing attacks are mitigated. In the case of a timing attack carried out by Relay_1 and the Website, adding Relay_2 or the Proxy to the collusion results in a relation. In the case of the customer's ISP colluding with the Website, adding Relay_2 results in a relation.

9.9. Three Relays, One Proxy

ISP	Relay_1	Relay_2	Relay_3	Proxy	Website	P(Relate)	P(Timing)	P(Unburn)
	x	x		x		$(\frac{r^2x}{n^3})$		
	x		x	x		$(\frac{r^2x}{n^3})$		
x		x		x		$s(\frac{rx}{n^2})$		
	x		x		x	$w(\frac{r}{n})^2$		
x		x	x		x	$sw(\frac{r}{n})^2$		
	x				x		$s(\frac{r}{n})$	
x					x		sw	
	x	x			x			$s(\frac{r}{n})^2$
x		x			x			$sw(\frac{r}{n})$

10. SSL and TLS Vulnerabilities

SSL and TLS are complicated protocols, receiving a constant stream of security updates as implementation flaws are discovered. Unfortunately, users sometimes delay upgrading their software, use untrustworthy or poorly written software, and misconfigure their software. To protect users as possible, the Orchid Protocol provides "sanity check" features.

10.1. SSL Downgrade Attacks

In so-called *SSL Downgrade Attacks*, the attacker causes a secure connection to use poor quality encryption ([44]). To perform this attack, the attacker simply removes mention of more secure encryption methods supported by the client from the initial key negotiation packets. To prevent this attack, the Orchid Client automatically does the inverse where possible – it removes mention of insecure options from the key negotiation packet (an "SSL upgrade" attack.)

10.2. Old Browsers and Phone Apps

SSL and TLS security vulnerabilities are periodically found and patched in web browsers. However, not all users can be assumed to use up-to-date browsers. A similar situation occurs with mobile phone apps, where developers sometimes omit things like SSL certificate validation.

To address these issues, the Orchid Client automatically verifies certificate chains using an up-to-date copy of "Boring SSL" – the open source SSL library used in Google Chrome.

11. The Orchid Market

The Orchid Market is a P2P network similar in structure to a Distributed Hash Table (DHT), which serves as a meeting ground for buyers and sellers of bandwidth.

11.1. Fundamental Market Operations

At a high-level, the operations provided by the Orchid Market are:

- A method for Peddlers to join the Orchid Market.
- A method for asking Peddlers what services they have for sale.

- A method for selecting a subset of all peers, randomly weighted by computational resources, such that the *lookup property* holds:

$$lookup(rand(address)) \approx rand(Peddler)$$

The *lookup property* is important because it allows customers to know with assurance that their chosen Peddler is an attacker with probability $\frac{a}{n}$.

To implement these operations, the Orchid Market takes the structure of a DHT with no keys and values. To perform the random selection, a user simply generates a random address and locates the Peddler closest to that point.

11.2. Fundamental Peddler Operations

The operations supported by Peddlers on the Orchid Market are:

- *Get Routing Table and Medallion.* This returns the Peddler's proof of work and signed routing table for inspection, along with the cost of relaying traffic to members of the routing table.
- *Relay Traffic.* Pays the Peddler to forward traffic to one of the peers in its routing table.
- *List Services.* Asks the Peddler for a list of services it sells.
- *Purchase Service.* Employ the Peddler as a service provider.

The first two of these are used by customers to navigate to a Peddler of interest, while the second two are used to negotiate the purchase of services once that Peddler is found.

Navigation through the Orchid Market takes a form similar to that used in Chains. A customer connects to some known Peddler (found through bootstrapping, see 11.7), inspects its routing table, and pays to forward traffic to the Peddler closest to its chosen point. As we will see in the section on routing tables, this allows customers to keep their IP addresses secret, while still providing relatively efficient random access to Peddlers of $O(log^2 n)$ packets.

11.3. Chord Routing Structure

Peddlers are connected in the Orchid Protocol using the same scheme used in the Chord DHT. We have chosen Chord over Kademlia due to a more mature literature, and the existence of machine-checked correctness proofs[86].

Addresses are viewed as a ring of size 2^{256}, and the distance between peers at addresses $a, b \in [0, 2^{256} - 1]$ is defined to be n s.t. $0 \leq n < 2^{256}$ and $a + n \equiv b \pmod{2^{256}}$.

For a collection of Peddlers in an Orchid Market, A, the forced connections for a given Peddler e are defined to be $\min_{f \in A} dist(e + t, f)$, for each of 256 target addresses $t \in \{1, 2, 4, ..2^{255}\}$.

We chose to use this routing structure because of its maturity, successful track record in deployed systems, and correctness proofs. Readers interested in learning more are encouraged to read[81]. For our purposes, it is enough to note that the following two properties are provided by this routing scheme:

1. *Finite, Deterministic Connections.* Every Peddler has a number of forced connections ≤ 256.
2. *Logarithmic Traversal Distance.* Given a random address t and a random connected Peddler e with connections C, that $distance(e, t) \approx 2 * \min_{f \in C} distance(f, t)$. Because the distance will halve with each hop, the expected traversal length on the network is $log_2(n)$ where n is the network size.

11.4. Medallions on the Orchid Market

To prevent attackers from choosing their location on the Orchid Market, the location of Peddlers is defined to be the hash of their Medallion. To prevent an attacker from running more Peddlers than is proportional to their share of the Orchid Market's total computational power, every Peddler checks the validity of all its connections' Medallions every Medallion cycle. In the event that a valid Medallion is not supplied, it is disconnected from the network.

11.5. Signed Routing and Eclipse Attacks

One of the issues that arises in distributed networks is that no one (except, perhaps, an attacker) has a global view of the network. For example, imagine if in the above routing scheme an attacker chose to lie about what connections it has – if the buyer has no way of detecting this, they might be led off to a fake Orchid Market in which all "participants" were owned by the attacker. To put a stop to this situation, Peddler routing tables are verified non-malicious by the peers contained on them.

When a node would like to establish a forced connection, that node must prove to each node on its forced connections list that each other node on that list is a member of the same Orchid Market. To do this, we first select a random Peddler G by finding the Peddler with an address closest to the hash of all the connections in the routing table $H(C_i)$. Then we supply:

1. Proof that all the Peddlers on the list can all route to G.
2. Proof that G can route to each Peddler
3. Proof that each Peddler on the list is indeed a forced connection.

These proofs all take the form of signed routing table chains which lead from C_i to G, or in the case of (3) the chain of signed routing tables which led from the Entry Peddler to each C_i. Once such proof has been provided, all of the peers on the new routing table sign the table, and the connecting Peddler signs theirs. For those elements of C_i for whom the new Peddler is a forced connection, the same proof is sent to each of their connections for signature.

Because this is the only method for adding Peddlers to the Orchid Market, these requirements form an inductive proof of the Orchid Market's soundness. If one of the nodes in C_i attempts to supply a fake routing table, it will not route to the same G as the other Peddlers in C_i. If one of the nodes C_i is not a member of the Orchid Market, G will not be able to route to them. If the Peddler seeking to connect has lied about C_i being nearest nodes to his forced connection points, (3) will demonstrate that to be false.

From these properties, we can see that the avenues left for an attacker are:

- If an attacker can generate a Medallion address such that all C_i are controlled by them, the above system will cease to function. This will happen with probability $(\frac{a}{n})^{log(n)}$. If such a collision occurs, $(1 - \frac{n-log(n)}{n})^{log(n)}$ percent of all queries will be compromised. To put these numbers in perspective, if an attacker controls 10% of the network, at 1 million nodes there is a $1 \times 10^{-8}\%$ chance of such a collision happening, and if it does occur around $1 \times 10^{3}\%$ of all system queries will be impacted. At 100 million nodes the chance drops to $1 \times 10^{-12}\%$, causing disruption of 1e-5% of queries. Note this damage is repaired during Regeneration (see Section 11.6).
- If an attacker has now joined the network, but was forced to use a valid routing table, the only attacks it can perform are related to selling services, not routing traffic on the Orchid Market. As this is the situation expected in the rest of our attack models (that an attacker will control a number of Peddlers proportional to the computational resources), we do not consider this an attack.

11.6. Eclipse Attacks and Regeneration

Long lived P2P networks suffer from Eclipse attacks. Although the above signed routing scheme can make these arbitrarily difficult by involving ever increasing number of peers for verification, another approach is simply to limit the lifespan of peers. For this reason, Peddlers on the Orchid Market must change keys every 100 Ethereum blocks.

11.7. Finding Entry Nodes

The distribution of Entry Nodes is a difficult topic. If oppressive governments are able to access this list, they will block user's abilities to access the list. We have therefore located essential services that would be internet-breaking if they were blocked, and have devised methods for adding Entry Node information to the data contained in them.

11.8. Identifying *the* Orchid Market

The above discussion of security is ultimately meaningless if there is no way to locate "the right Orchid Market" on a fresh machine. Any distribution method which exists for *Entry Peddlers* can not be presumed immune from infiltration by Entry Peddlers controlled by an attacker. To do so, we simply estimate the computing power of a given Orchid Market, and select the market in possession of the most total computational power.

- Density Estimation. Because a Peddler's forced connections are defined to be the Peddlers nearest to some set of points in a 2^{256} address space, in any real-world situation there will be measurable gaps between the ideal connections and the actual ones. To estimate density in this space, we can observe that these connections as the result of a random binomial process: every point between the ideal point and the actual point is a failure, and the actual point is a success. Therefore, for a given number of missing nodes M and a given number of realized connections C, The uniform prior MAP estimate of network density is
$$\frac{C}{C+M}2^{256}$$
.
- Traversal Distance. The Orchid Market provides address look ups in $O(log_2(n))$ hops. We can use this in reverse to estimate network density.

One might be inclined to believe that density estimation is enough, however a clever attacker in possession of a sybil network of modest size, will have free choice for which node is to be put forward as the Entry Peddlers for the false network, while the Entry Peddlers from the "real Orchid Market" will have a density which is a random sample from the network. To make matters worse, if the traversal distance is chosen as the metric, one might imagine an attacker who anticipates this, and so creates sub-optimal routing tables which require longer than the $O(log_2(n))$ to traverse. Thankfully, sub-optimally connected Orchid Markets will perform worse on the density metric. The verification method used in the Orchid System is to traverse to a random address, saving the routing tables along the way, and then perform a density estimate using the routing table from all but the first two hops.

11.9. Proxy Whitelists

Some users wishing to offer Proxy services may not be comfortable offering "open access". For example, allowing users to access facebook.com has a risk profile similar to acting as a relay, while allowing arbitrary connections to the Internet may result in a visit from local law enforcement. Peddlers on the Orchid Market may therefore set a whilelist of websites they will allow users to contact when using them as a Proxy, and specify their whitelists in their responses to *Get Offers*.

12. Firewall Circumvention Features

The above system would be of little use if only users already possessing free and open access to the Internet could use it. In this section we will discuss features which ease access for those users whose Internet access is provided by their attacker.

Please note that if an adversary is willing to completely block all Internet access, no defense in this area is possible. All defense analysis in this section therefore assumes that the attacker suffers some cost for blanket blocking, and seeks to maximize this cost in the hope that sufficiently costly attacks will not be performed.

12.1. Bootstrapping

One of the first attacks we anticipate firewall providers to attempt against the Orchid Network is to create a list of Entry Peddlers, and to block all access to them. This is because if customers can not access Entry Peddlers, they could not use the network. Complicating matters, a competent attacker must be assumed to have any list of IP addresses available to customers.

To address this initially, we will provide a service which allows users to learn fresh Relay IP addresses in exchange for proof-of-work. To hinder blocking of the bootstrapping back and forth itself, we will provide access to this boostrapping service via web, email, and popular instant messaging platforms. The user will copy/paste a challenge from their client's options screen into the most convenient communications mechanism, then copy/paste the reply back into the client.

12.2. DPI, Inference, and Active Probing

More sophisticated firewalls employ methods such as Deep Packet Inspection (DPI; analysis of the contents of packets rather than just the headers), timing inference (the use of aggregate statistical measures over packet size, quantity, and timing), as well as active probing (attempting connection with the user or the server they are connecting to in an attempt to identify the service being provided.)

We do not anticipate the use of deep packet inspection or active probing to provide significant information. Through our use of WebRTC, all communication is encrypted and there are no open ports unless an active WebRTC offer has been issued. Since this matches the behavior of all other uses of WebRTC, this behavior can not disambiguate Orchid users.

Timing inference is potentially an effective method for detecting Orchid users, as the timing and size of web requests over an encrypted stream are unlikely to look like other kinds of WebRTC traffic ([63]). To address this, users accessing the Orchid Network in situations where inference attacks are likely are encouraged to use "bandwidth burning" (Section 8.2).

12.3. Disclosure: Ethereum Traffic

Because the current client employs an Ethereum client to track payment statuses, and Ethereum has its own non-hardened networking signature, detection related to this Ethereum traffic is likely to be the weak link. Firewall operators may simply ask "is the computer running Ethereum *and* consuming large amounts of WebRTC traffic?"

To maintain project focus, hardening of Ethereum and/or serving Ethereum traffic over the Orchid Network is not a feature of our initial release. We plan on addressing this in future versions, see Section 15.3.

13. Performance Scaling

In this section we examine how the system will function as the number of users grows.

13.1. Algorithmic Performance

Broadly, there are three parts to the Orchid Protocol: Ethereum-based payments, manifolds, and the Orchid Market.

Ethereum-based payments scale with Ethereum as normal transactions. Having reviewed the Ethereum system design, we are confident that even if the Orchid Network is extremely successful, and becomes a significant percentage of the Ethereum's total transaction volume, this component will function within design tolerances.

Manifolds are chains of bandwidth sellers (Relays and Proxies) all of which have performance characteristics independent of the total number of Orchid Network participants.

The core operations of the Orchid Market are based on the well-studied Chord DHT. The number of connections that Peddlers must maintain grows at a rate of $O(log(n))$, to a maximum of 256 connections. Queries on the network require $O(log(n))$ hops. Although these operations do become more burdensome as the network increases in size, we do not believe any significant impact on performance will result.

13.2. Allocation of Scarce Resources

The Orchid Protocol is built around tokens. These tokens will allow, through price discovery, for graceful handling of a change in balance between buyers and sellers.

For example, if Relays are in short supply, rather than providing all customers with a slow experience, customers will engage in a bidding war to determine who can use the system until the shortage is corrected. Conversely, if Relays are in abundant supply, some Relays may leave the system until such time as prices rise.

13.3. Real-World Performance

As the software is not yet complete, we do not have concrete numbers to provide here. On release we will update this document with the following graphs:

1. Chain setup time as a function of Orchid Market size.
2. Orchid Market join time as a function of Orchid Market size.
3. How quickly the price adjusts to scarcity, abundance.
4. Add any interesting ideas here!

14. Attack Analysis and Attacker User Stories

14.1. Oppressive Web Applications

Attacker Goals: Identify all Orchid Relay and Proxy IP addresses.

Because the Orchid Market contains all Relays and Proxies, this is the inverse attack of the one discussed in Section 12.

The number of forced connections on the Orchid Market grows at $O(log(n))$ where n is the network size. If an attacker holds $m\%$ of global computation, they will learn $log(n)$ IP addresses each time they complete a proof-of-work computation. In c epochs they will therefore learn $1 - (1 - \frac{log(n)}{n})^c$ percent of the Relay and Proxy IP addresses.

Readers familiar with the how the blocking of Tor traffic panned out may worry the above describes a dire issue for the system. Fortunately, this is not the case. Tor has around 1,000 exit nodes, which allowed for easy filtering. In our case, largely due to our support for whitelists, we expect to have hundreds of thousands of exit nodes. In addition to this forming a much larger computational challenge to unmask using the above method, blocking these IP addresses would result in the oppressive Web Application blocking their own users.

14.2. Corporate Networks and "Great" Firewalls

Attacker Goals: Prevent usage of the Orchid network by users to whom you provide Internet access.

Features related to this are discussed in more detail in Section 12. The outlook for this attacker is bleak: Orchid network usage presents as WebRTC connections relaying a fixed amount of data. There are no open ports to probe, and no IP addresses which can be relied on to "out" them.

14.3. Passive Monitoring and Inference, perhaps with Sybil Attacks

Attacker Goals: Learn Customer IP Identification, and Website Identification.

Analysis related to this class of attack are discussed in more detail in Section 9.2. The outlook for this attacker is bleak: the difficulty of positioning yourself in several positions of a Chain requires possessing (and dedicating to this attack) a large percentage of global computation power.

14.4. Small-Time Trolling and QoS Attacks

Synopsis: An attacker would like to cause mayhem on as much of the network as possible.

Security minded readers looking for a good time are encouraged to perform analysis in this domain. The task here is, given a limited budget (perhaps on the order of $10,000 USD), to disrupt the network as much as possible.

14.4.1. Attacking Chains

Attackers may try a variety of attacks here: randomly dropping packets, providing only very slow service, providing intermittently slow service, or simply disconnecting. In all cases, the customer simply replaces the node in question, leading to a minor inconvenience spread across all customers. An additional inconvenience may be caused to other relays in the case of dropping packets, since there may be no way of determining if A did not forward the packet of if B is lying about not having received it. In this case the customer replaces both Relays.

14.4.2. Attacking the Orchid Market

Attackers have similarly many options for this situation.

One option is to improperly implement the joining protocol. We do not view this as an attack, as in this case our "attacker" is simply paying other Peddlers for packet forwarding.

Another option is to join the Orchid Market and refuse to provide routing table information to users, or refuse to forward packets. This will result in some additional routing burden on $\frac{log(n)}{n}$ of the Orchid Market queries until the Peddler in question is disconnected from the network.

A third option is to sit on the Orchid Market while offering no services. In this case auctions performed by customers become less efficient (suffering from the loss of one participant $\frac{log(n)}{n}$ of the time).

15. Future Work

The items in this section fall into two categories: nice-to-haves, and features we are internally conflicted about releasing to the public. We believe this is conflictedness is universal – although almost everyone has favorite examples of power being used for oppression, there are also countless examples of power being used for good. Protocols like Orchid have no judgement of their own, and so cannot tell if they are routing traffic for a freedom fighter or a terrorist, villain or hero.

15.1. Proof of Space

As mentioned in Section 5, we are very interested in exploring alternative proof types. This is an important issue both because of the environmental impact of proof-of-work systems, and because our current proof-of-work algorithm requires full blown computers to act as network routers.

We are excited to explore the possibility of using disk space to be the scarce resource at the core of our security, which might allow old phones or similar hardware to profitably participate in the Orchid network.

15.2. Protecting Content Hosts

Many prior approaches (Section 3) discovered that content hosts sought similar protections as web users. We are internally conflicted on this point, as we do believe there is content which it is in the public interest not to have freely distributed (information related to the manufacture of nuclear weapons for example). However, should unforseen circumstances demand it, Orchid could be extended to support such "unrestricted, unsurveilled Hosts" as seen in the following diagram:

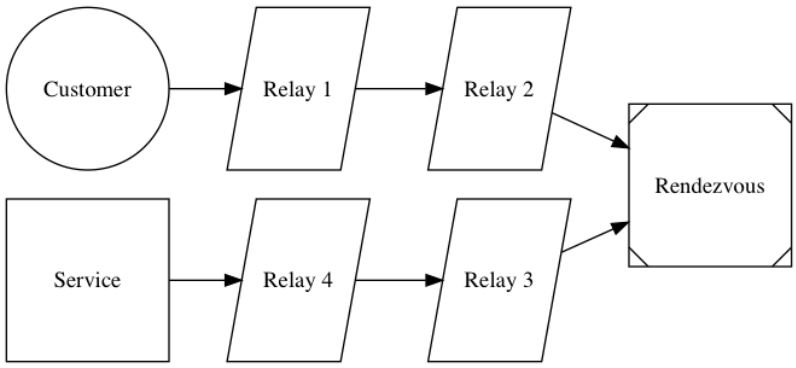

Figure 3: A rendezvous node acting as a relay between a Service and a Customer

15.3. Securing Ethereum Traffic

As discussed in our section on firewall avoidance (Section 12), the Ethereum network traffic of clients is likely to be the weak link. Because all nodes must maintain this information, use of the Orchid protocol to

distribute Ethereum information seems like a natural fit.

Unfortunately, relying on those you are paying for information about payments leads to tricky issues. We hope to add this in the near future, but will not be including it in our initial release.

15.4. Orchid as a Platform

Although we anticipate that design of the core system will take up much of our time for the immediate future, we are very interested in the possibility that adding features to support the following use cases may drastically increase the amount of bandwidth routed through the Orchid Network.

1. APIs for websites to directly interface to the network, and incorporate tokens into their service.
2. On-Network file storage and static website hosting.
3. File Sharing.
4. Email/Messaging service.
5. An arbitration/moderation service.

References

[1] URL: http://www.meshlabs.org.

[2] URL: https://en.wikipedia.org/wiki/Payment_service_provider.

[3] URL: https://en.wikipedia.org/wiki/ISO/IEC_7816.

[4] URL: http://www.ebics.org/home-page.

[5] URL: https://www.swift.com.

[6] URL: http://www.nyce.net/about.

[7] URL: http://www.investopedia.com/terms/r/reconciliation.asp.

[8] URL: https://www.quora.com/What-are-common-credit-card-processing-fees.

[9] URL: https://www.nerdwallet.com/blog/banking/wire-transfers-what-banks-charge.

[10] URL: https://www.economist.com/blogs/dailychart/2010/12/remittances.

[11] URL: https://www.valuepenguin.com/what-credit-card-processing-fees-costs.

[12] URL: https://bitcoin.org/en/developer-guide#transactions.

[13] URL: https://bitinfocharts.com/comparison/bitcoin-transactionfees.html.

[14] URL: https://bitinfocharts.com/comparison/ethereum-transactionfees.html.

[15] URL: http://ethgasstation.info.

[16] URL: https://etherscan.io/token/OmiseGo.

[17] URL: https://blog.ethereum.org/2015/07/05/on-abstraction.

[18] URL: https://blog.ethereum.org/2015/12/24/understanding-serenity-part-i-abstraction.

[19] URL: https://github.com/ethereum/EIPs/issues/662#issuecomment-312709604.

[20] URL: https://blog.coinbase.com/app-coins-and-the-dawn-of-the-decentralized-business-model-8b8c951e734f.

[21] URL: http://onchainfx.com.

[22] URL: https://0xproject.com/.

[23] URL: https://etherdelta.com/#REQ-ETH.

[24] URL: https://augur.net.

[25] URL: https://gnosis.pm.

[26] URL: https://medium.com/@vishakh/a-deeper-look-into-a-financial-derivative-on-the-ethereum-blockchain-47497bd64744.

[27] URL: http://mojonation.net.

[28] URL: https://en.bitcoin.it/wiki/Payment_channels.

[29] URL: https://en.bitcoin.it/wiki/Contract.

[30] URL: https://lightning.network/lightning-network-paper.pdf.

[31] URL: https://en.wikipedia.org/wiki/Mining_pool#Mining_pool_methods.

[32] URL: https://bitcoin.stackexchange.com/questions/1505/what-is-a-share-can-i-find-it-while-mining-solo-or-only-when-pool-mining.

[33] URL: https://tools.ietf.org/html/draft-goldbe-vrf-00.

[34] URL: https://blog.ethereum.org/2017/10/12/byzantium-hf-announcement.

[35] URL: https://github.com/ethereum/EIPs/pull/213.

[36] URL: https://github.com/ethereum/EIPs/pull/212.

[37] URL: https://en.bitcoin.it/wiki/Address_reuse.

[38] URL: https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki.

[39] URL: https://ropsten.etherscan.io/address/0x5e10d764314040b04ac7d96610b9851c8bc02815#code.

[40] URL: https://z.cash/technology/zksnarks.html.

[41] URL: https://solidity.readthedocs.io/en/develop/assembly.html.

[42] URL: https://hackernoon.com/zksnarks-and-blockchain-scalability-af85e350a93a.

[43] URL: https://hackernoon.com/scaling-tezo-8de241dd91bd.

[44] URL: https://p16.praetorian.com/blog/man-in-the-middle-tls-ssl-protocol-downgrade-attack.

[45] URL: https://theethereum.wiki/w/index.php/ERC20_Token_Standard.

[46] Martin Abadi et al. "Moderately hard, memory-bound functions". In: *ACM Transactions on Internet Technology (TOIT)* 5.2 (2005), pp. 299–327.

[47] Eli Ben-Sasson et al. "SNARKs for C: Verifying program executions succinctly and in zero knowledge". In: *In Advances in Cryptology–CRYPTO 2013*. Springer. 2013, pp. 90–108. URL: https://dspace.mit.edu/bitstream/handle/1721.1/87953/880419628-MIT.pdf.

[48] Adam Bergkvist et al. "Webrtc 1.0: Real-time communication between browsers". In: *Working draft, W3C* 91 (2012).

[49] Daniel Bernstein, Tanja Lange, and Peter Schwabe. "The security impact of a new cryptographic library". In: *Progress in Cryptology–LATINCRYPT 2012* (2012), pp. 159–176.

[50] Jean-Luc Beuchat et al. "High-Speed Software Implementation of the Optimal Ate Pairing over Barreto–Naehrig Curves". In: *4th International Conference on Pairing-Based Cryptography*. Springer. 2010. URL: https://eprint.iacr.org/2010/354.pdf.

[51] Alex Biryukov and Dmitry Khovratovich. "Equihash: Asymmetric proof-of-work based on the generalized birthday problem". In: *Ledger* 2 (2017).

[52] N. Bitansky et al. "Recursive composition and bootstrapping for SNARKs and proof-carrying data". In: *In Proceedings of the forty-fifth annual ACM symposium on Theory of computing*. ACM. 2013, pp. 111–120. URL: https://eprint.iacr.org/2012/095.pdf.

[53] Nikita Borisov et al. "Denial of service or denial of security?" In: *Proceedings of the 14th ACM conference on Computer and communications security*. ACM. 2007, pp. 92–102.

[54] Michael Brown et al. "Software implementation of the NIST elliptic curves over prime fields". In: *Topics in Cryptology—CT-RSA 2001*. Springer. 2001, pp. 250–265. URL: https://pdfs.semanticscholar.org/ac3c/28ebf9a40319202b3c4f64cc81cdaf193da5.pdf.

[55] David Brumley and Dan Boneh. "Remote timing attacks are practical". In: *Computer Networks* 48.5 (2005), pp. 701–716.

[56] Vitalik Buterin. *Ethereum: A next-generation smart contract and decentralized application platform*. https://github.com/ethereum/wiki/wiki/White-Paper. Accessed: 2016-08-22. 2014. URL: https://github.com/ethereum/wiki/wiki/White-Paper.

[57] David L Chaum. "Untraceable electronic mail, return addresses, and digital pseudonyms". In: *Communications of the ACM* 24.2 (1981), pp. 84–90.

[58] Shuo Chen et al. "Side-channel leaks in web applications: A reality today, a challenge tomorrow". In: *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE. 2010, pp. 191–206.

[59] Alessandro Chiesa et al. "Decentralized Anonymous Micropayments". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2017, pp. 609–642.

[60] George Danezis. "The Traffic Analysis of Continuous-Time Mixes". In: *Proceedings of Privacy Enhancing Technologies workshop (PET 2004)*. Vol. 3424. LNCS. May 2004, pp. 35–50.

[61] Roger Dingledine, Nick Mathewson, and Paul Syverson. *Tor: The second-generation onion router*. Tech. rep. Naval Research Lab Washington DC, 2004.

[62] Cynthia Dwork, Moni Naor, and Hoeteck Wee. "Pebbling and proofs of work". In: *CRYPTO*. Vol. 5. Springer. 2005, pp. 37–54.

[63] Kevin P Dyer et al. "Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail". In: *Security and Privacy (SP), 2012 IEEE Symposium on*. IEEE. 2012, pp. 332–346.

[64] Christoph Egger et al. "Practical attacks against the I2P network". In: *International Workshop on Recent Advances in Intrusion Detection*. Springer. 2013, pp. 432–451.

[65] Ittay Eyal and Emin Gün Sirer. "Majority is not enough: Bitcoin mining is vulnerable". In: *International conference on financial cryptography and data security*. Springer. 2014, pp. 436–454.

[66] Mike Hearn. *[Bitcoin-development] Anti DoS for tx replacement*. Bitcoin development mailing list. 2013. URL: https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2013-April/002417.html.

[67] Martin Hellman. "A cryptanalytic time-memory trade-off". In: *IEEE transactions on Information Theory* 26.4 (1980), pp. 401–406.

[68] J Herrera-Joancomartí. "Research and challenges on bitcoin anonymity". In: *In Data Privacy Management, Autonomous Spontaneous Security, and Security Assurance*. Springer. 2015, pp. 3–16. URL: https://www.researchgate.net/profile/Jordi_Herrera-Joancomarti/publication/281773799_Research_and_Challenges_on_Bitcoin_Anonymity/links/55f7c7d408ae07629dcbc471.pdf.

[69] Douglas R. Hofstadter. *Metamagical Themas: Questing for the Essence of Mind and Pattern*. New York, NY, USA: Basic Books, Inc., 1985. ISBN: 0465045405.

[70] Don Johnson, Alfred Menezes, and Scott Vanstone. "The elliptic curve digital signature algorithm (ECDSA)". In: *International Journal of Information Security 1, no. 1 (2001)*. Springer, pp. 3–63. URL: http://residentrf.ucoz.ru/_ld/0/34_Digital_Signatu.pdf.

[71] Joshua A. Kroll, Ian C. Davey, and Edward W. Felten. "The economics of Bitcoin mining, or Bitcoin in the presence of adversaries". In: *Proceedings of WEIS (Vol. 2013)*. WEIS. 2013, p. 11. URL: http://www.thebitcoin.fr/wp-content/uploads/2014/01/The-Economics-of-Bitcoin-Mining-or-Bitcoin-in-the-Presence-of-Adversaries.pdf.

[72] Thomas Locher et al. "Free riding in BitTorrent is cheap". In: *Proc. Workshop on Hot Topics in Networks (HotNets)*. 2006, pp. 85–90.

[73] Daniel Lorimer. *Momentum–a memory-hard proof-of-work via finding birthday collisions, 2014*. URL: http://www.hashcash.org/papers/momentum.pdf.

[74] Sarah Meiklejohn and Rebekah Mercer. "Möbius: Trustless Tumbling for Transaction Privacy". In: 2017. URL: https://allquantor.at/blockchainbib/pdf/meiklejohn2017moebius.pdf.

[75] Adnan Noor Mian et al. "Enhancing communication adaptability between payment card processing networks". In: *IEEE Communications Magazine, 53(3)*. IEEE. 2015, pp. 58–64. URL: https://www.researchgate.net/profile/Abdul_Hameed46/publication/273835174_Enhancing_communication_adaptability_between_payment_card_processing_networks/links/58454f7408ae2d21756751f7.pdf.

[76] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. 2008.

[77] Arvind Narayanan et al. *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton University Press, 2016.

[78] Rafael Pass and Abhi Shelat. "Micropayments for Decentralized Currencies". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2015. URL: https://pdfs.semanticscholar.org/bca9/92b35d844160b30edbbf1809e17551d867ea.pdf.

[79] Ronald L Rivest. "Electronic Lottery Tickets as Micropayments". In: *International Conference on Financial Cryptography*. Springer. 1997. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.135.2668&rep=rep1&type=pdf.

[80] Rabin Silvio and Vadhan. "Verifiable Random Functions". In: *Foundations of Computer Science, 1999. 40th Annual Symposium on*. IEEE. 1999. URL: https://dash.harvard.edu/bitstream/handle/1/5028196/Vadhan_VerifRandomFunction.pdf.

[81] Ion Stoica et al. “Chord: A scalable peer-to-peer lookup service for internet applications”. In: *ACM SIGCOMM Computer Communication Review* 31.4 (2001), pp. 149–160.

[82] John Tromp. “Cuckoo Cycle: a memory-hard proof-of-work system.” In: *IACR Cryptology ePrint Archive* 2014 (2014), p. 59.

[83] Liang Wang and Jussi Kangasharju. “Real-world sybil attacks in BitTorrent mainline DHT”. In: *Global Communications Conference (GLOBECOM), 2012 IEEE*. IEEE. 2012, pp. 826–832.

[84] David Wheeler. “Transactions using bets”. In: *International Workshop on Security Protocols*. Springer. 1996, pp. 89–92.

[85] Gavin Wood. *ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER*. URL: `https://ethereum.github.io/yellowpaper/paper.pdf`.

[86] Pamela Zave. “How to make Chord correct”. In: *arXiv preprint arXiv:1502.06461* (2015).

A. Auctions

When purchasing bandwidth, price-sensitive customers can be taken advantage of by attackers offering very low prices. For example, imagine a customer who plans to purchase a length 4 chain on a lowest-bid basis. An attacker who knows this can set their prices to the minimum possible amount, thereby achieving greater than $\frac{a}{n}$ chance of being picked for each node in the chain.

To address this, customers using the Orchid Market select a random subset of the bandwidth selling population, then select their providers randomly from the set of affordable chains of the required length that can be created from that random subset.

This creates a circumstance where attackers need to "get lucky" both in their assigned location, and also in the price they select relative to the prices set by the other randomly selected sellers. Even given this constraint, familiar market properties such as the ability of a seller to influence demand through price changes are maintained.

A.1. Appendix Overview

In this Appendix, we will look at what strategies are available to privacy minded bandwidth buyers, and what the performance of the different strategies are.

A simplified model of the Orchid Market is introduced, suited to analysis of this problem, and several example approaches are worked out both in theory and for concrete examples. It is our hope that readers of this section will come away with an understanding of the trade-offs which were made in selecting our algorithm for bandwidth purchases, and may be dissuaded from hard coding their client to employ a different method.

A.2. Simplified Model for Analysis

The full complexity of the Orchid Market need not be considered to evaluate auction strategies. To simply our analysis, we introduce here a set of assumptions about participant goals:

1. **Sellers**. Sellers are in possession of r "bandwidth slots" to which they would like to sell access. The sole goal of sellers is to maximize their earnings from this source.
2. **Attackers**. Attackers are in possession of $a \geq 2$ sellers. Their sole goal is to have a single buyer purchase more than one of the slots from their sellers, which is termed a successful attack.
3. **Buyers**. Buyers seek to buy three bandwidth slots from three different sellers (thereby thwarting attackers), at the lowest price possible.

Rather than concern ourselves with the details of the Orchid Market, we will assume that all buyers instead possess an up-to-date list of all current medallion holders and their current bandwidth price. We will also not concern ourselves with the distinction between Relays and Proxies, or the complexities introduced by whitelists and other capability filtering.

Now that we have a basic structure, and an understanding of participant goals, let us flesh out a game structure:

1. **Setup** The strategy to be used by buyers is told to all sellers, and all attackers. The strategy to be used by sellers, which may change depending on buyer strategy, is told to attackers.
2. **Phase One**. All sellers select a price. Seller prices are then revealed to the attackers, who then select their prices.
3. **Phase Two**. All prices are revealed to the buyers. Buyers are asked in random order to select up to three available offers from the list.

4. **Phase Three**. Profit is distributed.
   (a) Sellers and attackers receive money equal to their offer price from any buyers who selected them.
   (b) Buyers receive some profit amount specific to each buyer if they purchased three slots without suffering an attack, less the price paid for those slots.
   (c) Any attackers who successfully attacked a buyer receives a buyer-specific bounty U_b.

The market is then an n-player game for which we seek three strategies: what buyers should do, what sellers should do, and what attackers should do. We have made the buyer "go first" on purpose in the above design, as buyers are the least economically normal in their needs.

Some readers may take issue with the idea that a buyer's strategy is shared with attackers in the above game. We do this explicitly because a motivated attacker who is initially unsuccessful in an attack may still acquire information on the rough prices paid by a given buyer (for example by determining the IP address and pricing of all sellers, then monitoring the Internet connection of that buyer to determine which bandwidth seller was used as the first hop.) Over time this information leakage may be used to infer the strategy being employed, hence we feel it is best to simply assume it is possessed by the attacker for the purposes of this analysis.

A.2.1. Success Criteria

A successful solution will be determined by performance on the following criteria:

1. **Security**. The customer is maximally protected from attacks given a fixed budget and chain length.
2. **Stability**. No member of any of the three groups can improve their personal utility by changing strategies.
3. **Economically compatible**. Sellers can raise and lower prices to modulate the number of purchase orders they receive, allowing for familiar methods to be employed for maximizing seller profit.

Stability should be given special attention by those readers familiar with optimization but unfamiliar with game theory, as there is considerable temptation to propose group-level "optimal strategies" wherein the members of a group band together to protect their interests as a whole. Although such behavior may appear rational on its surface, the incentive structures present in fully anonymous, distributed markets prevent them from being stable. For example, it might appear that sellers in a situation where the total slots on offer exceeds demand might band together and set a minimum price (a *trust* in economic terms, superrationality[69] in game theoretic terms, a class revolution in Marxist terms.) Unfortunately, because additional profit will be available to any sellers who lower their price below the trust price, such an arrangement is not stable. For this reason, we will not be considering them in this analysis. This is not to rule out their eventual application in this domain; perhaps future advances in blockchain technology will allow for distributed verification of adherence to such agreements.

Another potential surprise is that we attempt to explicitly determine how an attacker should behave so as to maximally exploit buyers, and include the stability of such strategies as a criterion for "success." We do this because there is no other way to provide bounds on the security of a given approach.

A.3. Selection Attacks

Before we delve too far into discussion of buyer strategy, let us first consider the goals of the attacker, and what constitutes an attack. If we imagine a perfectly paranoid buyer, with infinite budget and no information other than what is contained on the list of sellers, it is plain they can do no better than random selection. This gives attackers a successful attack rate of $(\frac{a}{n})^2$. As this is the best possible result, we consider a strategy affording such odds of attack secure.

An attack in this area is then any method whereby an attacker might increase the chance of successful attack above $(\frac{a}{n})^2$ probability.

A.4. Candidate Strategies

With that background out of the way, we now proceed to evaluate buyer strategies.

A.4.1. Lowest-Price

If the buyer elects to go with the lowest cost provider, a component attacker will set their prices to the lowest allowed (0 tokens). This yields a successful attack of $\frac{a}{z}$ where z is the number of nodes charging zero.

For example, consider a market with three genuine sellers each offering a single slot at the price points: {2, 4, 6}, and a buyer with maximum total price of 12. A competent attacker will enter the market at the prices {0, 0}, and so will be guaranteed the sale, putting the chance of successful attack at 1.

A.4.2. Price-Weighted Random

If the buyer elects to make the chance of purchase some monotonic function of the difference in cost of bandwidth, this results in moderately better performance of:

$$\frac{af(0)}{\sum_{e \in S} f(e)}$$

Returning to the above example, and using the inverse squared increase in cost as our function, we arrive at $\frac{288}{337} \approx 85\%$ chance of successful attack. While 85% is much better than 100%, it is still unsatisfying.

A.4.3. Random Selection From Affordable Relays

If the buyer elects to select randomly from sellers charging less than $\frac{1}{3}$ of their maximum price, a component attacker will set their prices to be at or below that maximum. When in doubt, the attacker can again select a price of 0.

Returning to our example, a component attacker will enter the market at the prices 0 and 1, or equivalent, leading to two non-attack combinations: {(0, 2, 4), (1, 2, 4)}, and two attack combinations: {(0, 1, 2) and (0, 1, 4)}. This yields a chance of successful attack of $\frac{1}{2}$.

A.4.4. Random Selection Subject to Cost

If the buyer elects to select randomly from triples of sellers (S_i, S_j, S_k) such that the total cost is less than or equal to the maximum cost, the attacker can then select their prices so as to (1) maximize the number of triples (A_i, A_j, S_k) which the buyer can afford, (2) minimize the number of triples (A_i, S_j, S_k) which the buyer can afford.

Returning to our example, a competent attacker will enter the market at the prices 1 and 4.1, or equivalent, leading to 5 non-attack combinations: {(1, 2, 4), (1, 2, 6), (1, 4, 6), (2, 4, 4.1), (2, 4, 6)} and three attack combinations: $\{(1, 2, 4.1), (1, 4, 4.1), (1, 4.1, 6)\}$ Hence the chance of successful attack is $\frac{3}{8}$.

Note the effective "crowding out" that is accomplished by the attacker selecting the 4.1 price point – only one of four combinations which include 4.1 is *not* a successful attack. It is remarkable that even after accounting for such behavior on the part of an attacker, random selection subject to cost is still better than discarding the seller charging 6 from consideration.

A.4.5. Random Cost Selection, Normalized by Peer

A natural question to ask is: what if we bias our random sample to prevent peers from being underrepresented? Unfortunately, the answer is attackers will use this to their advantage.

Continuing the above example, a component attacker will select 1 and 6.1 as their prices, or equivalent. Because we the buyer can only afford 6.1 when paired with 1, adjusting the odds results in a $\frac{3}{7}$ chance of a successful attack.

A.4.6. Random Cost Selection, Normalized by Pair

Another idea, similar to the one above, is to ask: what if we bias our random sample to normalize the probability of *pairs* of sellers being selected? By controlling which pairs are rare, the attacker again increases the liklihood of successful attack.

Continuing the above example, a component attacker will again select 1 and 6.1 as their prices, or equivalent, this time realizing a success rate of $\frac{24}{49}$, as pairs involving 6.1 were quite underrepresented.

A.5. Stability Analysis

Now that we have outlined some candidate approaches, the question arises: are buyers incentivized to deviate from a given strategy, and if so what happens to the security properties of each strategy as attackers seek to exploit a mixed population of buyers?

To avoid analysis for analysis' sake, we have omitted sections for strategies which are suboptimal/unstable from both a pricing perspective and from a security perspective.

A.5.1. Lowest-Price

Lowest-price is stable against economic incentives as the price chosen is already the lowest possible. From a security perspective, however, it is sub-optimal and so unstable. Security-conscious buyers will use a different strategy, presumably Random Selection Subject to Cost.

This leads to an interesting situation wherein the attacker is forced to decide how to allocate their resources between these two types of buyers, to the security benefit of both groups, as discussed in the next section.

A.5.2. Random Selection Subject to Cost

As the inverse of the previous discussion, this strategy is stable from a security standpoint, but not stable to economic incentives – those buyers who are not interested in security will employ Lowest-Cost selection.

Some readers may be surprised by the claim that this strategy is stable from a security standpoint. Perhaps some buyers on the Orchid Market will use their knowledge about how an attacker should go about exploiting Random Selection Subject to Cost, to create their own improved selection method? As previously discussed, the Orchid Market is not secure against inference of buying strategies, and more troubling there is no way of knowing the extent to which an attacker may have inferred an alternative chosen method. Therefore for sufficiently paranoid buyers this method is stable, as it performs best under worst-case assumptions.

However, in as much as some number of the Orchid Market buyers ignore this advice, or simply employ the Lowest-Cost selection method, this is good news for security conscious buyers. Any secondary attack optimization will only cause attackers fail to optimally exploit Random Selection Subject to Cost.

A.6. Economic Compatibility Analysis

We will now turn our attention to the question of seller strategies. Our goal here is to show the extent to which the usual sort of economic algorithms can be employed by sellers.

We have omitted sections for strategies which are suboptimal/unstable from both a pricing perspective and from a security perspective.

A.6.1. Lowest-Price

As this approach is the expected case in economics, it is fully economically compatible.

A.6.2. Random Selection Subject to Cost

Although it may not initially appear that this strategy is economically compatible, when a population of buyers which do not share a maximum price are considered, the frequency of interest in a seller's goods takes on the familiar shape of price sensitivity.

Since sellers can modulate the number of purchase orders they receive by raising and lowering their prices in the usual way, Random Selection Subject to Cost is economically compatible.

A.7. Conclusion

We have now walked through our analysis of those auction methods suited for buyers on the Orchid Market, and thereby showed how Random Selection Subject to Cost was selected for general use.

The reason we have selected a "pure random" approach stems from the assumption an attacker will both full knowledge of a buyer's strategy, and from the assumption that attackers will pick their prices after legitimate sellers have chosen theirs. In this context, the best that a biased sample can do is nothing, while at worst it allows the attacker to increase their chance of selection. Rather than bias our sample, we have maximized the number of options available and selected from that space uniformly.

Readers who are worried about this increase in cost to buyers relative to a more traditional auction model are encouraged to consider the premium to be "the price of security" – as we have seen it is trivial to achieve a lower price by leaving oneself wide open to attack.

B. Related Work

The Orchid project draws upon a large body of work in the areas of peer-to-peer networks (P2P), blockchain, cryptography, and overlay networks. Orchid then combines the insights provided by those earlier works with more recent P2P research in Blockchain technology, notably Ethereum[**11**] and Zcash[**12**].

The following sections describe the role previous work plays in the Orchid project.

B.1. Virtual Private Networks

Virtual Private Networks (VPNs) use encryption to securely transport a VPN subscriber's traffic across a larger insecure network. This encryption may prevent tracking of browsing habits and unique online identifiers, such as a user's IP address, and circumvent access restrictions.

VPN users should not assume that their VPN connection is truly secure or anonymous. Some VPN service providers track their customers' network activity, then sell the data to third-party commercial entities without

the approval, or even the knowledge, of the VPN subscribers. The IP addresses of a VPN provider's network nodes may also be identifiable. That can enable governmental or commercial entities such as Netflix to block traffic to and from a VPN provider's servers.[**13**].

Those weaknesses in VPNs led to the development of decentralized overlay networks. Decentralized overlay networks provide VPN services with a continuously changing set of exit nodes. If a site blocks traffic one from VPN exit node, one or more alternative exit nodes are dynamically put into service.

B.2. Peer-to-Peer Protocols

Peer-to-peer protocols date back to the Napster file sharing network.[**42**]. Napster used incentives to encourage subscribers to host music files in return for being able to download files from other peers.

B.2.1. The Napster Network

Napster used a centralized directory service that indexed files and the locations of peers. The centralization of knowledge that resulted from Napster's approach rendered them susceptible to legal action by the MPAA (Motion Picture Association of America). That in turn eventually forced Napster out of business.

The vulnerability of Napster's centralized directory inspired the designers of Napster's successor, Gnutella, to distribute the index of files and node addresses across each peer in the network[**43**].

B.2.2. The Gnutella Network's Distributed-Index Response

The Gnutella network's designers remedied the shortcomings of Napster centralized directory by implementing a distributed-index approach. This approach delivered improved resilience and scalability over Napster. Those improvements also inspired the development of additional frameworks for distributing indexes across P2P networks. One notable example is the adoption of Distributed Hash Tables (DHTs) to enable efficient discovery of nodes and resources in P2P networks.

B.2.3. The Tor Network

Tor was developed in the mid-1990's by the United States Navy. Since then, development by the open-source community and the use of Tor has remained flat. There are today roughly 7,000 nodes, 3,000 exit nodes, and approximately 2 million users worldwide.

Tor's use of centralized node selection and reliance on volunteers to provide node services, including exit node services, negatively impacts Tor's throughput. This stems from Tor's inability to use BitTorrent or other P2P file sharing systems, and from the ability of exit nodes to inspect the contents of exit traffic. In addition, Tor has no mechanism for preventing exit nodes from being forced to access illegal or dangerous information on behalf of other users.

Despite those issues, Tor's relatively small development community continues to investigate how the Tor network might be made faster, more reliable, and more secure[**25**]. A key part of that discussion is how to bring lower latency / higher bandwidth nodes into the network[**20**, **21**, **22**, **23**, **24**].

To achieve those goals, the Tor network must find a way to offer users' incentives. Retrofitting Tor with a means of receiving financial contributions from users presents multiple obstacles. Tor's inability to tightly couple payment with routing makes it difficult to effectively manage anonymous digital payments. The Tor community's insistence that some nodes continue to route for free while others receive payment for being "gold star" members adds yet another layer of complexity.

Another non-technical reason for Tor's limited growth is that it is often perceived as a tool designed primarily to enable technically sophisticated users ("techies") to access illegal services or dark web sites. One example

of that sort of hidden service was the *Silk Road* web site that offered a variety of illegal goods and services.[**18**, **19**].

In contrast, the Orchid Network will not enable hidden services and focus only on open, secure, anonymous access to the Internet.

B.2.4. Onion Routing

The techniques of Onion Routing described here (and Garlic Routing described in Section B.2.5), combined with encryption, deliver a greater level of anonymous routing across P2P networks.

Onion Routing is a "layered" approach to data encryption that creates paths through a P2P network. Messages are repeatedly encrypted by the originating node, then decrypted successively by each node that the message transits through. Intermediate nodes receive only the routing instructions needed to route the message. Only the final (exit) node receives both the routing instructions and the message.

One frequently cited example of Onion Routing is the Tor network (Section B.2.3).

B.2.5. Garlic Routing

The Invisible Internet Project (*I2P*) is a decentralized anonymizing network based on principles similar to Tor (B.2.3) but designed from the ground up to be a self-contained darknet. A key design feature of I2P is its use of Garlic Routing[**26**].

Garlic Routing bundles multiple messages together into a single packet referred to as a *bulb*. Each message in a bulb is in turn encrypted in the layered encryption style of Onion Routing. The bundling of messages means that accessing I2P is significantly faster than Tor for hidden services. I2P only partially supports routing to the wider Internet making a full comparison of the performance improvements difficult to assess. Bundling also makes traffic analysis more difficult to determine.

I2P users connect to each other using peer-to-peer encrypted tunnels but without a centralized directory as used by Tor. I2P completely separates incoming and outgoing traffic. Packet switching, rather than circuit switching, is then used to provide transparent load balancing of messages across multiple peers. These design features are combined to improve both security and anonymity.

One aspect of I2P which requires significant improvement is its management of its distributed database of nodes. I2P originally used Kademlia as originally designed in 2002[**27**]. The initial version of Kademlia continuously consumed so much CPU and network bandwidth that it could not be scaled. I2P then transitioned to an algorithm they called *floodfill*. However, the floodfill mechanism also suffers from design flaws that can be exploited to corrupt and manipulate the information in I2P's distributed database[**28**].

B.3. Blockchain Platforms

Blockchain protocols enable permissionless, decentralized consensus on global state and the use of cryptographic tokens to provide incentives to run nodes.

Blockchain designs such as those of Bitcoin, Ethereum, Zcash, and others are examples of blockchain protocols that use state transition functions to add or change entries in their global state. These protocols also reward nodes for validating transactions and forming consensus on their ordering using techniques such as Proof-Of-Work[**44**].

B.3.1. Ethereum

Ethereum[56], along with Bitcoin[76], have pioneered new forms of application-specific cryptographic tokens[20]. By supporting smart contracts based on arbitrarily selected methods of computation, blockchain

systems can be used to create custom ledgers that provide application-specific capabilities such as voting, administrative functionality, and fee payments.

Ethereum is a decentralized blockchain platform that has the capability to execute and deploy *smart contracts*. Ethereum's smart contract code is immutable and fully deterministic in its execution (unless non-deterministic behaviour is explicitly added). This allows any node to verify the execution of a smart contract and audit the resulting changes to application state. This enables Ethereum's smart contracts to run exactly as programmed.

Ethereum applications run atop a powerful shared global infrastructure. Applications can rapidly transfer value and represent ownership of assets, enabling software developers to create markets, store registries of debts or promises, move funds in accordance with rules created in the past — and more. All without a third party provider or counterparty risk.

Ethereum's capabilities are useful in general, nowhere more than in emerging markets that must contend with server downtime, corruption, and fraudulent behavior.

B.3.2. Ethereum and the Orchid Market ERC20 Tokens

Most tokens deployed on the Ethereum network conform to the ERC20 standard[45]. This standard specifies a compact and simple API for the transfer of tokens and metadata that can easily and quickly integrate tokens into hardware or software user wallets.

For example, the Augur[24] and Gnosis[25] platforms use ERC20 tokens to create prediction markets from token data. ERC20 enables arbitrary smart contracts to easily interface with the Orchid protocol. This can be valuable to IoT devices seeking to securely access Internet endpoints.

ERC20-compliance also makes it easier for token exchanges and applications to benefit from the expanded capabilities provided by new types of ERC20 tokens. This in turn enables different applications to use ERC20-compliant tokens to exchange information and status.