Anonymous Communication using Onion Routing

Aman Bansal
Syamantak Kumar
November 2018

Contents

1 Abstract .................................................. 2

2 Introduction ............................................. 2
   2.1 Mix Networks ....................................... 2
   2.2 Onion Routing ..................................... 2
       2.2.1 Basic Details .................................. 2
       2.2.2 Terminology ................................... 3
       2.2.3 Routing Overview .............................. 3

3 Specifications .......................................... 4
   3.1 Proxies .............................................. 5
       3.1.1 Application Proxy .............................. 5
       3.1.2 Onion Proxy .................................... 6
   3.2 Entry and Exit Funnels ............................ 6
   3.3 Onions .............................................. 7
   3.4 Onion Router Interconnection ..................... 8

4 Threat Model ........................................... 9
   4.1 Security Goals .................................... 9
   4.2 Adversary Model .................................. 10
   4.3 Possible Side-Channel Attacks ................... 11

5 Security Analysis ...................................... 11
   5.1 Remote-COR Configuration ....................... 12
   5.2 Local-COR Configuration ......................... 13
1 Abstract

Most of the works in cryptography mainly focus on preventing eavesdropping and providing integrity. However, these schemes do not provide any means for anonymous communication. Anyone can easily know who is talking to whom and there is no way to communicate anonymously. Tracking such information is known as traffic analysis. Onion Routing was developed to provide a way for communicating anonymously. We will now explain what exactly is Onion Routing, its specifications, and its security analysis. We thoroughly studied [1] and [2] to build up this report and the presentation.

2 Introduction

2.1 Mix Networks

Mix Networks were first described in a seminal paper on anonymous email by David Chaum in 1981 [3]. A mix network is a routing protocol that uses a chain of proxy servers to make a communication hard to trace. Each message is first encrypted under a sequence of public keys. Each mix node first decrypts the message using its secret key and then finds which is the next node and then sends the message to it.

![Figure 1: Example of a mix network](https://upload.wikimedia.org/wikipedia/commons/4/4f/Red_de_mezcla.png)

Various senders send messages to a particular mix node and the node keeps collecting the messages and decrypting them and after it has collected a sufficient number of messages it shuffles all the decrypted messages before sending them to the next destination (possibly to the next mix node). This breaks the link between the message received by the node and the message sent by the node. This makes tracing of the end-points of a communication hard.

2.2 Onion Routing

2.2.1 Basic Details

Onion Routing (OR) is based on the concept of mix networks. It uses what is called an onion routing network to provide anonymous communication between applications who want to communicate with each other. An OR network is very similar to a mix network and consists of specially designed machines called onion routers which correspond to the mix nodes in the mix network. These are interconnected with each other using longstanding (permanent) connections, as shows in Figure 2b. From now on, we will refer to onion routers as just routers and will specify whenever we mean normal network routers. The job of
these routers is quite similar to the job of mix nodes except that they have to be real time and hence cannot keep holding the messages like mix nodes. As we will see, OR uses synthetic traffic to overcome this.

![Diagram](image)

**Figure 2: Difference in public network and OR network**

Source: [4]

### 2.2.2 Terminology

We will now discuss some of the technical terms used in the onion routing literature (details are given in Section 3):

- **Initiator**: The application which initiates the connection.
- **Responder**: The application to which the connection is requested.
- **Forward Direction**: The path from the initiator to the responder.
- **Backward Direction**: The path from the responder to the initiator.
- **Application Proxy**: An application specific translator whose job is to take application data and convert it into an application-independent format of fixed length cells which can be sent in the onion routing network. After conversion, it passes the cells to the onion proxy.
- **Onion Proxy**: An onion management layer which determines the path which a particular connection data cells are going to take and then takes the fixed length data cells from application proxy and sends them accordingly.
- **Onion**: A fixed-size multi-layered data structure. Each layer of the onion contains encrypted information meant for a particular router. The layers are ordered according to the sequence of routers which will be followed, i.e., the outermost layer corresponds to the first router and the innermost layer for the last router. Each router, after receiving an onion, peels off its the outermost layer and decrypts it. The decrypted layer contains a lot of information about which cryptographic protocol to follow, which router is the next hop etc.
- **Entry Funnel**: A special router which multiplexes the connections from onion proxies to the onion routing network.
- **Exit Funnel**: a special router which multiplexes the connections from onion routing network to the onion proxies.

### 2.2.3 Routing Overview

There are three phases for any communication:
• **Setup:** For the connection setup, the initiator’s onion proxy has to first decide upon the sequence of onion routers through which all the messages will be sent. This path is fixed and can’t be changed during the course of the communication. The proxy then builds an onion from the data and adds layers to it. The layers are added for each router in the path. The outermost layer corresponds to the first router while the innermost layer for the last router. Each layer contains information meant for that particular router such as the cryptographic algorithms to be followed while transferring the data, the next hop etc. and is encrypted with the public key of the router. It then sends it over the entry funnel. Each router after receiving an onion, peels off its outermost layer and decrypts it using its private key and then proceeds according to the information present in the layer. Before sending to the next hop, the router pads this onion to maintain its fixed size.

After the last router in the path sends the completely peeled onion to the responder’s onion proxy, the onion proxy converts the onion to fixed-size cells and sends it to the application proxy which combines these cells to form the original data stream and sends it to the application.

• **Data Movement:** The connection is already established in the setup phase. The path is fixed and every router in the path knows what cryptographic algorithms and keys it has to use for this communication. Now for sending the data in the forward direction, the onion proxy divides the data into **DATA cells** (details in Section 3.4), encrypts the data for each layer in order (using the cryptographic algorithm and key corresponding to that layer’s router) and sends it. Each router then decrypts the layer accordingly and the final plaintext data reaches the exit funnel and is transferred to the responder’s onion proxy. This is shown in Figure 3a.

![Figure 3: Data Movement](source: [4])

For the backward direction communication, the procedure is exactly opposite to the one we followed for forward direction. This is shown in Figure 3b. The responder application sends the message to the application proxy which in turn sends it to the onion proxy. The onion proxy creates DATA cells from the data and passes it to the exit funnel without encrypting anything. Every onion router through which these cells pass already knows the next hop for this reverse-direction communication from the setup phase and hence can transfer this onion correctly. Before sending the onion, each router encrypts the data in the cells. Finally, when the initiator’s onion proxy receives the onion, it decrypts it multiple times to get the original data.

• **Termination:** Either end of the communication can terminate the connection. However, since we are fixing the path of the communication, it may be possible that an onion router in the path gets failed in which case the connection will have to be closed. For both sides, it will appear as if the other side has closed the TCP connection.

3 Specifications

In this section, we will give detailed specifications of each component involved in Onion Routing.
3.1 Proxies

A proxy is a transparent interface between two applications which should make a direct socket connection to each other but cannot. For examples, a firewall might not allow direct socket connections between internal and external machines due to security reasons and a proxy running on the firewall has to be used. OR uses two types of proxies: application proxy and onion proxy.

3.1.1 Application Proxy

Since it’s not feasible to either ask every application to change its way of communicating when it is using an OR anonymous connection nor to make OR anonymous connection interface specific for every application, we will have to use a proxy which is application specific and can interface with the standard OR’s anonymous connections. This proxy is known as the application proxy. As is clear from its use, it must be application specific and must understand both application protocols and OR protocols. This proxy allows the implementation of the onion proxy, entry funnel, exit funnel to be application independent. Its interfacing details are as follows:

- **Connection Setup:** The proxy waits for a connection setup request from the application. Upon receiving a request, it first decides whether it will handle or deny the request. If denied, then it will send an application-specific error code to the application and wait for the next request. If accepted, it will make a socket connection to the onion proxy's well-known port. Before sending any data messages, it first sends a standard structure as shown in Figure 4, followed by the ultimate destination address and waits for an error code to be received from the exit funnel. It doesn’t send any data before receiving the error code. If the received error code corresponds to success then it continues normally else it shows an error to the application in a way that the application can understand.

The details of each field in the layer are as follows:

- **Version:** Denotes the version which is being used. In this report, whatever we discuss from now on will be for the first version.
- **Protocol:** Denotes the protocol being followed for the communication. It is 1 for RLOGIN, 2 for HTTP, or 3 for SMTP.
- **Retry Count:** Specifies the number of times the exit funnel should retry to connect to the ultimate destination.
- **Addr Format:** Defines the format in which the destination address is given. Set to 1 for NULL terminated ASCII string with the hostname or IP address (in ASCII form) immediately followed by another NULL terminated ASCII string with the destination port number. All other values are undefined.

- **Data Movement:** After the connection is successful, it starts the task of transferring data. At the sender’s end, it takes the data stream from the application, converts it into fixed-size (independent of the application) data cells and sends it to onion proxy. This is shown in Figure 5. At the receiver’s end, it receives the data from the onion proxy in the same form of fixed-size data cells.
and converts it into data streams according to the requirement of the application. One optional task is to implement a privacy filter to sanitize the data before sending it over the OR network.

- **Termination**: If the connection is terminated, it passes the appropriate error code to (or from) the application.

### 3.1.2 Onion Proxy

Thanks to the application proxy, onion proxy can be made application independent. It waits for a request from the application proxy and on receiving such a request, it first decides whether it wants to serve the request (based on parameters such as destination details, application proxy id etc.). If rejected, it sends an appropriate error to the application proxy, closes the socket and waits for the next request. If accepted, it proceed as follows:

- **Connection Setup**: It selects a path for reaching the final destination, builds an onion encoding the path information and passes this onion to the entry funnel as shown in Figure 5. The onion mentioned above establishes the anonymous connection, sets up the key values for encryption/decryption at each router and an ACI (see Section 3.4) for each link in the connection. It then sends the standard structure and the destination address to the exit funnel over the anonymous network.

- **Data Movement**: After connection setup, it just relays the data between the OR network and the application proxy without doing anything except for changing the format appropriately.

- **Termination**: The application proxy after receiving the termination error code, closes the socket with the application proxy.

![Figure 5: Overview of a OR network w.r.t. a particular anonymous connection](http://ntrg.cs.tcd.ie/undergrad/4ba2.05/group10/index.html)

### 3.2 Entry and Exit Funnels

Entry Funnel is just a router with an additional functionality of multiplexing the connections from various onion proxies to the OR network. Every entry funnel is associated with some routers and whenever any onion proxy has to send an onion on the network, it opens a socket connection with the entry funnel of the first onion router and sends the onion to it which in turn sends it to the onion router and hence the onion enters the OR network.
Exit Funnel is just the opposite. It multiplexes the connections from OR network to various onion proxies. Whenever any onion router receives an onion which specifies 0 as the next hop’s address and the port, it knows that it is the terminal router and passes the onion to its exit funnel. The exit funnel will receive the standard structure as the first messages of the connection. It looks up the destination’s IP address and port and tries to establish a socket connection with it. It then returns an appropriate 1-byte error code back to the network (which reaches the initiator’s onion proxy and then the application proxy) about failure or success of the connection. For the rest of the data (assuming a successful connection), it just relays the data between the last router and the onion proxy.

### 3.3 Onions

As we have already discussed, an onion is a multi-layered data structure which encodes the path which the communication is going to follow. The outermost layer of the onion is meant for the first router just after the entry funnel and the last layer for the router just before the exit funnel. Each layer is encrypted using the public key of the router for which it is intended. The structure of a layer is shown in Figure 6.

![Structure of the Onion layer](image)

The details of each field in the layer are as follows:

- **The first bit:** As we will see, this bit is always set to 0 for RSA to succeed.
- **Version:** The version number, currently defined to be 1.
- **‘Back’ field:** The cryptographic function to be applied to the data moving in the backward direction using key
  $\text{key}_2$ (defined below).
- **‘Forw’ field:** The cryptographic function to be applied to the data moving in the forward direction using key
  $\text{key}_3$ (defined below).

The cryptographic functions defined in Version 1 are 0 for Identity (no encryption), 1 for DES OFB and 2 for RC4.

- **Destination Address and Port:** The details of the next onion router to which the peeled onion should be sent. Both are 0 for exit funnel.
• **Expiration Time:** The time is given in seconds (according to standard UNIX time format) and specifies till when should the onion track to avoid replay attacks.

• **Key Seed Material:** The seed for finding the keys to be used in the cryptographic function. It is a 16 Bytes long field. In version 1, this seed was hashed three times using SHA to produce 3 keys \((key_1, key_2, key_3)\) of 128-bits each (first 8 bytes used for DES and 16 for RC4). Details on these cryptographic operations can be found in [5] and [6].

RSAs with 1024 bit modulus size was used for RSA in version 1 and hence the plaintext block size should be less than that 1024 bit. That's the reason behind appending that compulsory 0 bit in the message. The procedure of forming the multi-layered onion after the onion proxy decides the path is as follows:

1. Currently the onion consists of 100 Bytes of randomly chosen data.

2. For each layer in order from innermost to outermost:
   
   (a) Prepend the layer to the onion
   
   (b) Use the RSA key public key of the router to which this layer corresponds to encrypt the first 128 Bytes of the onion
   
   (c) Encrypt the remainder of the onion using DES OFB with an IV (Initialization Vector) of 0 and \(key_1\) defined above

### 3.4 Onion Router Interconnection

The adjacent routers communicate using the long-standing link between them (note that this link is itself made up of various normal network routers and hence can take different paths for different messages belonging to the same connection). During network setup, each onion router is assigned a public-private key pair, all the longstanding links are established and keyed and each onion router knows its neighbor.

The protocol for opening a connection with neighboring routers has two phases: connection setup and keying. The initiating onion router opens a socket to a well-known port of its neighboring router and sends its IP address and well-known port (note that using the port allows multiple onion routers to run on a single machine). This is followed by the keying phase. Secure Token Service (STS) [7] is used to get two DES 56-b keys. The link encryption over the link is done by using DES OFB as the encryption scheme using these keys (one for each direction).

After successful keying, data can be transferred between these two onion routers. To allow more multiplexing of messages, the data is divided into fixed sized **cells** which are sent in order and then collected and combined at the receiving end. The cell structure is shown in Figure 7.

![Figure 7: Structure of a cell](source: [1])

There are 4 types of cells. The Anonymous Connection Identifier (ACI) field is the identifier of a connection over a particular longstanding link. It is not globally unique but unique for a particular link. The command field contains the type of the cell. The ACI and the Command field are always encrypted (independent of the type of cell) using the link encryption scheme. The types of cells are:
• **CREATE:** Length field is link-encrypted while the payload is already encrypted. Create cells are used when a new connection is being created. The router peels off the outermost layer of the onion to find the neighbor to which it should create a connection. It then chooses a new ACI for this connection over the link and keeps a mapping which maps the incoming link ACI to the outgoing link ACI. Since both the routers share the same ACI space for that link, the protocol requires that the router with the higher IP/port chooses the ACI values in the top-half of the address space while the other chooses ACI values in the lower-half of the address space. This is done to avoid states where both the routers choose the same ACI. After this, the initiating router divides the peeled onion (padded with random bits to not leak the size) into various CREATE cells (the last cell will need padding) and sends these cell over the link in order.

• **DATA:** Both the Length field and the Payload field are encrypted using the cryptographic operations defined by the onion. The data is sent over the OR network in DATA cells. For the forward direction, the length and payload field is repeatedly encrypted using the keys defined by the onion initially and as the data cell goes through the network, each router decrypts using the key it has and forwards the cell. At the exit funnel, the length field and the payload field is the actual data and its length. The exact reverse happens in the backward direction.

• **DESTROY:** This cell is sent when any connection has to be closed because it got terminated by either the communicating end-points or due to failure of any router in the path. This is shown in Figure 8. Both Length and Payload fields are link-encrypted. If a connection gets broken, this cell is sent to clean up the states of all routers. The ACI field of this cell contains the ACI field of the broken connection. After receiving a DESTROY cell, the router must acknowledge the sender router and pass the cell to the next router after appropriately changing it. If the acknowledgment of receiving a DESTROY cell by the next router is successfully received by a router, then the router must not send any more cell on that connection and can remove the ACI mappings and use it for some other connection.

• **PADDING:** Both Length and Payload are link-encrypted. This is a useless cell and is used only to inject data to further confuse traffic analysis. They are thrown away on receipt.

![Figure 8: DESTROY cells as a result of Y getting failed](Source: [4])

4 Threat Model

4.1 Security Goals

We want both anonymous as well as private communication and hence we define the following as the goal which our scheme must achieve in order to be called anonymous and private scheme.

• **Sender Activity:** The knowledge of the fact that the sender has sent something

• **Receiver Activity:** The knowledge of the fact that the receiver has received something
These two properties are to ensure that the scheme used is anonymous. No one can identify that a particular sender sent something and hence can’t get the knowledge of his activity.

- **Sender Content:** The knowledge that the sender sent a particular content
- **Receiver Content:** The knowledge that the receiver received a particular content

These two properties are to ensure that the scheme used is private. No one can get any information about the content that the sender sent/receiver received. If the message was end-to-end encrypted then there is no ‘actual’ content revealed but for the perspective of onion routing, we consider that as the content leak.

Note that both anonymity and privacy are independent security goals. A scheme can be anonymous without being private or private without being anonymous.

- **Source-destination Linking:** The knowledge that a particular sender is sending something to a particular receiver.

This property states that it is untraceable who communicates with whom. This fifth property is weaker than the sender or receiver anonymity as it may be possible to trace who sent which messages as well as who received which messages but there is no link between any message sent and any message received. Onion Routing does provide this unlinkability in some configurations but this is not a general goal for all communications.

### 4.2 Adversary Model

In this section, we discuss the powers that we allow the adversary to have. This is a very critical part in analyzing the security of any scheme. These powers essentially determine what all we must include in our scheme and what can be sacrificed for more efficiency.

Here are the adversaries which we will consider:

- **Observer:** Can only observe a connection and can’t affect the connection in any possible way.
- **Disrupter:** Can disrupt the communications on a particular link but can’t observe the message going through the link. For example, delaying of packets for an indefinite amount of time, dropping of traffic, corruption of messages etc.
- **Hostile User:** Can initiate or destroy a connection with a particular route. The traffic which is sent over this route can be arbitrarily adjusted.
- **Compromised Core Onion Router (COR):** The adversary controls one of the CORs of the network. Hence, the adversary gets some information about the connections passing through this COR and can manipulate the connections arbitrarily (including new connections which pass through this router).

The adversaries which we will consider can be a combination of any of these basic adversaries. For example, many hostile users communicating with each other or any other honest user over an onion routing network containing various compromised CORs and many observers and disrupters working on different links.

**Claim:** Proving the security of the network w.r.t. the adversaries which are composed of one or more CORs is sufficient (pessimistically) for proving the security of the network w.r.t. all the adversaries formed by various combinations of the above 4 basic adversaries.

**Proof:** It is easy to see that if we have an observer or disrupter over a link and we remove those observers and disrupters and compromise one of the end-point CORs of the relevant link (if not already compromised) then we get a stronger adversary. The effect of the hostile user can be modelled by compromising the specific set of routers that the hostile user is using for it’s adversarial analysis. Therefore, a hostile user can also be replaced by a set of CORs. Hence proved.
Now since we only have to focus on this most capable adversary, the one with one or more compromised CORs, we further categorize this class of adversaries:

- **single adversary**: only one compromised COR
- **multiple adversary**: A fixed, randomly distributed proper subset of the CORs is compromised
- **roving adversary**: There is a fixed upper bound on the number of adversaries which can be compromised at a particular time. Note that at different points in time the set of compromised CORs can be different.
- **global adversary**: All COR’s are compromised.

Before proceeding further we would like to inform that the onion routing doesn’t provide any security guarantee against the global adversary. If all the CORs are compromised, the adversary can easily find out who is the initiator and the responder of any connection. The content transferred on the connection can be seen at the last router when the message is passed to the responder.

### 4.3 Possible Side-Channel Attacks

Here we will discuss some possible ways to extract information from the OR network which is not expected to be revealed. A *session* refers to the data carried over a single anonymous connection.

- **Marker attack**: A *marker* is basically a data which upon being sent generates a observable pattern in the encrypted traffic. Various types of markers have been identified till now, for e.g. web-page markers [8] which use the predictability of encrypted traffic generated from marked web-pages. Using these markers, we can identify the set of outbound connections that some distinguished markers may have been forwarded upon. Sending multiple markers on the same session can led us to reduce the set of possible next hops for that connection.

- **Timing attack**: Each compromised router tracks the data rate of a particular session (called *timing signature* of that session) and if two colluding nodes have similar data rates distribution for some connection then with high probability they belong to the same connection.

- **Volume attack**: Colluding router can keep track of the number of cells passed over a given session and if the cell totals come out to be very close then they most probably belong to the same session.

### 5 Security Analysis

As discussed before, we will only consider the roving adversary. Our adversary is characterized by $c$, the bound on numbers of CORs which can get corrupted. We will make the following assumptions for the analysis:

- Each next hop is randomly chosen (except that cycles of length 1 are not allowed). This means that the path from the sender to the receiver is a random walk.
- The CORs which were affected in the previous round but are unaffected now are assumed to be instantly 'healed'.

**Notation:**

- $C_i$ denotes the set of CORs which are compromised in the $i^{th}$ round.
- $r$ denotes the total number of CORs in our network
- $n$ is the (variable) length of the route $R = \{R_1, R_2, ..., R_n\}$.

We will do security analysis in 2 configurations : Remote-COR and Local-COR.
5.1 Remote-COR Configuration

In this configuration, the user has remote access to the first COR in the route. It has to transfer everything to the first route using some secure communication method.

For the $i^{th}$ round, there are 3 cases possible:

1. $R_1 \notin C_i$ and $R_n \notin C_i$ The adversary learns nothing since it is not possible to infer the sender, receiver or the content of any particular connection that the compromised router is not a part of given only the incoming and outgoing cells.

2. $R_1 \in C_i$ The first node is compromised. Only the sender activity is compromised. Sender content is not compromised because the sender always pre-encrypts traffic. The probability of this happening assuming random route selection and random node compromises is $P_1 = c/r$.

3. $R_n \in C_i$ The last node is compromised. Only the receiver activity and the receiver content is compromised. Probability of this happening is $P_2 = c/r$.

4. $R_1$ and $R_n \in C_i$ Both the first node and the last node are compromised. Sender activity, receiver activity, receiver content is compromised. Moreover, this allows the adversary to do various attacks like counting the number of cells sent by the sender and received by the receiver etc. This compromises source-destination linking. Consequently, sender activity is compromised as it can be inferred by the receiver’s content. Probability of this happening is $P_3 = c^2/r^2$ for $n > 2$ ($P_3 = c(c-1)/r^2$ for $n = 1$ as 1 length cycles are not allowed).

<table>
<thead>
<tr>
<th></th>
<th>$R_1 \in C_i$</th>
<th>$R_n \in C_i$</th>
<th>$R_1$ and $R_n \in C_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>sender activity</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>receiver activity</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>sender content</td>
<td>No</td>
<td>No</td>
<td>Yes (inferred)</td>
</tr>
<tr>
<td>receiver content</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>source-destination linking</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>probability</td>
<td>$c/r</td>
<td>$c/r</td>
<td>$c^2/r^2$</td>
</tr>
</tbody>
</table>

Figure 9: Attack scenarios and their properties

Source: [2]

Figure 9 summarizes the result of the above findings. We establish that the goal of adversary should be to compromise the first or the last or both routers. Since many connections are short-lived (like email, web surfing), they will rarely last for more than one round and hence $P_3$ gives us the probability that the adversary compromises all properties of a connection. So we will consider long-lived connections now.

At route-setup time, the probability that at least one COR on the route of length $n$ is in $C_i$ is given by

$$1 - P(R \cap C_1 = \phi) = 1 - \frac{(r - c)^n}{r^n}$$

We can assume that if the adversary doesn’t compromise a node which is in the route then he will not attempt the attack.

If the adversary compromises a node which is in the route then it can use timing analysis of route setup and other attacks to find the compromised nodes which are closest to the source and farthest to the route (they maybe the same). Then in each round, the adversary uses timing analysis and other attacks to find out the next hop in the route of the current farthest compromised node and previous hop of the current closest compromised node. So the adversary can reach both $R_1$ and $R_n$ after some number of rounds (linear in route length). The adversary may also apply the heuristic of randomly selecting the rest of $c - 2$ nodes to find if he gets any closer to $R_1$ or $R_n$. 
5.2 Local-COR Configuration

In this configuration, the user owns a COR and uses it as the first COR. Since the user can protect the router it owns by using various means such as firewall, we assume that the first and the last routers in the path can’t be compromised. This essentially means that the adversary can’t compromise any of the security property we have defined in this configuration.

References


