McGill University

Onion Routing: A Quest for Internet Anonymity, and Perhaps also Unparalleled Brilliance

ECSE 414 Final Report

Camilo Garcia La Rotta (260657037)
Dennis Liu (PUT ID HERE)
Stuart Mashaal (260639962)
Spiros-Daniel Mavroidakos (260689391)
Jastaj Virdee (260689027)
Harley Wiltzer (260690006)
Edward Zhao (260690376)

December 5, 2017

Contents

Introduction and Background

Suppose it is desired to communicate sensitive information across the Internet to a trusted source. Of course, it should not be possible for an arbitrary observer to read this information. The problem of hiding information can be solved by encrypting data, taking advantage of the advances in the field of cryptography, so that only the trusted source may be able to unscramble (and thus effectively read) the message that was sent. But what if this communication was so sensitive that it is even desired that no arbitrary observer should even know that the source and trusted destination are communicating with each other? This presents another interesting problem that goes beyond cryptography – in fact, it may only be solved by the design of a clever communication protocol.

The desire to participate in anonymous communication over the Internet is quite popular nowadays, and many choose to accomplish this by communicating through a Virtual Private Network (VPN). This works by sending all messages through an intermediate server, so that observers can only see that the sender is sending messages to the VPN server, and therefore cannot tell where the sender's desired destination is. Moreover, the destination itself is receiving messages from the VPN server, so observers monitoring the destination cannot tell who the sender is. On the surface, it may seem like this method does, in fact, provide a means of anonymous communication. Since no observer can know the source and destination of a message sent through a VPN, does that not imply that anonymity was achieved? Unfortunately, although arbitrary observers cannot discern the source and destination of a message, there is one party that may retain this information, and that is the VPN server itself. Users of VPN services must trust that the owner of the VPN server has no malicious intentions, and does not, for example, keep logs of the packets it receives. Although many users may trust the VPN servers they use, it is important to note that the VPN service does not, in fact, provide complete anonymity.

This report will explore and provide details of the implementation of the Onion Routing protocol, a method of communication that was shown to provide complete anonymity [1]. Onion Routing may be seen as an extension of the VPN scheme described above, which makes creative use of several intermediate servers to hide the endpoints of a communication channel from any potential observer. An Onion Routing network consists of a set of *onion routers*, denoted by $\mathcal{O}$, and a *directory node*. The directory node is responsible for knowing the details of all onion routers and selecting a random path of n onion routers, $\mathcal{P} \triangleq \{o_1, \ldots, o_n\} \subseteq \mathcal{O}$ for a user to communicate through. Given $\mathcal{P}$, the sender sends his message to o_1, who relays the message to o_2, and so on, until o_n receives the message and forwards it to the destination. Therefore, no intermediate node in $\mathcal{P}$ communicates with both the sender and the receiver of a given message, assuming $n > 1$ (note that, for $n = 1$, the scheme collapses to the VPN paradigm described above). However, so far, this scheme *does not* provide complete anonymity. Although no router in $\mathcal{P}$ directly communicates with the sender and destination, the routers in $\mathcal{P}$ must still know how to route a message across $\mathcal{P}$, meaning its messages must contain the addresses of all routers in $\mathcal{P}$. If a router knows $\mathcal{P}$, the owner of the router may trace the messages it forwards to the destination! Thankfully, Onion Routing solves this issue by layering encryption on messages in such a way that all routers in $\mathcal{P}$ can only know the address of the server before it and the server after it in the message's path. This prevents any router or observer from gaining full knowledge of $\mathcal{P}$, which effectively hides the endpoints of a communication channel.

It is up to the designer of the Onion Routing Protocol to come up with a way of designing the encryption scheme that hides $\mathcal{P}$ from observers. Ultimately, this scheme will be the backbone of the security of the protocol. In the implementation described in this report, this was accomplished by an elegant virtual circuit establishment procedure, which was inspired by a establishment process suggested by Reed, Syverson, and Goldschlag [2] but heavily modified, as will be described later on in the report.

All in all, the Onion Routing scheme ideally provides some drastic improvements to the VPN scheme with respect to security:

1. The virtual circuit establishment phase eliminates the possibility of any router or observer directly knowing both the sender and destination of a message
2. A large amount of available onion routers on the network provides severe difficulty of guessing a message path
3. If there is lots of traffic in the network, it is infeasible for an observer to try to trace the path of a message

Therefore, Onion Routing is a very promising technique for providing complete anonymity of a communication channel. However, it is also a drastically more complex protocol then communicating through a VPN, for example. The added complexity may cause many considerable design challenges, performance reductions, and practical limitations, which may ultimately affect its feasibility.

The remainder of this report will describe the implementation details of such a protocol, and will discuss the design challenges that were faced. Furthermore, a thorough *exposé* of the limitations of the protocol, as well as the costs of mitigating these limitations, will be given. Moreover, performance details such as the added delay imposed by multiple intermediate routers and encryption or decryption will be observed. Although the Onion Routing protocol is promising, it remains to be seen if it is powerful enough for practical use. This report aims to determine if the proposed Onion Routing scheme can be used in practice, and will provide discussion concerning the cost of improving its practicality and the practicality of anonymous communication altogether.

Methodology

Onion Routing is a general enough concept that particular implementations of it must, on their own, make and incorporate many design choices about communication protocol and software architecture. For this reason, this section of the report serves to explore the technical details of this implementation's protocol and architecture, as well as explain, where applicable, the design choices behind these details.

Protocol Design

Say some sender $\mathcal{S}$ wishes to send a message to some recipient $\mathcal{R}$ through this implementation of an onion routing network $\mathcal{O}$. First, $\mathcal{S}$ must establish a virtual circuit through $\mathcal{O}$ over which to communicate with $\mathcal{R}$. Establishing this virtual circuit is a two step process as follows:

1. Acquire a path $\mathcal{P}$ from the *directory node* $\mathcal{D}$
2. Negotiate symmetric encryption keys with each node $o_i \in \mathcal{P}$

Once $\mathcal{S}$ has established $\mathcal{P}$ in $\mathcal{O}$, it can then wrap a message M intended for $\mathcal{R}$ in an *onion* before sending the onion through $\mathcal{P}$ to $\mathcal{R}$. The exact details of the onion data structure will be illucidated shortly, but for now it shall suffice to say that an onion is a message that is incrementally encrypted by $\mathcal{S}$ such that each node o_i can decrypt the message before passing it along to o_{i+1}.

To begin the circuit establishment process, $\mathcal{S}$ must acquire $\mathcal{P}$ from $\mathcal{D}$. However, it must do so securely so that no observer can know $\mathcal{P}$. Fortunately, $\mathcal{D}$, just like every onion router in $\mathcal{O}$, has an assymetric keypair $(pub_{\mathcal{D}}, priv_{\mathcal{D}})$. So, $\mathcal{S}$ generates a random symmetric key $sym_{\mathcal{S},\mathcal{D}}$ for $\mathcal{D}$ to use to encrypt the path $\mathcal{P}$ that will be its response to $\mathcal{S}$. $\mathcal{S}$ then encrypts $sym_{\mathcal{S},\mathcal{D}}$ with $pub_{\mathcal{D}}$ and sends that to $\mathcal{D}$. $\mathcal{D}$ then decrypts with $priv_{\mathcal{D}}$ so that now both $\mathcal{S}$ and $\mathcal{D}$ share $sym_{\mathcal{S},\mathcal{D}}$. Now that it shares a symmetric key with $\mathcal{S}$, $\mathcal{D}$ constructs a path $\mathcal{P}$, uses $sym_{\mathcal{S},\mathcal{D}}$ to encrypt a message containing $\mathcal{P}$ and sends the encrypted message to $\mathcal{S}$. To complete its path acquisition, $\mathcal{S}$ decrypts $\mathcal{D}$'s response to get $\mathcal{P}$.

Having finished acquiring $\mathcal{P}$ from $\mathcal{D}$, $\mathcal{S}$ can move on to step 2 of the circuit-establishment process: negotiating symmetric keys with each node $o_i \in \mathcal{P}$. $\mathcal{S}$ negotiates symmetric keys with all $o_i \in \mathcal{P}$ just as it did with $\mathcal{D}$: by generating the symmetric key randomly and then encrypting it with its recipient's public key before sending it.

The unit of communication is the so called onion. It is a data structure that contains the clients message wrapped in successive layers of encryption. Its creation is handled by the Client node at the sending stage. At the reply stage the plain response is iteratively encrypted as it traverses the nodes in the path back to the original sender.

Reed, Syverson, and Goldschlag suggested a separate Onion Routing process for establishing the connections between all routers involves in a communication channel [2, 3], which was a very practical implementation idea that drastically simplified the routing of data messages and elegantly streamlined the implementation that will be shown. A *mélange* of symmetric and asymmetric cryptography (see Appendix A) was used to securely establish virtual circuits. The virtual circuit establishment process consisted of generating symmetric keys $\{s_1, \ldots, s_n\}$ and sending them to their respective onion routers encrypted with the desired onion router's public key, so that only the desired onion router may know its symmetric key. When an onion router receives its symmetric key, it maintains the connection through which the symmetric key was sent. The sender therefore sends n successive public-key-encrypted symmetric keys, and sends them all through o_1 to avoid the scenario of an observer deducing $\mathcal{P}$ by watching where the sender is sending messages. The sender sends the public-key-encrypted s_i keys encrypted successively with $s_{i-1}, s_{i-2}, \ldots, s_2, s_1$, and each node in $\mathcal{P}$ decrypts with its symmetric key before forwarding the establishment message. This hides the location of o_i until o_{i-1} receives the establishment message, at which point o_{i-1} decrypts the last layer of symmetric encryption and makes a connection with o_i. Once all connections are made, there is no longer any need to send addressing information for data messages. Therefore, the suggested implementation does, in fact, provide secrecy of the path $\mathcal{P}$.

As in the virtual circuit establishment, the sender encrypts the desired data message successively with the intermediate onion routers' symmetric keys when sending messages, and onion routers always decrypt forward-propagating messages (message towards the destination) with their symmetric keys, so that the intermediate routers cannot read the message that is being sent, and observers cannot trace a given message across the onion network. When the message's receiver sends back a response, all intermediate nodes encrypt the backward-propagating message with their symmetric keys before sending the message back towards the sender, which similarly hides the data in the response message.

In a favorable scenario, $|\mathcal{O}|$ is very large, and the amount of communication channels throughout the onion network is also very large. This provides even more security, as it makes things extremely difficult for observers to deduce the path of a message in the network. As the traffic in the network gets greater, it becomes practically infeasible for an observer to try to do this.

Say a Client wishes to send its traffic through a random ordered set of onion routers. First it must determine which routers will participate in its virtual circuit, and then it must set up a secure communications channel through them. It delegates the task of router selection to the Directory Node. When asked to do so by a Client, the Directory Node will construct a random ordered list of onion routers and their associated crypto keys. The Client will then receive this list of routers and their crypto keys, and will use it to incrementally establish the secure channel through the routers in the list. After it receives the acknowledgement from the three router nodes in the path it proceeds to create the data Onion such that the first node can peel the first encryption layer and so on.

From the perspective of the routers, the communication establishment process begins with receiving a symmetric key encrypted with its public key. It decrypts the symmetric key using its private key and responds with an ACK the node that sent it the message, who passes the ACK back to the client. From this moment the router will set two separate threads, the first one will handle the backwards communication, the second one the forwards communication. From the Destinations perspective, all received messages are in plain text and it will respond to all its parallel open TCP connections back in plain text. It is up to the Routers on the path to successively encrypt the reply Onion until it reaches the Client. Who can the peel them off in forward order with the symmetric keys obtained from the circuit establishment.

Architecture and Implementation

We leveraged Pythons socket and Threading library to avoid having to implement the lower level requirements of this project. The nodes that comprise the Onion network are: the Directory, the Router, the Client and the Destination. The directory and the router nodes sit in the inside of the network, only listening on predefined ports. Whereas the client and the destination sit on the edge of the network, being able to communicate on other ports. All the nodes in the network are multithreaded to support multiple parallel communication between different clients, destination pairs. The Directory node keeps track of participant router nodes and generates a random path of 3 nodes whenever a client node wishes to communicate through the network. The router node has 2 primordial function: to pass along the path a setup onion and to handle the bidirectional transfer of onions to the next and prior node in the path. The destination, nicknamed dummy destination, has the sole purpose of responding to any incoming message on an arbitrarily selected port. The client acts as the command line interface for the client using the network. Given a destination it will request a path from the Directory, setup the virtual circuit and build the onion to send to the destination. All the nodes on the network use a custom-made library for encryption, wich we named Stealth and provides convenient object for symmetric and asymmetric encryption. This implementation of the onion router is built upon TCP communication and relies on a globally defined socket for all internal nodes.

Results

Nodes

MAYBE THIS SECTION SHOULD BE IN METHODOLOGY???

Our onion router network has the following implemented:

1. Directory node: Responsible for the specifying a path/ series of router nodes to pass through when a message is sent and to also send the respective keys for the routers

2. Onion router node: Responsible for peeling and encrypting messages depending if the message is in the forward or backward channel. Depending on how the routers are placed in a path, when a message is sent from one router to another, the receiver will send an acknowledgment message to confirm it has received the correct message.

3. Client node:

    (a) This is the node that will send the layered encrypted message in the forward channel

    (b) The client node will also send a message to all the nodes given by the directory node to set up a channel for it to send a message to the destination node

    (c) When it receives the encrypted confirmation message from the destination node, it will then decrypt the message to allow to user to see

4. Destination node: This is the node that will receive the complete message also send back a confirmation message in the backward channel. This message will have a layer of encryption added on at every onion router node until it has reached the client node to be decrypted

Multiple Users

Once the onion network has been set up, it is capable of handling more than just one user/client. To show this feature, the first client must set up the whole network. Afterwards, anyone else simply has to set up a client node and they will be one the same network that the first client set up. All clients on this node will be able to send messages without any interruptions or errors occurring.

Message Size

1. The maximum message size that can be sent in our network is 833 characters, this can be easily changed by simply modifying a small portion of the code.

2. The message size also correlates to another feature of encryption. Our goal was to make it more difficult for an outside observer to be able to distinguish who is the destination and who is the originator by keeping the message at a constant size. In order to do see every time a layer of encryption is peeled away, our application will add a layer of padding such that it is difficult to see when it may arrive at the originiator.

Round Trip Time of Messages

Although the use of onion routing increases security and anonymity, there is a concern regarding the time needed for a message to be send and received. The reason for this is instead of the message going from the client to destination and back, it must now also pass through several other nodes. Another reason why timing may be a concern is the fact that the encrypting and decrypting process may take some time.

To go about determining whether the onion routing system took a long time to send and receive messages, we compared the timing taken by the onion routing system and checked whether it was noticeably longer than the timing of a normal TCP connection.

To compute the average time taken by the onion routing system to send and receive a message, the code of the system was edited so that it would print the times corresponding to when the message was sent and received. The sent time was then subtracted from the received time to determine how long it took for the message to reach the destination and return to the client. Ten messages were sent, and the average was taken of these times to determine the average time taken by the onion routing system. The messages were of various sizes to ensure a more complete testing.

To compute the average time taken by a normal TCP connection, a simple python script was created to send messages from a client to a destination. This was the script "TCPClient.py" shown in THE TEXTBOOK. However this script was edited so that it also printed the times where the messages were sent and received. Once again ten messages were sent, and the average was computed of these ten times. The ten messages that were sent were the exact same as the 10 messages that were sent on the onion network. This was to ensure that the only variable determining the average time was the type of connection (onion vs. TCP).

The following graph and table demonstrate the data collected:

Message	RTT TCP	RTT Onion
hi	0.000942	0.004191
hello	0.000875	0.004517
hello world	0.000814	0.004243
hello newman. Jerry	0.001639	0.004739
hello how are you	0.00202	0.004359
bye	0.001519	0.004865
hello how are? Good thanks how are you?	0.001353	0.00534
hi	0.001274	0.004572
⟨alphabet⟩	0.00089	0.00501
⟨alphabet x3⟩	0.00081	0.004543
Average	0.0012136	0.0046379

ADD EXCEL GRAPH

As one can see the difference between the round trip time of the onion system and the normal TCP system is about 3 milliseconds. This is not perceptible to humans so it can be confirmed that timing does not have to be sacrificed for the increased security and anonymity of the onion routing system.

Appendices

A Brief Conspectus of Cryptography

The Onion Routing protocol described in this report makes extensive use of cryptography for encrypting and decrypting sensitive data. Encryption is the process of obscuring data in such a way that only the intended recipient of the data may read the data, via decryption. In order for this to be accomplished, the sender and the receiver of the data use calculated *keys*, which are similar in purpose to passwords, to compute the scrambled data and to retrieve the unscrambled data. Two main paradigms for achieving encryption and decryption are used in this implementation, and will be described below.

Terminology

The list below describes some terms related to cryptography that may be used sporadically in the report.

- **Plaintext**: Regular text that has not been encrypted.
- **Ciphertext**: Scrambled, incomprehensible text that is the product of encryption. Must be *decrypted* to be understood.
- **Encryption**: The calculated process of scrambling data to create ciphertext in such a way that it can be unscrambled only by a desired party.
- **Decryption**: The process of unscrambling ciphertext into plaintext.
- **Key**: A byte string that ultimately cryptographically identifies a communicating party, which is used to encrypt and/or decrypt data.

Symmetric Cryptography

The simpler form of cryptography that will be used is called "Symmetric Cryptography", and is characterized by the notion of the sender and receiver sharing a secret key, called a symmetric key. The scheme is defined by the tuple $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$. The KGen parameter is pseudorandom generator that generates a key $k \in \mathcal{K}$ with uniform probability over $\mathcal{K}$, where $\mathcal{K}$ is the keyspace, or the set of all keys. $\mathsf{Enc}, \mathsf{Dec}$ are families of functions, where $\mathsf{Enc}_k : \mathcal{M} \to \mathcal{C}$ and $\mathsf{Dec}_k : \mathcal{C} \to \mathcal{M}$ are encryption and decryption functions for some key $k \in \mathcal{K}$, $\mathcal{M}$ is the message space (set of all possible messages), and $\mathcal{C}$ is the ciphertext space (set of all possible ciphertexts). Symmetric schemes are called symmetric because encryption and decryption are both done using the same key, so for all messages $m \in \mathcal{M}$,

$$\mathsf{Dec}_k(\mathsf{Enc}_k(m)) = m \tag{A.1}$$

This property is crucial for providing bidirectional encrypted communication without revealing the message's route, so symmetric encryption and decryption is a primary function of the onion routers. Furthermore, symmetric key encryption and decryption are asymptotically much faster than their asymmetric counterparts [4] (described in the next section). The downside, of course, is that establishing a shared secret key and keeping it secret may be difficult.

The symmetric cryptography scheme that will be used in the Onion Routing implementation described in this report is called Advanced Encryption Standard (AES) [5]. The 128-bit key variant is used for ease of development, especially in the testing phase, but the `stealth` library developed for this implementation provides accomodations for easily changing the key size to 192- or 256-bit keys. It is generally frowned-upon to manually implement cryptographic schemes, as it is always preferable to use libraries that have been thoroughly tested and validated. The PyCrypto library was used to provide primitive AES encryption and decryption, as well as cryptographically-secure random key generation, in this implementation.

Asymmetric Cryptography

Although symmetric cryptography certainly has its benefits, it requires two parties to share a secret key, which may not be easily done, especially if the key must be communicated over a network. Asymmetric cryptography solves this issue, though it has issues of its own which will be described soon.

Instead of two parties sharing a shared secret key as in symmetric cryptography, asymmetric cryptography schemes give each party both a *private key* and a *public key*. As the names suggest, the public key can be known by anyone, but the private key should be kept secret. Note however that although the private key must remain secret, it is not shared by anyone, which removes the main drawback of symmetric encryption. Like symmetric encryption, asymmetric encryption schemes are defined by $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$, with $\mathsf{Enc}_{pub} : \mathcal{M} \to \mathcal{C}$ and $\mathsf{Dec}_{priv} : \mathcal{C} \to \mathcal{M}$. The encryption/decryption process has the following property:

$$\mathsf{Dec}_{f(k)}(\mathsf{Enc}_k(m)) = m \tag{A.2}$$

In equation (A.2), k is the public key, and $f(k)$ yields the private key. Asymmetric cryptography is only effective if computing $f(k)$ is an NP problem – that is to say, computing $f(k)$ is computationally infeasible. Given this property, if user $\mathcal{A}$ wants to send an encrypted message to $\mathcal{B}$, user $\mathcal{A}$ encrypts the message with user $\mathcal{B}$'s public key (which is publicly-known), and then $\mathcal{B}$ may decrypt the message using its private key. Since $f(k)$ is computationally intractible, the probability of someone other than $\mathcal{B}$ decrypting the message is negligible.

In the implementation of Onion Routing discussed in this report, asymmetric cryptography is used to communicate symmetric keys. Since public keys are publicly-known, the sender knows how to send encrypted messages to any onion router without needing a shared secret key. Therefore, asymmetric cryptography allows the sender to share symmetric keys without anyone aside from the desired router being able to see the symmetric key.

Although asymmetric cryptography does provide some incredible benefits, it also has its drawbacks. Firstly, asymmetric encryption and decryption is relatively slow [4], so it is beneficial to reduce the frequency of its use as much as possible without comprimising the security of the system. Furthermore, in some applications, such as the main purpose of onion routers, for example, it may be necessary to send encrypted *and* decrypted messages over the network. In this scenario, asymmetric cryptography cannot be used. In the case of the onion routers, they must encrypt response messages before forwarding them back towards the sender. Since the sender does not know the onion routers' private keys, the sender cannot decrypt the response! Therefore, clearly asymmetric encryption cannot be used in some scenarios.

The cryptographic scheme that is used in the Onion Routing implementation presented in this report is called RSA [6]. As in the case of symmetric cryptography, it is always preferred to use trusted libraries for implementing asymmetric cryptography. The PyCrypto library also implements the RSA cryptosystem, and it was used in the implementation described in this report.

List of External Resources

Several external, third party resources were used in the implementation described in this report. Below is a list of these resources, as well as references to where they can be found/downloaded and descriptions of what they were used for.

Resource	Origin	Reason for Use
PyCrypto python library	https://www.dlitz.net/software/pycrypto/	AES encryption/decryption, RSA encryption/decryption, importing and exporting of RSA keys, random byte string generation, pseudorandom number generators

Bibliography

[1] J. Camenisch and A. Lysyanskaya, "A formal treatment of onion routing," in *Annual International Cryptology Conference*, pp. 169–187, Springer, 2005.

[2] M. G. Reed, P. F. Syverson, and D. M. Goldschlag, "Onion routing network for securely moving data through communication networks," July 24 2001. US Patent 6,266,704.

[3] D. M. Goldschlag, M. G. Reed, and P. F. Syverson, "Hiding routing information," in *International Workshop on Information Hiding*, pp. 137–150, Springer, 1996.

[4] M. Agrawal and P. Mishra, "A comparative survey on symmetric key encryption techniques," *International Journal on Computer Science and Engineering*, vol. 4, no. 5, p. 877, 2012.

[5] J. Daemen and V. Rijmen, "Announcing the advanced encryption standard (aes)," *Federal Information Processing Standards Publication*, vol. 197, 2001.

[6] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.