The Anonymity Ring

By

Muqtadar Ahmed

Advisor: Mohamed G. Gouda (gouda@cs.utexas.edu)

Department of Computer Sciences

The University of Texas at Austin

http://www.cs.utexas.edu

Computer Science Research and Writing

CS 379H

Fall 2002

Abstract

In the vast amount of practical applications of the internet, there exists a class of applications which require anonymous communications. Towards the beginning of this paper (Sec. 2 and Sec. 3), we briefly discuss two closely related existing methods of anonymous communications, Mixer Networks and Onion routing, and present their advantages and limitations. Following this, we explain our method of anonymous communication, termed *Anonymity Ring* (AR)*,* and proceed to develop its formal specifications using Abstract Protocol [1] notation. In conclusion, we provide advantages and limitations of anonymity ring with other protocols and present areas in which further research could proceed.

The Anonymity Ring

1. Introduction

a. The issue

With the rapid and unprecedented growth of the Internet, it is being utilized in diverse tasks, from mere electronic communication among individuals to e-commerce. With these wide application domains comes the need for *secure communications*[1] Furthermore, certain communications require that not just the content, but also the fact that a communication exists be hidden. The above two situations are termed a*nonymous communication* and a*nonymous connections*, respectively.

In this paper, we seek to develop and present formal specifications, using the *AP notation* [2], of a protocol that provides anonymity in the aforementioned situations. This protocol, termed *the anonymity ring*, presents a communication system that presents a solution to the abovementioned problems without trusting a single third party. The anonymity ring incorporates various ideas from *Mixer Networks* and *Onion Routing*, discussed in section 2 and 3, respectively.

b. Outline

We start out with a brief introduction of mixer networks in section 2, followed by an introduction of onion routing in section 3. A discussion of the structure, organization and functionality of the anonymity ring follows in section 4 and 5. We then present the complete process specification of the anonymity ring protocol in section 6. Section 7 presents various configurations for message buffering system and discusses the advantages and disadvantages associated with each. A brief analysis along with implementation details is presented in sections 8 and 9. At the end of this paper, various figures are presented in Appendix A to visually demonstrate the concepts presented in this paper.

2. Mixer Networks

a. Core Theory

The concept of Mixes and Mixer Networks in telecommunication networks was first presented by David L Chaum [2]. With further research in Mixes, there have been various variations suggested to Mixes since then [5].

A Mix allows communication between two entities while concealing the transmitted message content and the identity of the communication partners without having to trust a third party. In a nutshell, each mix accomplishes this task using public-key encryption technology along with message permutation.

Public-key encryption works by having two functions that transform data, say f and g, called cryptographic functions. These functions have certain characteristics:

1. They produce a value based on two parameters: a number (called a 'key') and the data that needs to be encrypted or decrypted.

[1] Secure communications are communications where the message content is concealed from everyone except the receiver and the sender.

2. Function g with its key, say x, would be the inverse of function f with its key, say y, and
3. Decryption using ‘g’ without knowing the correct key used during encryption is impossible, or at least, impractical.

Each process, that wishes to participate, has at least one combination of these keys that makes the above conditions be true. The key ‘x’ of any process is said to be the public key and is known by everyone. The key ‘y’ is known as the private key and is known only by a single process, the process to which it was issued to.

Each Mix and participant process has its own private and public keys. Furthermore, all public keys are published and are accessible to anyone who chooses to access them. Thus, everyone can use the recipient’s public key to encrypt a message while only the recipient knows the correct private key and can use that to decrypt the message.

Public key encryption provides us with secure message content by allowing the sender to encrypt a message using destination’s public key. To avoid source destination linking, all the communication pipes used by the intermediary routers from the source to the destination could be encrypted, perhaps using some type of symmetric key encryption. This would work to provide resistance against source destination linking as long as we continuously have high network traffic and could trust the intermediary routers. Mixer networks are intended to allow us to communicate with similar degree of anonymity without having to trust the intermediary routers.

In a typical Mixer Network setup, multiple Mixes are cascaded together. This cascading provides stronger resistance against sustained and repeated attacks on the network by allowing route variation and by providing multiple nodes that transform message order and content. The public-key encryption technique provides resistance against eavesdropping of message content while message reordering provides resistance against source-destination linking.

We provide a visual depiction of Mix operations in Appendix A, figure 1.

b. Details of the Protocol

In this section, we describe the setup and behavior of a Mixer Network. We also discuss the transformation of a message from the time it leaves the source until the time it reaches the destination. Furthermore, using a single mix would be very much like trusting a third party. To avoid this, multiple mixes are cascaded together where each mix only knows its predecessor and successor mix. In this case, the first mix knows the sender and last mix knows the receiver; but since messages cannot be traced through the mixes, as each mix does not preserve FIFO ordering, the above information can be used for source or destination activity but is hardly useful for source-destination linking.

In a Mixer Network, certain machines are designated as ‘Mixes’ and their identity along with their public keys are available to all machines[2]. The identities and public keys of the rest of the participants are similarly publicly available.

To utilize a Mixer Network, the sender would use some of these Mixes as intermediaries in the message’s route to the destination. It would then encode the message using the destination process’ public key. This encrypted message along with the address of the destination process is encrypted again using the public key of the last Mix in the series of

[2] Obtaining public and private key combinations and distributing public keys could be done by using Certification Authorities and is not discussed here.
See http://developer.netscape.com/docs/manuals/security/pkin/ for further information.

Mixes in route to the destination. This data, along with the address of the last Mix is encrypted using the public key of 2nd to last Mix and so on. Discussed below are the encryption details and snapshot of the message along its route.

To hide the content of the messages traveling from source *A* to destination *B*, the source will repeatedly encrypt the message as described above. The final data that is to be sent would be in the following form:
→ Kn(Rn, K<n-1> (R<n-1> , … , K2 (R1 , K1 (R1, Kb (R0, M), #b))…))
In the above message encryption:

- #b is the address of the destination, *B*.
- Rx is some random data to avoid detection of encrypted message
- M is the actual message from *A* to *B*.
- Kx (Y) is the encryption of Y using the public key of x; ‘x’ being a number designating a Mix.
- ‘1’ through ‘n’ are labels for the Mixes chosen by the source process as intermediary Mixes between itself and the destination. The algorithm to pick these Mixes is up to the source process.

After the first Mix processes the message, it reaches the next mix in the following form:
→ K<n-1>(R<n-1>, K<n-2> (R<n-2> , … , K2 (R1 , K1 (R1, Kb (R0, M), #b))…)).
Eventually, the last mix will be able to access the necessary data, i.e. Kb (R0, M) and the destination address #b, to forward it to the appropriate destination. This technique would also prevent the message content from being revealed as the message M could only be decrypted at the final destination as it will be encrypted by the destination machines public key.

Furthermore, Mixer Networks allow the receiver to respond to the sender without divulging the identity of the sender or receiver to anyone. To accomplish this, a sender with address x, who wants to be able to receive a reply attaches K1 (Kx1, #x) and Kx2. In the above notation:

- K1 (Y) is the encryption of Y using public key of Mix 1. Here, we label the Mix that is closest to x as Mix 1.
- Kx1 is the public key of *x*, which also serves as random data for the encryption.
- Kx2 is another public key of *x* chosen for this occasion.
- #x is the address of *x*.

The destination, y, would be replying to a virtual address, K1 (Kx1, #x), with a message Kx2 (R0, M), again, following the above-mentioned encryption algorithm. When the final Mix in the response path (closet to *x*) receives this information, it decrypts part of the message (i.e. K1 (Kx1, #x)) and realizes the actual destination of the message is x. It then composes the message to x after encrypting it using Kx1 (i.e. it sends Kx1 (Kx2 (R0, M)) to #x). Since both the public keys, Kx1 and Kx2 were provided by *x*, it has the necessary private keys to access the message.

Further analysis and experimentation using Mixer Network could be done using a tool called MixDemonstrator. This tool, written in Java, allows one to model a Mix Network and perform various actions in the network. It provides detailed results that can be used to perform further analysis [3].

c. Configurations

Mixes could operate in two different modes: Pool Mode and Batch Mode. These two variations are discussed below.

Pool Mode: Mixes are setup by providing a threshold that designates the wait-time by specifying the number of messages to wait on. Once the designated number of messages has arrived, a Mix functioning in Pool Mode selects one of these messages and delivers it to the next Mix or the appropriate destination, as appropriate. This method does not guarantee any bounded waiting time for message delivery.

Batch Mode: Batch mode performs very similar to the Pool Mode. In this mode, a Mix still waits for the designated number of messages to arrive. Once the designated number of messages has arrived, a Mix in batch mode reshuffles all the messages it has and delivers all of them to their appropriate destinations. This mode guarantees steady progress of a message across the route until it reaches the destination.

d. Advantages and Limitations

There are numerous variations to Mixes and the advantages and limitations of Mixes vary based on those variations. In this section, we discuss some of the key advantages and limitations of Mixes.

An adversary can attack a particular Mix by sending (n-1) messages to it which would be possible if the adversary was aware of ‘n’ being the threshold of a Mix. Determining the threshold of a Mix is not a hard task as traffic analysis (analysis of incoming and outgoing messages) of a target Mix could easily reveal this number.

This attack is called a flood (sometime flush or blend) attack. The intention behind this attack is to isolate a target message i.e. have knowledge of all the messages in a Mix, except the target message. This way, when the Mix fires (delivers messages), the adversary could monitor the outgoing messages. Since it was the author of all the messages except one, it just needs to identify the ‘odd’ message of all the outgoing messages, and it could trace its target message. Furthermore, a comprised Mix could mount similar attack by waiting for the ‘right time’ before sending any messages to the next Mix. This sort of attack can allow an adversary to trace a message from the source process to the destination, thereby overcoming the ‘untracebility’ provided by Mixes without having all (or any) of the Mixes compromised. Similar attacks could easily be mounted against any Mix that has time threshold.

Pool Mode provides better resistance against this type of attack as in such mode as there is no guaranteed upper bound for message delay at each Mix. While the attacks in Batch mode were *certain* and *exact*, the attacks in Pool mode are *uncertain* and are accomplished at a much greater cost than in Batch mode [5]. Pooling mode does make it very resource intensive to mount attacks but it still leaves open the possibility to detect if any attempted attack was successful and allows for retry of the same attack. Further improvement on this can be made by making the detection of the results of an attack harder [5].

While the Pool Mode of a Mix has this advantage of resistibility against flood attacks, it has a draw back — no upper bound on the number of round of message delay. In pool mode, upon reaching the threshold, a single message is randomly picked and sent to the next Mix or destination. Since this selection process is random, there could potentially be infinite time delay before a message gets delivered.

Another type of attack that could be mounted against a Mix Network is that of ‘message replay’. A Mix overcomes this attack by discarding all duplicate messages that try to pass through the Mix. Checking for message duplicity could be resource intensive as all the messages that pass through a particular Mix must be logged. To avoid this intensive resource consumption, periodic key changes could be announced by Mixes that allow them to discard earlier message logs.

One impressive advantage of Mix Network is that it allows anonymous communications without having to trust a single third-party. Thus, in most cases, messages can be fully and easily traced or analyzed only if the entire set of intermediary Mixes are compromised. In a general encrypted connection, message is encrypted and sent to the destination. In such a case, compromised machines in the route from the source to the destination could cooperate to link the source to destination. Furthermore, sniffing network packets and comparing data allows an adversary to trace a message. In the case of Mixes, since the data changes along its route at each Mix, cooperating-compromised Mixes could only link the source to the destination only if all the intermediaries are compromised.

An interesting characteristic of Mix Network is the determination of the route (intermediary Mixes) by the sender. This attribute of Mix networks provides the source with the flexibility to choose the route, but alongside that exists the overhead of encryption. Repeated encryption by public keys for the entire set of intermediary Mixes is a fairly exhaustive operation and thus adds to the delay in message transmission and to the communication latency.

Another advantage of Mix networks is its low bandwidth requirement. As all the Mixes are triggered only by message or time thresholds, there is, for the most part, no activity going on between the Mixes, unless it is needed. In certain cases, Mixes do generate fake message, which might cause some activity aside from real communication activity. In such cases, the overhead on traffic due to fake messages would be determined by the trigger mechanism for generating the fake messages. This seeming advantage of low traffic in most cases also serves as a disadvantage. Due to this limited traffic, Mixes do not guarantee real-time communications. Batch mode provides for faster communication then pool mode as it is triggered by the arrival of certain predefined number of messages. Upon being triggered, batch mode would deliver all the messages, and thus would bind the message delay to the delay in arrival of predetermined number of messages. The situation gets worse in pool mode due to random message selection process upon receipt of a threshold number of messages.

Another type of attack that is prevented by Mixes is the re-encryption attack. A Mix concatenates random data to the actual message content to prevent someone mounting a re-encryption attack. A re-encryption attack works by applying brute-force. It could be mounted by any adversary by sniffing network packets. Common data sets (messages) could be encrypted and compared against the encrypted text. For e.g., to test if a message, say X, encrypted using machine A’s public key says, ‘Meet me at John Doe’s Place at 11:30 PM’, the text, ‘Meet me at John Doe’s Place at 11:30 PM’, is encrypted using A’s public key and compared against ‘X’. Concatenating random data makes it harder for anyone to mount re-encryption attack by (1) increasing the message size (more things to try) and by (2) convoluting common messages with random text.

3. Onion Routing

a. Core Theory

Onion Routing seeks to provide a system that is resistant to traffic analysis and to source-destination linking. It incorporates public key cryptography for hiding message content. It provides anonymous connections without having to trust a third party by implying the technique of Mixer Networks. It provides various onion routers that behave very much like the mixes as they seek to de-link the messages they receive and the ones that they send out. One important difference is that onion routers are real-time: i.e. instead of waiting for a certain number of messages, they wait on certain length of time before they mix and output messages. In this case, if an onion router does not receive enough messages, there is some chance of linking its input and output. Since we usually expect a minimum amount of traffic for onion routers to operate, the probability of this leading to source-destination linking is very low.

b. Details of the Protocol

In onion routing, the communication proceeds as follows: the source, A, instead of directly connecting to the destination, B, connects to a local *Application Proxy*, which in turn connects to an *Onion Routing Proxy*, and provides it with the destination address in a pre-defined format. The job of an application proxy is to filter source-identifying information from the data stream, provide fault tolerance (repost or flag any error's reported by the last onion router), and format the information provided by the source application to be readable by the onion proxy [4]. An Onion Routing Proxy's function is to allow the sender to communicate with the Onion Routers. Upon receiving a connection request, the Onion Routing Proxy sets up a connection from the sender to the receiver through a series of Onion Routers. After a connection has been setup, each time the Onion Routing Proxy receives a message from A intended for B, it routes it through the various onion routers which are in this connection route. To allow content anonymity, the Onion Routing Proxy repeatedly encrypts the message from A, which usually will be encrypted by the destination's public key, using the public keys of all the *Onion Routers* in the connection route as the source process did in Mixer Networks. These layers of encryption form an *Onion*. This Onion is passed through various Onion Routers in the connection route, each decrypting one layer of encryption, finding the next destination, padding the Onion to compensate for their layer of the Onion, and forwarding it to the next destination. Eventually, the Onion reaches the final Onion Router, which forwards it to a *responder's proxy*. The function of a responder's proxy is to pass the data from the onion network to the receiver. A source might send an encrypted copy of its real address to the destination to allow for response from the receiver. This might be done in a similar fashion as it is done in the Mixer Networks.

As with Mixes, each Onion Router has knowledge of only its adjacent Onion Routers. Furthermore, as with Mixes, it is hard to link the data being received and data being sent by a particular Onion Router as the data is shuffled and decrypted at each Onion Router. The above two factors provide resistance against source-destination linking. Onion Routing also provides a fixed message size and fixed number of messages in each network channel, which provides resistance against network analysis.

c. Configurations

There are three major configurations of Onion Routing.

Network Identity Protection Configuration:

In this model, the goal is to protect the identity of the domain of the message originator. In this case, the Onion Routing Proxy may be installed at a gateway/firewall and may seek to provide anonymity to the computers behind the gateway/firewall. As with all Onion routing proxies, this Onion Routing Proxy should also service requests by other computers.

This organization has the following qualities:

- The onion routing proxy is the most trusted one, as it knows both the source and the destination.
- If the receiver is outside the onion network, then the data must not contain information identifying the actual sender but might identify an onion router as the sender (such as the TCP header identifying the sender).

Individual Identity Protection Configuration (Customer-ISP Model):
In this configuration, the identity of an individual is to be concealed. This scenario is similar to a user connecting to his/her ISP, which runs onion routers, or at least links to other onion routers. In this model, the onion routing proxy is present on the customer's machines. This onion routing proxy, as with other onion routing proxies, can service other connection requests. Since the most trusted component of the onion network, the onion routing proxy, is present on the customer's computer, this provides more security. The customer's onion routing proxy generates various onions, which traverse through the onion routers present or linked with the user's ISP, and possible others, before they reach the destination. In this scenario, the other onion routers can know just the sender, or just the receiver. This provides stronger protection against source-destination linking as the source proxy resides on a trusted machine.

Remote Proxy Configuration:
In this configuration, the identity of an initiator is to be concealed but the initiator does not control any onion proxy by any of the above two models. In such a case, the initiator would make a secure connection to a remote onion routing proxy. As long as the initiator trusts this remote onion routing proxy, a sender would be able to communicate anonymously. In this case, sender activity could be established by any eavesdropper but establishing a source-destination link by any observer would still be a hard task. As long as the remote onion routing proxy is not compromised, it could establish a route through the onion network to the responder proxy and allow the initiator to communicate with the recipient. Thus, this configuration makes it possible for the source to anonymously communicate with the destinations as long as it could trust the remote onion routing proxy.

d. Advantages and Limitations

Since the concept of Onion routing is based on Mixer Networks, the advantages and limitations of Mixer Networks are inherited by Onion Routing. Certain differences in Onion Routing from Mixer Network do alleviate certain problems. In this section, we initially discuss the key differences between Mixer Networks and Onion routing and later present the advantages and disadvantages.

Onion Routing and Mixer Networks share the following key characteristics:

- Observed behavior similar to loose-sourced routing.: Both Mixers and Onion routing provide a set of router. The order in which data flows from through these routers is not pre-defined. Instead, the network either has a static topology or it has a dynamic topology which is published to be available to entities which might use anonymous communications. This way, the source or a source proxy would be completely responsible for determining the order and number of nodes that are visited as a message traverses this network. As we will see later on, the message progress steps in anonymity ring are predetermined and thus, AR does not behave as above.
- Decentralized trust points: In both of these networks, configurations exist that allow communication without trusting a single party. The trust is distributed between all the onion routers from the source to the destination, thus making the trust distributed. Thus,

in both systems, anonymous communications are compromised only if all the intermediary machines (mixes or onion routers) are compromised.

- Uncorrupted, causally delivered data: The protocols defined both by Mixers and Onion routers depend on the data that is being communicated to be uncorrupted and ordered. Thus, if a computer 'a' sends message 'm1' before, 'm2' to b, destination b must receive 'm1' before 'm2'. This is a realistic assumption as the Network protocols such as TCP do guarantee causal delivery and certain other algorithms exists that could be utilized to recovery corrupted data.
- Resistance against eavesdropping and traffic analysis: Aforementioned protocols aim to provide resistance against source-destination linking and content of the message being transmitted. Fundamentally, resistance against source-destination linking can be achieved by Mixes [2]. Resistance against eavesdropping is achieved by public key cryptography. It is important to note, however, that these protocols provide resistance to traffic analysis utilizing trivial techniques such as linking of incoming/outgoing messages through each machine by monitoring message order, message size or message data (bits).
- 'talk-back' to the sender: In both systems, message data could be used by destination to identify the sender, if the message data contains sender identifying information. This could be utilized by the receiver to respond back to the sender. This technique could be utilized if the sender intends to divulge its true identity to the recipient. Other mechanisms exist in Mixers (see sec 2.b), and onion routers (using connections) for the recipient to communicate with the sender without knowing the sender's true identity.

The following properties are limited to onion routing:

- Connections: Connections are established from the source to the destination before any communication can take place in onion routing. An entity called the onion routing proxy is responsible for setting up this connection. This entity is aware or both the source and is, therefore, the crux for allowing anonymous communications. Mixers function by proving the initiator with the ability to define a loose path a message would take to reach the destination. The ability of onion routing to work with an onion routing proxy makes for its easier integration with existing applications.
- Onion routing proxy and responder's proxy: Designated machines exists called onion routing proxy and responder's proxy that perform the task of communicating between the 'outside world' and the onion routing network. Onion routing proxy transfers message from the source into the onion routing network while the responder's proxy transfers the message from the onion routing network to the destination. This notion does not exist in Mixers.
- Communication characteristics: Onion routing provides a real-time and bidirectional setup for communications. Mixers provide a unidirectional communication pathway but do provide sender the ability to allow the receiver to 'talk-back'. Furthermore, communications in Mixes are not always real-time, even though this non real-time communication provide for additional security. Onion routing, on the other hand, is real-time, and due to this, unless fake messages are employed to generate fake traffic, messages could potentially be traced by observing network traffic.
- Application Independence: Onion routing provides complete application independence for any proxy aware application. Any proxy aware application would make connections to an onion routing proxy instead of the destination, and the rest of the details for anonymous communications are handled by the onion routing system. Furthermore, it could use NRaD redirector [8] to seamlessly integrate onion routing with TCP/IP.

- Connection establishment frequency: While the Mixes could establish and close connections between themselves as needed, the onion routers are connected using longstanding socket connections. A variable route through these static connections, along with other factors, is what contributes to anonymity.

One key advantage of onion routing is its seamless integration with various applications, without any modifications to the applications. This demonstrates the flexibility of onion routing and its practical applicability. Onion routing also provides real-time and bidirectional communications. Again, this makes it highly practical and great leap from the loose and non-temporal delay guarantees provided by Mixer networks. Furthermore, the various configurations of onion routers make it very convenient for various individuals and corporations to use it. The remote-proxy configuration makes it highly practical for anybody to utilize public onion routing networks without much effort.

A drawback of onion routing is that it requires most trust to be placed on the onion routing proxy. In certain configurations, this could cause the onion routing proxy (proxies) to become focal points for adversary attacks. Another drawback is that of its static topology setting and its inability to distribute or update public keys [4]. Yet another drawback is that of the overhead of connection setup.

4. Architecture of the Anonymity Ring

Our purpose of presentation of an anonymity ring is to define a protocol that incorporates the beneficial qualities of mixers and onion routing and presents a formal definition of such a protocol. In this section, we describe the basic communication infrastructure used in an anonymity ring, any assumptions we make of this infrastructure and provide a brief overview of each participant's expected behavior.

Anonymity Ring aims to provide anonymous communication without having to trust a single third party. It uses a structure similar to the pooling mode of a Mix for each of its ring-participant machines. Data is communicated between neighbors by using send and receive keys, which are frequently changed. Although duplicate messages could be discarded in an AR, as was done in Mixes, frequent changes to the symmetric keys between neighboring ring-participants decrease the severity of this problem in AR. AR also provides for message delays in the system itself. AR could be used by applications by using a structure similar to the application proxy in onion routing for seamless integration into current systems.

a. Organization of Anonymity Ring

The anonymity ring is composed of all the computers that would like to communicate anonymously. Moreover, among these, there will be a group of designated computer, called ring-participants, (at least two) which will have unidirectional connections amongst each other that form a ring (see Appendix A: figure 2 and figure 4). Each of these machines has exactly two connections with two distinct ring-participants: a send connection and another receive connection. Additionally, each of these machines can open send and receive connections to any other computer. In addition, each of the participants in the Anonymity Ring has their public key, previous ring-participant's address, next ring-participant's address, message clearing cycle number, handle, handle map clearing cycle number, current round number and a unique identifier for the anonymity ring they belong to publicly available. This data also allows non-ring participants to determine the size of an anonymity ring. The numbers published as *message clearing cycle* and *handle map clearing cycle* specify the numbers that, if they leave no remainder after division by the current anonymity ring round number, would clear the data stored for in-transit messages and handle map respectively. For e.g. if the message clearing cycle number is 3 and handle map clearing cycle number is 4 for a ring-participant, then in round number 9 of the anonymity ring, only the handle map would get cleared while in round 12, both the handle map and message data gets cleared. This published data and the implications of these values on an anonymity ring are discussed further in section 5b.

It is the participant's responsibility to make available its handle to any process that it wishes to receive messages from. A handle is very much like a nickname given to each participant. Only the ring-participant machines know how to translate from a handle to a real destination.

Each message being sent to the destination is divided into two parts for the purpose of transmission: the content message and a trigger message. Throughout this paper, we call the original message that needs to be sent, before this split, the *actual message* and the split parts during this transmission, together, as *component messages*. These two component messages proceed independently in the anonymity ring but are set to terminate on the same ring-participant machine. The content message contains that actual data that is to be received by the destination along with some routing information. It is set to terminate before the trigger message and idle in the cache of the last ring-participant. Before this cache is cleared, the trigger message is set to arrive to the same ring-participant to cause the idling content message to be sent to the (next) destination. Note that the trigger message does not contain the actual message content intended for the destination, but instead just enough information to authentically trigger the idling content message.

A *mapping message* is a message that is destined to a ring-participant and provides a mapping from a destination handle to an actual destination address, if it is not a fake message. This type of message is also transmitted by being split into a content and trigger message as above. In this case, the content message contains the mapping and the trigger message causes this mapping to be stored at the destination ring-participant. The mappings that the mapping messages generate are required for any actual message to reach its destination; otherwise, no translation from a destination handle to an actual destination would exist and the actual message would be discarded.

In an anonymity ring, it is recommended for the non-participant machines to regularly send 'fake' messages addressed to self or to any mock destination handle with invalid message content. This could also be accomplished by sending just one of the two component messages (content message or trigger message) needed for any successful communication. Such solitary component messages without their counterparts are ignored by the ring-participants and very well serve the purpose of 'fake' messages. Note that these messages are not labeled publicly to be 'fake' to avoid the detection of them being 'fake' from the ring-participants. Otherwise, if the ring-participants are aware that these messages are fake, these messages would not serve their purpose of confusing the compromised ring-participants.

We assume that all messages are delivered to the destination specified in the message header in an uncorrupted state. Furthermore, we assume that the messages are causally delivered. Another trivial assumption we operate under is that the private keys in the system are uncompromised. Additionally, we assume the following condition holds for each of the send (sk) key at a ring-participant and the receive (rk) key at its neighbor:

DEC (rk, ENC (sk, d)) = d = DEC (sk, ENC (rk, d)).

Using terminology from onion routing, the structure of anonymity ring could be described as follows: (1) all ring-participant machines can be considered responder proxies and (2) the onion routing proxies would always exist on the message initiator machines.

b. Initialization

In an anonymity ring, the non-ring participant computers are the machines that usually need to communicate anonymously. Ring-participants exist, much like Mixers, mostly to facilitate those communications.

In order for a computer with handle *A* to communicate to a computer with handle *B*, the following preparatory steps are involved:

1. Pick two starting ring-participants, say R0 and R1. R1 would be utilized as the starting ring-participant for the 'content' component of the communication and R0 will be utilized as the starting ring-participant for the 'trigger' part of the message.
2. Pick a number, say NUM_HOPS, between 0 and MAX_HOPS, where MAX_HOPS is the maximum number of allowed hops in an anonymity ring. MAX_HOPS should be greater than the number of ring-participants in an anonymity ring. This number will determine the number of ring-participants that will be utilized as intermediaries, inclusive of the last ring-participant, before the 'content' component of a communication reaches its final ring-participant machine. This concept could be further extended for inter-anonymity ring communications (see sec. 9b).
   It is best if this value is greater than 'n', the number of ring-participants in the AR, as this will ensure that the message traverses all the ring-participants and hence, all the ring-participants must be compromised for any information about this communication to be revealed.
3. Pick a multiplier, 'M'. This is utilized to identify the number of rounds a trigger message would perform of an anonymity ring before reaching the intended destination. For this purpose, the size of the ring is determined by the data published by each anonymity ring-participant (see sec 5a). A standard exponential distribution [7] could be utilized in picking M. This number should satisfy the following conditions: M >= 0 and ((M * RING_SIZE) + dist(R0, R1)) < MAX_HOPS. RING_SIZE is the number of ring-participants present in the anonymity ring while dist(x, y) is defined to be the number of hops it takes from x to reach y while traversing an anonymity ring only along its neighbors. The return value for dist(x, y) would range between 0 and RING_SIZE - 1.
4. Pick two numbers between 0 and (MAX_HOPS-NUM_HOPS-1) and 0 and (MAX_HOPS-((M * RING_SIZE) + dist(R0, R1))-1) respectively, say START0 and START1, using a statistical distribution similar to the standard gamma distribution [7]; this will designate the starting hop array indices for each of the two message components. This value is the value read in as the variable 'c' in our process specification presented in section 6a.

Step #1 is designed to provide a random starting point for the component messages. This random initial hop along with the number of hops determine the exit point and also dictate the encoding of a message as it traverses an anonymity ring. Step #2, as described above, serves to provide the number of hops a content message would take on its route to the destination. Anonymity ring gives us the possibility to have the content message delayed by a certain amount of time, until the trigger messages reaches the last router. For this delaying to work, it has to be ensured that the message content or the mapping from handle to actual destination do not get cleared before the trigger message causes them to be sent. This requires timing analysis by the source process to dispatch the messages at the appropriate times.

Step #3 determines the amount of the delay by identifying the number of additional rounds a trigger message would take around an anonymity ring before reaching the intended destination. It is important to note that while this time is being determined, the amount of delay generated by the extra cycle of trigger message around the anonymity ring does not exceed the message clearing delay. If we assume that the content message was delivered to the last router on round ‘x’, this time constraint could be assured by assuring that the trigger message reaches the last router within ((*message clearing cycle*) - (x mod *message clearing cycle*)) cycles of the content message. Alongside this, if ‘y’ was the anonymity ring cycle number at which the mapping message [3]reached the last ring-participant in route to the destination, to assure that the translation from the destination handle to the actual address exists in the last ring-participant, it must be assured that the trigger message (of the actual message) reaches the last router within ((*handle map clearing cycle*) - (y mod *handle map clearing cycle*)) cycles of the mapping message reaching it. Appendix A, figure 3 provides the structure of these messages.

In step#4, the random index-start value keeps the ring-participant at the first hop from trivially realizing that it has received a message from a message source computer. Note however that it is still possible for the first ring-participant computer to identify the sender by analyzing the header of the message it received and comparing it to its preceding ring-participant machine to discover it has received a message not from its preceding neighbor, and thus, an outsider. However, this will, at best, establish only sender activity. If the initiator machines generate fake messages (see sec. 5a), it will make it harder for the ring-participant at the first hop to even authentically establish the presence of valid sender activity. Inter-anonymity ring routing (see sec. 9b) could also further elude the detection of any sender activity.

c. Message composition in an Anonymity Ring

After completing these preliminary tasks of generating various values, the source process needs to generate two messages: a mapping message and an actual message. Both of these messages are split and each consists of a content message and a trigger message.

The mapping message would be sent to the last ring-participant in the route from the source to the destination. The source process might send some fake messages (see sec. 4a) to avoid this message activity from being detected.

As mentioned above, both of the above mentioned messages are divided into content message and the trigger messages. We discuss only the construction of content and trigger message for the actual message. A parallel of this algorithm could be used for the construction of content and trigger messages for the mapping message, except the data that would be sent would be a destination handle and an actual destination address and the messages would be destined for ring-participants. Following is the discussion how the content and trigger messages for the actual message are constructed.

- The message type would be “msg” and “fire” respectively. (see sec. 5 and 6).

[3] Mapping message is a message used to add a map entry from a destination handler to the actual destination address that is destined to a ring-participant machine. This dynamic property of these maps allow for further resistance against being traced. These types of messages are destined to ring-participants and are still transmitted after being split into content and trigger messages. In the protocol specification, the mapping is stored in the dstlkup array.

- The index of the content and trigger messages would have the values generated by the source process during initialization, namely START0 and START1 respectively.
- The values of the hop bits between the index of 0 to START0 - 1 and 0 to START1 - 1 for the content message and trigger message, respectively, are irrelevant.
- For the actual message's content message, the values from the START0 index to START0+NUM_HOPS-1 index of hops array are '0' in the hops array to indicate to the ring-participants to keep the content message in the ring and forward the message to the next ring-participant. The START0+NUM_HOPS index for hops array will have a value of '1', to indicate to the ring-participant to decode the fields and send the information to the actual destination.
- As above, for the trigger message, the values from the START1 index to START1+((M * RING_SIZE) + dist(R0, R1))-1 index of hops array are '0' in the hops array to indicate to the ring-participants to keep the trigger message in the ring and forward the message to the next ring-participant. The START1+((M * RING_SIZE) + dist(R0, R1)) index for hops array will have a value of '1', to indicate to the last ring-participant to decode the fields and send the corresponding content message to the actual destination.
- Content message would contain the data, i.e. the message content, in an encrypted form. In its place, the trigger message would contain the destination handle, perhaps, padded with random data.
- Message digest [1], in both messages, would be the message digest of the unencrypted data.
- Message index is a unique index given to this message by the source. The message index value in content message and the corresponding trigger message would be the same.

After constructing the messages, the source process goes through the process of encoding the message. It follows the following algorithm in encoding the content message, assuming 'd' is the data that is being sent, dh is the destination handle of the intended recipient, dst is the destination handle of the last ring-participant and that 'hop' is the array of hop bits (see sec. 6 for definition of other variables):

```
d = ENC (pklkup[dh], d)
For (int i = START0 + NUM_HOPS; i >= START0; --i)
   (t, hop[i], msgndx, md, d) :=
         ENC(pklkup[p[$dst+_{n}START0+_{n}NUM\_HOPS-_{n}i$]], (t, hop[i], msgndx, md, d))
e:= ENC(sk,(t, c, hop[0..m-1], msgndx, md, d))
```

After the preparation of the content message, the trigger message is prepared as follows:

```
For (int i = START0 + NUM_HOPS; i >= START0; --i)
   (t, hop[i], msgndx, md, dh) :=
        ENC(pklkup[p[$dst+_{n}START0+_{n}NUM\_HOPS-_{n}i$]], (t, hop[i], msgndx, md, dh))
e:= ENC(sk,(t, c, hop[0..m-1], msgndx, md, dh))
```

Notice that each of these fields is fixed size. We expect the encryption to preserve content length and data is assigned to various fields based on this assumption.

In addition to this algorithm, an initiator process might perform some addition tasks before executing this algorithm. For example, it could choose to embed information to allow a destination process to ‘talk-back’ by providing it with destination handle to write back to, unique id of the anonymity ring to use, the ring-participant to start at, number of hops to traverse and the start/stop time during which the communication can happen. All this information is needed by the destination since the source process would be the one that would be adding a mapping on the last ring-participant (in the reverse route).

5. Message processing in the Anonymity Ring

a. Types of Message

There are two anonymity-ring identifiable message types in the anonymity ring:

1. The ‘msg’ message: This type of message is used to send data to the destination. It is also used to fake sender activity. This message, by itself, will not be able to route itself to the destination. It only contains the data that is to be sent to the destination, encrypted repeatedly using the public keys of the intermediary processed (see sec. 4c). It stalls on the last ring-participant identified to it and idles there until a ‘fire’ message reaches the same ring-participant and triggers the ‘msg’ message’s send to the appropriate destination. It is the same as the content message identified in sec 4a.

2. The ‘fire’ message: The ‘fire’ message is used to accompany a ‘msg’ message and is used to trigger an idling ‘msg’ message by identifying the destination handle to the appropriate router. This could also be used to generate fake traffic. This is the same as the trigger message discussed in sec 4a.

Theoretically, there are also two types of message that are needed in the system. They are: mapping message and an actual message. An actual message is the original data that needs to be communicated. A mapping message exists to add a ‘handle to actual destination’ mapping in the last ring-participant to facilitate the actual message (see sec. 4a). Beside these two, we also have a ‘fake’ message used to generate traffic(see sec. 4a). A visual depiction of these is presented in figure 3 in Appendix A.

b. Details of message processing

As mentioned earlier, each neighboring process in the AR has a send key (sk) and a receive key (rk). The sk and rk satisfy the following properties:

- DEC (rk, ENC (sk, msg)) = msg;

Each time the message is forwarded from one router to another in the AR, it is encrypted using sk of source router, and later decrypted by the destination using its rk. This protects the data, which specifies the type of the message and its *c* value (the first and the second fields of any message). Furthermore, each ring-participant only knows the send/receive keys for its neighbors. Thus, these keys are not as widely known as the public keys of each ring-participant and hence, not any eavesdropper would be able to identify this information. Only ring-participants can identify this information, and only compromised ring-participants try to misuse this information. Since fake activity is generated by using any of the available message types, it will be hard for any compromised router to conclusively conclude anything using this information, even if it becomes available.

Messages in an anonymity ring could be broadly classified into four different categories: fake message, content message, trigger message and a system message. A fake message is used to generate fake activity in an anonymity ring, thus confusing any eavesdroppers. A content message serves to transfer data content from an initiator along the anonymity ring (see sec. 4a) to the destination machine. A trigger message exists to cause the content message to be fired to the (next) destination from its last hop (see sec. 4a and 9b). An actual message is comprised of content and a trigger message and is the original message that needs to be communicated anonymously. A mapping message is destined to a ring-participant, and is compromised of content and trigger message, and it serves to add a mapping entry from a destination handle to a destination address. A system message is intended to be a correspondence from any ring-participant to any other ring-participant. It allows ring-participants to inform each other of updated symmetric keys and other administrative things internal to the anonymity ring.

Although four different kinds of messages exist in an anonymity ring, only selected ring-participants are able to identity the type of any given message. In our process specification (see sec. 6), only the last ring-participant is able to identify the type of the message and process it accordingly. The awareness of the intermediary routes of message type information not only unessential, it could have a negative impact. For example, if all intermediary routers are aware that a message traversing through them is a 'fake' message, it would essentially provide a simple way for any compromised router to discard fake activity.

With the above information in mind, the behavior of the anonymity ring protocol at each intermediary ring-participant proceeds as follows:

As we had discussed earlier, each neighboring process utilizes a symmetric key pair to allow for secure yet quick communications between each other. Thus, as a message is received, the recipient ring-participant decrypts it using it's receive key. The decryption is performed as follows (see sec. 6 for variable declarations):

(t, c, hop[0..m-1], dt) :=**DEC** (rk, e);

After this decryption, the recipient will have access to the index (hop counter) value. This value allows the anonymity ring to identify the appropriate 'hop' bit which will determine the next destination of the message. However, this value is encrypted, along with some other values, using the recipient's public key to protect anyone else from accessing it. Thus, the recipient now proceeds to decrypt this value as follows:

(t, hop[c], msgndx, md, dt) := **DEC**(pk, (t, hop[c],dt));

After this decryption, the destination process now has access to the correct hop bit (hop[c] above) which will determine the processing of this message. It is important to note that, at this point if hop[c] is false, nothing can be guaranteed about the value of 't'.

If this hop bit is false, it implies that the message is still in transit through the anonymity ring and should be forwarded to the next ring-participant. Thus, in this case, the index bit is incremented by one so that the next ring-participant would be able access the correct hop bit. The responsibility of incrementing this value could be taken away from the ring-participants for additional security (see sec. 9b). Additionally, the hop bit that was utilized currently is overwritten by some random data so that the next process cannot depend on that information. This new message is encrypted using the send key [part of symmetric key with the destination neighbor] and passed on to a buffering system along with the destination it needs to be sent to. The buffering system will, depending on its configuration, order this message to be sent and will also return a message along with a destination that needs to be sent now. Next, the current process waits until it is feasible for the message to be sent (i.e. the traffic in the 'outpipe' is less than the amount allowed) and then sends the message to the destination. The above steps could be represented in the AP notation as follows:

```
c, hop[c], d := $c +_m 1$, random, dstlkup[$i +_n 1$];
e := ENC(sk, (t, c, hop[0..m-1], msgndx, md, dt) )
BUFFMSG(in e, d out e, d);
sleep(#ch.p[i].p[$i +_n 1$] >= k);
send msg(e) to p[d];
```

If this 'hop bit' it true, it implies that the message has reached its terminating point in this anonymity ring. Thus, messages are now treated based on the message type. The details of this message processing are as follows (categorized by their occurrence in AP notation's specification):

Content Message and Fake Message: If the received message was a content message, then the recipient ring-participant extracts the data from the message it has received and the storage index at which this data needs to be placed at. The storage index is one provided by the source process and is formed by appending a message index[4] to the message digest, both provided by the source. If the index is invalid[5], then it is assumed that this was a fake or tampered message and is discarded. See sec. 9b to see how this protocol could be advanced to detect tampered messages using receipts. In AP notation, the behavior would appear as follows:

```
if ((msgndx, md) = 0) → continue;
□ ((msgndx, md) $\neq$ 0) → msglkup[(msgndx, md)] := dt;
fi
```

Trigger Message, Mapping Message and Fake Message: If the ring-participant received a trigger message, it would analyze the message content to identify the destination handle. Using this destination handle, it indexes the storage array (mentioned above as it is used to store data from content message) to see if any data exists. If no data exists, then it assumes the message it received was a fake message and discards the data. If data exists, then the recipient ring-participant analyzes the destination handle and compares to its own destination handle to see if it itself is the intended destination. If so, this would be a mapping message and a mapping is added in the ring-participant's map array from the destination handle to the actual destination address provided by the stored message. Otherwise, it implies that the current ring-participant is the hop before the message is delivered to the destination. In such a case, the received message is prepared to be of the same size as the received message and encrypted using the send key. It is then added to the send buffer. As mentioned above, a message is returned by the BUFFMSG function and this message is sent to its destination as soon as the traffic on the out channel falls below the maximum traffic limit. The AP notation for the processing of these messages would be as follows:

```
(dh, pad) := dt
dt : = msglkup[(msgndx, md)];
if ((dh = i) Λ(dt $\neq$ 0)) →{this is a mapping message}
        (dh, d) = DEC(pk, dt); {decrypt mapping }
        if ((dh $\neq$ 0 Λ(d$\neq$ 0)) →{if this isn't a fake message, store the mapping}
```

[4] usually a unique number; provided by the source process

[5] In our specifications, only a zero index in invalid

```
                dstlkup[dh] := d
            □ ((dh = 0 V(d = 0)) → continue
            fi
        □ (($dh \neq i$) V (dt = 0)) →
            msglkup[(msgndx, md)] := 0;
            if (dt = 0) → continue;
            □ ($dt \neq 0$) →
                {dh and dt have readable values}
                hop[c], c = random, $c +_m 1$
                dt = ENC(sk, (t, c, hop[0..m-1], msgndx, md, dt));
                BUFFMSG(in dt, lkup[dh] out e, d);
                sleep(#ch.p[i].p[$i+_n 1$] >= k);
                send msg(e) to p[d];
            fi
        fi
```

The complete specifications of the protocol are presented in the next section. The behavior of the send buffer and reordering can be tailored as desired; a few options for it are provided in section 7.

6. Process Specification

a. Main process

b. Functions

The following is the complete protocol specification of the Anonymity Ring using Abstract Protocol [1] notation.

```
process [i: 0..n-1]

inp m                 : integer  { max # of hops for each message }
    k                 : integer  { max # of msgs in each link }
    s                 : integer  { size of message buffer }
    rk, sk, pk        : integer  { rcv, snd and private keys for the process }
    pklkup            : array[integer] of integer
                    {buffer to translate from a destination handle to the pk of the specified
                     destination handle. pklkup[x] = 0 is a sentinel indicating the invalidity
                     of the destination handle 'x'. Maintained and updated as necessary
                     from outside this process. }

var  e                : integer    { Encrypted message }
     dstlkup          : array[integer] of integer
                    {buffer to translate from "handle" to address of original destination.
                      dstlkup[x] = 0 is a sentinel indicating that the router has no
                      translation for handle x. The values in this array are provided by any
                      computer. Frequently reset based on periods of time and lack of
                      activity. }

     msglkup            : array[integer] of integer
                    {buffer to translate from a msgndx, md provided by the source process
                      of a message to the actual message content. msglkup[x] = 0 is a
                      sentinel for no message existing for the indexing message digest.
                      Index zero is invalid and no source process should use index zero for
                      any of the messages it sends. Index zero of this array has a constant
                      value of '0'.  Furthermore, data in this data structure is periodically
                      cleared to avoid resource exhaustion. A regular clearing interval is set
                      and a specific entry with no activity for a specified # of rounds results
                     in it being cleared. }
     msgndx             : integer
                    {An index provided by source process. }
     t                 : {msg, fire}
                    {Identifies the type of the received message. Fake messages are
                      messages sent to self or with invalid information. They are used to
                      confuse all the processes esp. the adversaries. See figure 3}
     c                 : 0..m-1
                    {Hop counter bit. Identifies the bit in the hops array that will indicate
                      whether to send msg to ultimate destination, or to keep it in the AR.
                      Also referred to as the index value }
     dh, md, dt        : integer                  {dest. handle, msg digest, data}
     d                 : integer                  {msg destination}
     hop               : array[0..m-1] of boolean {the hop bits}
     ctr               : 0..m-1                   {Used as a start address of fake
                                                    message.}
     pad               : integer                  {used to read & ignore padding}
par  g                 : integer                  {any machine}


begin
rcv msg(e) from p[g] →
      (t, c, hop[0..m-1], dt) :=DEC (rk, e);
      (t, hop[c], msgndx, md, dt) := DEC(pk, (t, hop[c],dt));
      if (t = msg) Λ(hop[c] = true) → { store message }
            if ((msgndx, md) = 0) → continue;
            □ ((msgndx, md) ≠ 0) → msglkup[(msgndx, md)] := dt;
            fi
      □ (t = fire) Λ(hop[c] = true) →
            (dh, pad) := dt
            dt : = msglkup[(msgndx, md)];
             if ((dh = i) Λ(dt ≠ 0)) →{this is a mapping message}
                    (dh, d) = DEC(pk, dt); {decrypt mapping }
                    if ((dh ≠ 0 Λ(d≠ 0)) →{if this isn't a fake message, store the mapping}
                          dstlkup[dh] := d
                    □ ((dh = 0 V(d = 0)) → continue
```

Comment: At this point, if hop[c] is false, nothing can be guaranteed about the value of 't'

Comment: A fake msg. Ignore it. Timeout action will take care of sending more messages as necessary. 'Continue' implies the end of this rcv action. Similar to 'break' in looping constructs.

Comment: Validate and send message to destination as we are the last ring-participant for this message.

Comment: It's a fake message. Jump to the end of this rcv action.

```
                        fi
                    ▯ ((dh ≠ i) V (dt = 0)) ➝
                        msglkup[(msgndx, md)] := 0;
                        if (dt = 0) ➝ continue;
                        ▯ (dt ≠ 0) ➝
                            {dh and dt have readable values}
                            hop[c], c = random, c $+_m$ 1
                            dt = ENC(sk, (t, c, hop[0..m-1], msgndx, md, dt));
                            BUFFMSG(in dt, lkup[dh] out e, d);
                            sleep(#ch.p[i].p[i$+_n$1] >= k);
                            send msg(e) to p[d];
                        fi
                    fi
                ▯ (hop[c] = false) ➝
                    c, hop[c], d := c $+_m$ 1, random, dstlkup[i$+_n$1];
                    e := ENC(sk, (t, c, hop[0..m-1], msgndx, md, dt) )
                    BUFFMSG(in e, d out e, d);
                    sleep(#ch.p[i].p[i$+_n$1] >= k);
                    send msg(e) to p[d];
                fi


timeout #ch.p[i].p[i$+_n$1] < k ➝
        t, c, hop[0..m-1], msgndx, md, dt, dh := random, random, 0, 0, 0, random,random;
        ctr, dh := random, dh mod n; { send fake message only to ring-participants }
        hop[ctr] := true;
        do (ctr ≠ c) ➝
                if (pklkup[p[dh$+_n$ctr$-_n$c]] = 0) ➝ halt; { something bad happened. Our
                                                          outgoing neighbor does not exist }
                ▯ (pklkup[p[dh$+_n$ctr$-_n$c]] ≠ 0) ➝
                        (hop[ctr], msgndx, md, dt) :=
                                ENC(pklkup[p[dh$+_n$ctr$-_n$c]], (hop[ctr], msgndx, md, dt))
                fi
                ctr = ctr $-_m$ 1;
        od
        (hop[ctr], msgndx, md, dt) := ENC(pklkup[dh], (hop[ctr], msgndx, md, dt))
        e:= ENC(sk,(t, c, hop[0..m-1], msgndx, md, dt));
        BUFFMSG(in e, dstlkup[dh] out e, d);
        send msg(e) to p[d];

end
```

Comment: Clear entry in the msglkup data structure.

Comment: Indirect data verification performed by using message digest as index. A value of '0' for dt indicates data tampering might have occurred or that this was a fake message that we ourselves generated. Invalid values will be regularly cleared and message tampering shouldn't cause the resources to eventually be exhausted. Furthermore, this value could also be utilized to allow for inter-AR routing.

Comment: Send message to destination

Comment: This process/thread blocks as long as the condition specified as parameter to the sleep function is true.

Comment: Any send here might even be sending the message to another anonymity ring for inter-AR routing. See sec. 9b.

Comment: The message is not at its last router. Hop it to the next router.

Comment: This process/thread blocks as long as the condition specified as parameter to the sleep function is true.

Comment: Could also be i$+_n$1.

Comment: Halting implies a complete halt in the operation of the current process.

7. Message Buffering

a. Description of Interface – BUFFMSG Procedure

The interface for adding a message in the buffer is as follows:

- BUFFMSG (**in** inmsg, indst **out** outmsg, outdst) where
    - Inmsg is the message to enqueue
    - Indst is the destination for the enqueued message.
    - Outmsg is the message to be sent.
    - Outdst is the destination of the message to be sent.

In an anonymity ring, the message buffer is always full. Thus, whenever a message is enqueued, another message is dequeued and sent.

Message buffering is a very important part of the anonymity ring. The configuration used for message buffering determines the amount of traffic resistance provided by a ring-participant. Furthermore, it determines the delay in transmission caused due to the reordering. These two aims are usually in conflict with one another. We want the message buffering system to provide an optimal balance between the two. Note that since our anonymity ring protocol specifications allow only the last ring-participant to identify the message type, no optimizations could be done in the BUFFMSG function by considering the message type information in ordering the messages. It is, however possible, that once a ring-participant detects that it is the last hop for a message, it would convey the information to the BUFFMSG function; this information could then be used by the BUFFMSG function to optimize to a certain extent (see sec. 9b).

BUFFMSG requires two input values: *param1*, *param2* and which are used as follows: BUFFMSG adds *param1*to the buffer, updating the priority array entry for this message as appropriate, depending on the configuration. It updates the destination array entry for this message with a value of *param2*. After these updates, depending on the configuration, it might organize the messages in each of these arrays in an order which “tries to” put the higher priority messages at the front of the arrays. At the end of its execution, it returns a message and the destination address the message needs to be sent to.

Below, we provide various versions of message buffering and discuss their advantages and disadvantages.

b. Configurations

i. Simple FIFO Configuration

1. Specification

The message buffering system could be configured to behave like a simple queue. The following protocol describes the behavior.

```
procedure BUFFMSG (in inmsg, indst out outmsg, outdst)
local
static var
        bff                   : array[0..s-1] of integer  {buffer of stored messages}
                                                          {init. with fake messages}
                                                          {priority array is unnecessary in this
```

```
                                                 case}
    dst             : array[0..s-1] of 0..r-1    {next hop of stored messages}
                                                 {init. i+n1}
var
    frnt            : 0..s-1;                    {indicates the index of the message that
                                                 has stayed in the queue the longest}
begin
    outmsg := bff[frnt]
    outdst := dst[frnt]

    bff[frnt] := inmsg;
    dst[frnt] := indst;

    frnt = frnt $+_s$ 1;
end
```

2. Advantages and Limitations

Even though the above protocol provides strictly bounded waiting (i.e. ‘s’ rounds of waiting) it is not practical as it provides a linkage between the messages sent and received. Thus, any observer could trace messages from the source to the destination using this configuration.

ii. Random-pick configuration

1. Specification

In this configuration, any message could be randomly picked to be sent out and the enqueued message is placed in its position. The AP notation for this protocol follows:

```
procedure BUFFMSG (in inmsg, indst out outmsg, outdst)
local
static var
    bff             : array[0..s-1] of integer   {buffer of stored messages}
                                                 {init. with fake messages}
                                                 {priority array is unnecessary in this
                                                 case}
    dst             : array[0..s-1] of 0..r-1    {next hop of stored messages}
                                                 {init. i+n1}
var
    pck             : 0..s-1;                    {indicates the index of the randomly
                                                 picked message}
begin
    pck := random;
    outmsg := bff[pck]
    outdst := dst[pck]

    bff[pck] := inmsg;
    dst[pck] := indst;
end
```

2. Advantages and Limitations

The configuration is similar to the proposed behavior of a Mix in pool mode. It would serve for the anonymity ring as it provides no clear linkage between the incoming and the outgoing messages. However, the protocol does not provide any kind of strict bound on the transmission delay but only provides a probabilistic bound. It is true that the expected value for the number of rounds required to pick message x (after it was enqueued) is ‘s’, but this does not guarantee us a bound. Thus, in this configuration, it is conceivable for a message to be “stuck” in the buffering system for a long time, thus causing a huge delay in communication.

iii. Priority Queue Configuration

1. Advantages and Limitations

We now look at the behavior of the process if we were to tweak the Simple FIFO configuration to behave like a priority queue, which enqueues messages in a queue based on the provided priority. In assigning the priorities to the incoming messages, there are two possibilities:

Possibility I: The priority for data messages could be some constant to start out with and decreases each time a new message arrives. The messages would still proceed in a FIFO order. Thus, this system is very similar to the FIFO configuration, and thus, impractical.

Possibility II: We could assign a random priority to incoming data messages. Messages are always dequeued from the front of the queue. At the end of each round, the priority of the messages existing in the queue decreases by one. This configuration would provide us with anonymity as it does not display any clear association between received and sent messages. Furthermore, the expected value for the number of rounds a message has to wait, in this configuration, is s/2.

The AP specification of this behavior would be very similar to the FIFO configuration and is not provided.

iv. Intensifying-probability configuration

1. Specification

The protocol specifications for the Intensifying-Probability configuration follow:

```
procedure BUFFMSG (in inmsg, indst out outmsg, outdst)
local
static var
        bff                 : array[0..s-1] of integer  {buffer of stored messages}
                                                        {init. with fake messages}
        pri                 : array[0..s-1] of [0..s-1] {priorities of stored messages}
                                                        {init. s-1}
        dst                 : array[0..s-1] of 0..r-1   {next hop of stored messages}
                                                        {init. $i+_n1$}
var
        buff2               : array[0..s-1] of integer  {buffer of messages
                                                         used to update and reorder buff,
                                                         priority and dest arrays}
```

```
pri2            : array[0..s-1] of [0..s-1]  {stores priorities for the above
                                              stored msgs.}
dst2            : array[0..s-1] of integer   {destinations of each of the above
                                              msgs.}
free            : array[0..s-1] of boolean          {init false}
                                             {stores boolean values that indicate
                                              if a particular spot in the above
                                              buffer is free}
done            : boolean                    {init false}
                                             {indicates whether the message we are
                                             dealing with was reordered}
tmp1, tmp2      : integer                    {loop control variables}
```

```
begin
    outmsg := buff[0];      { msg that is to be sent }
    outdst := dst[0];       { destination of the message to be sent }
    buff[0] := inmsg;       { store the given message }
    dst[0]=indst;
    pri[0] := s-1;
    {update priorities of existing messages}
    tmp1 := 0;
    do (tmp1 ≠  s) →
        if (pri[tmp1] ≠ 0) →
            pri[tmp1] := pri[tmp1] – 1;
        ▯ ((pri[tmp1] = 0) → skip
        fi
    od
    {done inserting new elements and updating priorities. Now, do reordering}
    tmp1, tmp2:=0, 0;
    do (tmp1 ≠  s) →
        done=false;
        do (done = false) →
            tmp2=random mod pri[tmp1];
            do (tmp2 ≠ s)
                if (free[tmp2]= true) →
                    free[tmp2]:=false;
                    buff2[tmp2]:= buff[tmp1];
                    pri2[tmp2]:=pri[tmp1];
                    dst2[tmp2]:=dst[tmp1];
                    done, tmp2:=true, s;   { exit the loops }
                ▯ (free[tmp2]:= false) → tmp2 := tmp2 +1
                fi
            od
        od
        tmp1 := tmp1+1;        {One more message was reordered}
    od
```

```
        {Now, go through and update buff, dest and pri}
        tmp1 := 0;
        do (tmp1 ≠ s) →
                pri[tmp1]=pri2[tmp1];
                dst[tmp1] = dst2[tmp1];
                buff[tmp1]= buff2[tmp1];
        od
end
```

2. Advantages and Limitations

This configuration works in the following manner:

- The message at the front of the queue will be dequeued and sent.
- The priority of all messages existing in the buffer is decreased by '1', unless it is '0' (the highest priority).
- The incoming message is placed at the front of the queue and is assigned a priority of s-1.
- For each message in the buffer, a random number is generated, which ranges from 0 to priority of the message being dealt with. This number specifies the place the message will be placed. If the specified index is full, the array is traversed starting at the number that was generated until a free spot is found. This will be future index of the message.

Thus, the higher the priority (low number implies higher priority) associated with a message, the closer it will be to the front of the queue. Thus, messages with higher priority are usually sent, and not delayed further.

The protocol does not provide any relationship between the incoming and outgoing message from a ring-participant. Again, this protocol does not provide us with a bounded waiting. The expected number of rounds a message has to wait in this protocol is 's'. Just as was the case with Random-pick and Priority Queue configuration, it is possible that a message gets delayed for an indefinite period of time in this buffer, but it is highly unlikely.

8. A Walkthrough

a. Message through AR

In this section, we provide a detailed walkthrough of how a message would travel from a source to a destination. We start out by listing the assumptions utilized in our walkthrough and then proceed with the detailed overview of communication through anonymity ring. To simplistically yet thoroughly demonstrate the behavior of an anonymity ring, we utilize an anonymity ring with only three ring-participants.

Assumptions:

1. The source process should be generating periodic fake messages which are either incomplete (have only one of the component messages. See sec. 4a) or destined to self. This provides greater resistance against adversaries seeking to establish source/destination activity or seeking to link source with the destination.

2. Initially, the current round number published by each ring-participant is zero and is updated (incremented by one) as the rounds progress.
3. The source process may use more than one anonymity ring (discussed in sec. 9b) but the demonstration will only deal with the message traversal through only one of the anonymity rings.
4. Ring-Participant 'A' exists in Anonymity Ring 'AR' (unique identifier of the anonymity ring) with public key A.bk, previous ring-participant 'C', next ring-participant 'B', message clearing cycle number 4, destination handle for self 'A' and handle map clearing cycle number 3;
5. Ring-Participant 'B' exists in Anonymity Ring 'AR' (unique identifier of the anonymity ring) with public key B.bk, previous ring-participant 'A', next ring-participant 'C', message clearing cycle number 4, destination handle for self 'B' and handle map clearing cycle number 3;
6. Ring-Participant 'C' exists in Anonymity Ring 'AR' (unique identifier of the anonymity ring) with public key C.bk, previous ring-participant 'B', next ring-participant 'A', message clearing cycle number 4, destination handle for self 'C' and handle map clearing cycle number 3;
7. For the mapping message, first hop ring-participants for content message is 'B' and for trigger message is 'A'. The content message is picked to have 4 hops. The value of 'M' (see sec. 4b) is 0. Thus, in this case, there are 4 hops for content message and 5 hops for the trigger message.
8. For the actual message, first hop ring-participants for content message is 'B' and for trigger message is 'C'. The content message is picked to have 4 hops. The value of 'M' (see sec. 4b) is 1. . Thus, in this case, there are 4 hops for content message and 6 hops for the trigger message.

Processing:

We are simulating a case where the source process 'X' needs to communicate with the destination process 'Y'. Initially, the source process analyzes the publicly published information about the different anonymity rings and decides to use an anonymity ring, say AR, with 3 ring-participants, A, B and C.

The next step for the source process 'X' is to compose a mapping message. A mapping message is composed of a content message and a trigger message. Figure 3 provides the view of content and trigger messages that form the mapping message as it would be seen by the destination ring-participant. The content and trigger message are prepared as described in section 4c. From now on, will use MMC and MMT to refer to this content and trigger messages, respectively. The parameters utilized in this preparation are provided in assumption #7, above. MMC in this case will have 4 layers of encryption using public key encryption, with the outer most layer formed by encrypting using public key of 'B'. MMT will have 5 layers of encryption using public key encryption, with the outermost layer formed by encrypting using public key of 'A'.

Similarly, an actual message (as defined in sec 4a) is prepared comprising of content and trigger message. From now on, will use AMC and AMT to refer to this content and trigger messages, respectively. The parameters for this preparation are provided by assumption #8, above. AMC in this case will have 4 layers of encryption using public key encryption, with the outermost layer formed by encrypting using public key of 'B'. AMT will have 6 layers of encryption using public key encryption, with the outermost layer formed by encrypting using public key of ‘C’.

After preparing this message, the sender now has to dispatch each message at exactly the right time, keeping in mind the storage and mapping clearing rounds in assumptions 4, 5 and 6 above. The following diagram shows all the messages that need to be sent and the relevant information.

Message	Destination	First Hop	Number of Hops	Route
MMC	B	B	4	B, C, A, B
MMT	B	A	5	A, B, C, A, B
AMC	Y	B	4	B, C, A, B
AMT	Y	C	6	C, A, B, C, A, B

The following timeline depicts when the messages are dispatched and received:

Round No.	x	x+1	x+2	x+3	x+4	X+5
MMC	At source	At source	Sent to B	Processed at C	Processed at A	Processed at B
MMT	At source	Sent to A	Processed at B	Processed at C	Processed at A	Processed at B
AMC	At source	At source	Sent to B	Processed at C	Processed at A	Processed at B
AMT	Sent to C	Processed at A	Processed at B	Processed at C	Processed at A	Processed at B

As all the messages reach the destination at the same time, the following events happen.

Firstly, the mapping message would add a mapping from a destination handle (used by the actual message) to the actual destination address. MMC and MMT are processed to accomplish this. Secondly, AMC reaches the last ring-participant and the data is extracted from this message and added to the storage map. Lastly, the AMT is processed and it utilizes the entry in the storage map and the destination handle mapping added by MMC and MMT to cause the content message to finally be associated with its appropriate destination. This destination and message pair is added to the send buffer to be sent at a later time, based on the processing BUFFMSG function.

In the above simulation, we assume that the message steadily moves through the anonymity ring, one hop in each round. As long as we constant traffic in the channels connecting the ring-participants, this steady progress assumption should not allow an adversary to establish source-destination linking. Furthermore, as we discussed earlier, this behavior of steady progress is determined by the BUFFMSG function and any change in the BUFFMSG function would imply changes in the timings discussed above. Additionally, introducing the concept of receipts (see sec. 9b) to anonymity rings would alleviate the overhead associated with timing the messages while sending them to the destination of while receiving a response. For example, the behavior of the last hop ring-participant could be modified to send a message back to the source (the source ‘handle’ would be provided to this ring-participant in an encrypted format) to acknowledge receipt of a mapping or a content message. This could, in turn, cause the source to take further actions as required (for e.g., the source could send the trigger message upon receiving a receipt for the corresponding content message).

b. Implementation Suggestions

In this section, we discuss key hurdle that might be encountered during the implementation of an anonymity ring.

The underlying implementation for communication in this protocol could be achieved through TCP/IP. The TCP/IP protocol would deal with various underlying details such as message reordering, message loss, etc, which are of little concern to us. The TCP/IP protocol could also be used in our communications in the following manner:

If two non-ring participants, **A** and **B**, want to communicate, **A** would create the message and send it to a ring-participant machine using TCP/IP. The ring-participant machines use TCP/IP to communicate with each other in the anonymity ring. Finally, when the message is dispatched to the final destination, the last ring-participant machine uses TCP/IP to send the message to the final destination.

Thus, for implementation, TCP/IP is essentially used in the following three instances:

1. To send message from a non-ring participant machine to a ring-participant machine for anonymous delivery.
2. Used by each ring-participant machine to communicate with its successive ring-participant.
3. Used by ring-participant machine to deliver a message to its final destination.

The communication pipes in all three cases should be encrypted to provide resistance against eavesdropping. This can be done, cost effectively, using symmetric keys. Also, the handle-destination mapping and the storage maps could be regularly cleared to avoid excessive resource consumption

9. Concluding Remarks

a. Advantages and Limitations of Anonymity Ring

In this section, we list the various advantages and limitations of anonymity rings and concisely elucidate them.

Following are the key characteristics of an anonymity ring:

Advantages:

1. Anonymity ring aims to provide anonymous communication without having to trust a single third party. It uses a structure similar to the pooling mode of a Mix for each of its ring-participant machines.
2. Data is communicated between neighbors by using send and receive keys, which are frequently changed. This provides tolerance to replay attacks. If message repeats occur before change to symmetric keys, AR could choose to discard duplicate messages, as was done in Mixes instead of changing symmetric keys.
3. AR provides for message delays in the anonymity ring itself. This could be particularly useful for resistance against eavesdropping. It also allows us to change or cancel a sent message within a period of time, should we decide to do so.
4. It could be used by applications by using a structure similar to the application proxy in onion routing for seamless integration into current systems.

5. AR uses a msglkup array that contains the data being transmitted without any associated destination. This separates message data from routing information. It also saves us valuable encoding time by cutting down on the amount of repeated public key encryptions on the actual message. This independence also allows for inter-anonymity ring routing (see sec 9b). Because of the separated nature of message content and its destination in an AR, a message could come from one AR while the destination information for that message could come from yet another AR. This adds to the resistance against eavesdropping and traffic analysis.
6. Fake traffic is frequently generated to hide any communications, both by ring-participants and by non-participants. It also lets source processes to easily insert fake messages which are cleared regularly. It provides constant traffic in the communication pipes of all ring-participants using fake messages to avoid the detection of any activity. This provides resistance against eavesdropping and traffic analysis. For e.g., as AR uses pool-mode type configuration, it will be fairly hard to mount a successful flooding attack with a lot of fake traffic.
7. It provides the initiator with the flexibility to add mappings from 'destination handles to actual destination' at ring-participants and allows it to clear these mappings at will. This allows a single computer to be addressed by multiple handles.
8. It allows for dynamic changes in the topology of an anonymity ring, as long as those changes are published. Furthermore, this also allows anonymity ring networks to dynamically list themselves for public utilization.
9. It provides the initiator to allow for a response from the destination in a bounded time frame (see sec. 9b) and dictate the terms and conditions of the response. Frequent clearing of the above mapping lists at each ring-participant also allows a source to dictate how and when it could receive a reply. For example, for a recipient to talk back to the sender, the sender must provide the recipient with the destination handle to write back to, unique id of the anonymity ring to use, the ring-participant to start at, number of hops to traverse and the start/stop time during which the communication can happen. If the sender always uses the same destination handle, it could just provide the recipient with just the destination handle and it must make sure that all the ring-participants contain the mapping at all times. Since this is reset frequently, sender could also use these mapping messages to generate fake activity. Furthermore, if it suspects that information might be compromised, it could overwrite the entry at its own will to point to something else or '0'.
10. AR could be extended to allow for receipts to isolate malfunctioning ring-participants (see sec. 9b).
11. Because of the structure of an anonymity ring, the graph structure which is usually used to hold the topology of the communication network in onion routing and mixers could be cut down to a simple list in AR.

Disadvantages:

1. Splitting the actual message in different parts inherently causes some timing details to be handled. For example, the trigger message of the actual message has to precede the content message of the actual message, which in turn must be preceded by the mapping message. Loosing this timing could cause the loss of a message, and it also adds to the overhead of communication along with overhead of resend(s). BUFFMSG function hence has a very important role to play as it determines the transmission delay boundaries at

each ring-participant. Without receipts, taking these delays into account is very important to ensure delivery of a message to the destination.

2. Limits have to be established on the number of hops allowed in an anonymity ring to have efficient communications. Additionally, to conserve resource usage, the storage mapping and the destination handle mapping have to be periodically cleared. This attempt at efficiency adds to the complexity at the source in timings the messages.
3. Maintaining a constant traffic through the channels in the anonymity ring conserves a great amount of bandwidth. Furthermore, constantly publishing information (see sec. 4) about each ring-participant also conserves a lot of bandwidth. The bandwidth from the source process is further conserved because of resends. Without enhancements (see sec. 9b), there would be no concept of receipts in anonymity rings. This implies that there would be no guaranteed message delivery after in enters the anonymity ring. Thus, without receipts, repeated sends of same message have to be executed to ensure delivery of messages. Consequently, the anonymity ring has the drawback of conserving a lot of bandwidth.

b. Areas of Further Research

Listed below are areas in which anonymity ring could be further improved:

- Frequent changing of send and receive keys: A new message type could be introduced to process specification, 'sys'. This will indicate system/administrative message intended to carry out activities internal to the anonymity ring. These messages could be used for frequently changing the symmetric keys between two neighboring ring-participants.
- Application proxy: An application proxy could be installed on the source machine which would handle the details of communications. The behavior of this application proxy would be similar to the application proxy in an onion routing network. The goal of such a proxy is to allow easier integration will existing proxy aware applications.
- BUFFMSG optimization: The last ring-participant is able to identify the type of message it has received. We could use this information to efficiently queue the messages in BUFFMSG by identifying fake messages. Similar queuing could be done in BUFFMSG for all fake messages generated at any ring-participant. This queuing could provide for faster transmission of valid messages and low priority transmission of fake messages for efficient and faster communications.
- Anonymity ring could be changed to provide receipt of delivery. This would allow the source to confirm that a message was delivered to the destination. Additionally, it would help in implicating any compromised ring-participants by allowing the source process to seek receipts from all the intermediary ring-participants. Currently, AR does not provide acknowledgements, but with the facility of receipts, message delivery acknowledgement could be provided. Acknowledgements could be further utilized in guaranteeing message delivery by making the communications fault-tolerant.
- Messages could be routed from one anonymity ring to another. This would provide for additional resistance against source destination linking. This could be accomplished by having the destination of a message be a ring-participant of a different anonymity ring than that used to route it. Thus, routing of messages could be layered through various anonymity rings, perhaps with a few hops at each anonymity ring. This would provide stronger resistance against traffic analysis but would come at the price of timing, bandwidth, encryption and calculation overhead to the source.
- The index value (designated by 'c' in sec. 6) could also be encrypted repeatedly at the source thereby relieving the ring-participants of the task of incrementing it at each hop. This

will provide resistance against compromised routers tampering with the index value or not incrementing it properly to cause the message to be discarded in its route.

- Further logic could be added to detect low network traffic thus implicating the ring-participant that should be the source of the traffic. This implication could cause the anonymity ring itself to shutdown by causing the neighboring process that witnesses such traffic to shutdown. This would reduce the chances of a user utilizing an anonymity ring with compromised ring-participants.
- Currently, the destination is able to talk back to the source only if the source encrypts its own handle and provides it to the destination. The anonymity ring could be attuned to provide this facility as an inherent feature. This will alleviate the overhead at the source process to allow for the response.

c. Acknowledgements

This research paper would not have been possible without the help of Dr. Gouda. He guided me in the right directions for research and helped me throughout in development and organization of this paper. I would like to thank him for his supervision and support. Additionally, I would like to thank all my friends and peers who contributed to this paper by proof-reading it and by identifying things that were inconsistent and those that needed improvement.

References

[1] Gouda, M. G. Elements of Network Protocol Design. John Wiley & Sons. 1998.

[2] Chaum, David L. "Untraceable electronic mail, return addresses, and digital pseudonyms." Communications of the ACM 24 (1981): 2:84-88.

[3] Jendricke, Uwe and Rannenberg, Kai. "A MixDemonstrator for teaching Security in the Virtual University" Proceedings of the IFIP TC11 WG 11.8 First World Conference on Information Security Education, (1999): 83-98.

[4] Michael G. Reed, Paul F. Syverson, and David M. Goldschlag. "Anonymous Connections and Onion Routing." IEEE Journal on Selected Areas in Communication Special Issue on Copyright and Privacy Protection. 1998.

[5] Serjantov, Andrei, Dingledine, Roger and Syverson, Paul. "From a Trickle to a Flood: Active Attacks on Several Mix Types." 5th Workshop on Information Hiding. Oct. 2002. LNCS (v.2578)

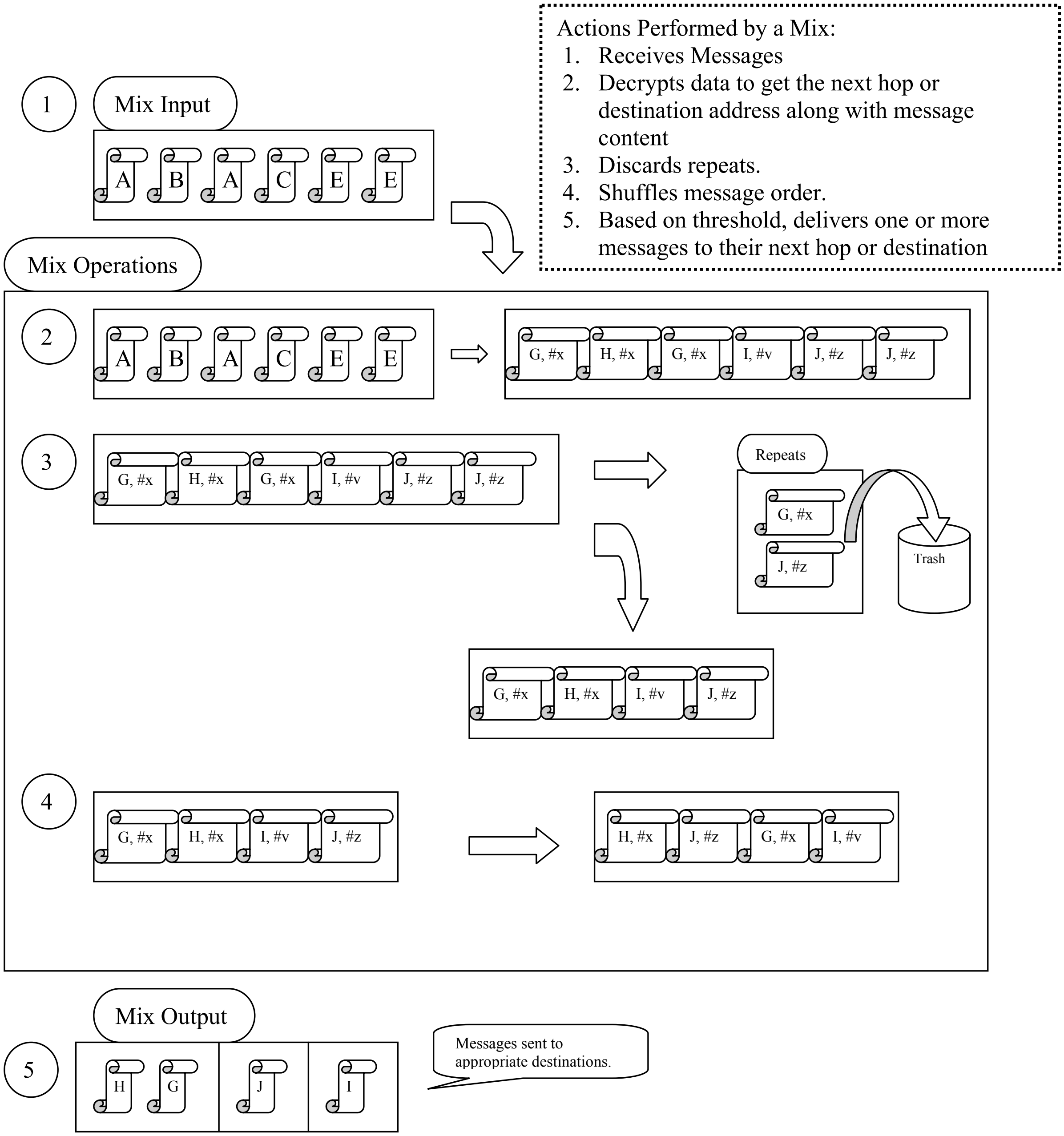
[6] Introduction to Public-Key Cryptography. < http://developer.netscape.com/docs/manuals/security/pkin/>. 9 Oct. 1998.

[7] NIST/SEMATECH e-Handbook of Statistical Methods, <http://www.itl.nist.gov/div898/handbook/eda/section3/eda366.htm>. 15 Nov. 2002.

[8] Onion Routing. < http://www.onion-router.net/>. 28 Jan. 2000.

Appendix A

Figure 1:



Actions Performed by a Mix:

1. Receives Messages
2. Decrypts data to get the next hop or destination address along with message content
3. Discards repeats.
4. Shuffles message order.
5. Based on threshold, delivers one or more messages to their next hop or destination

Figure 2:

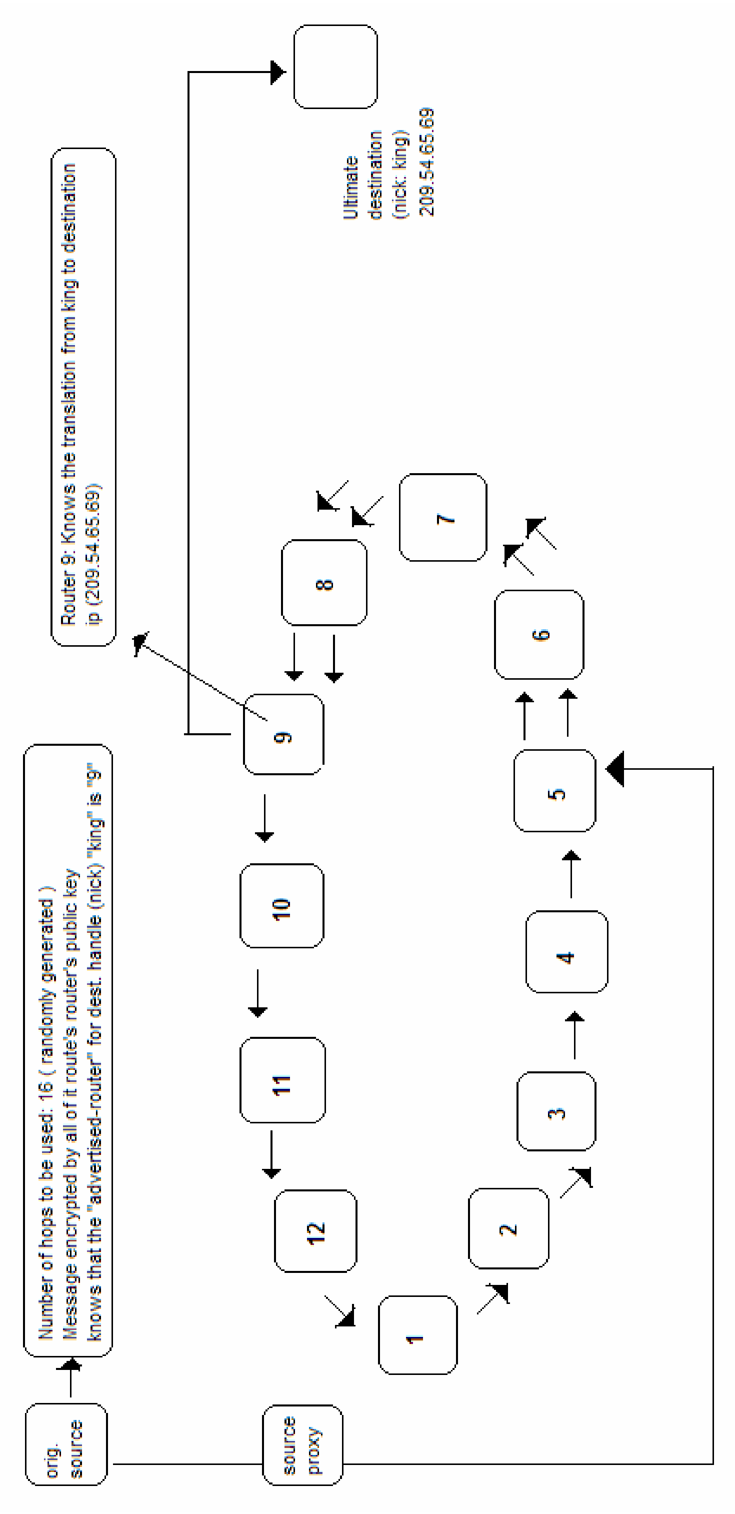
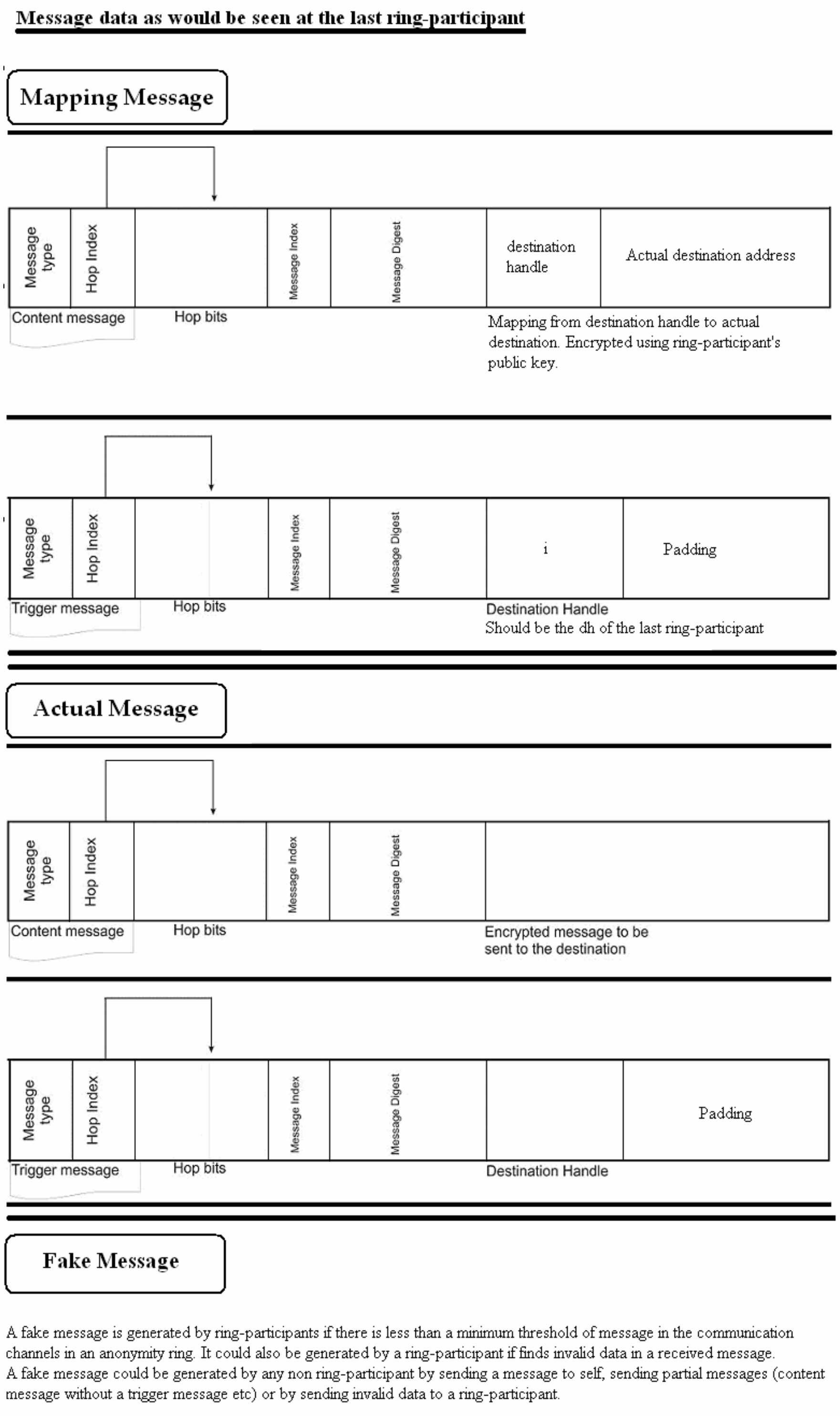
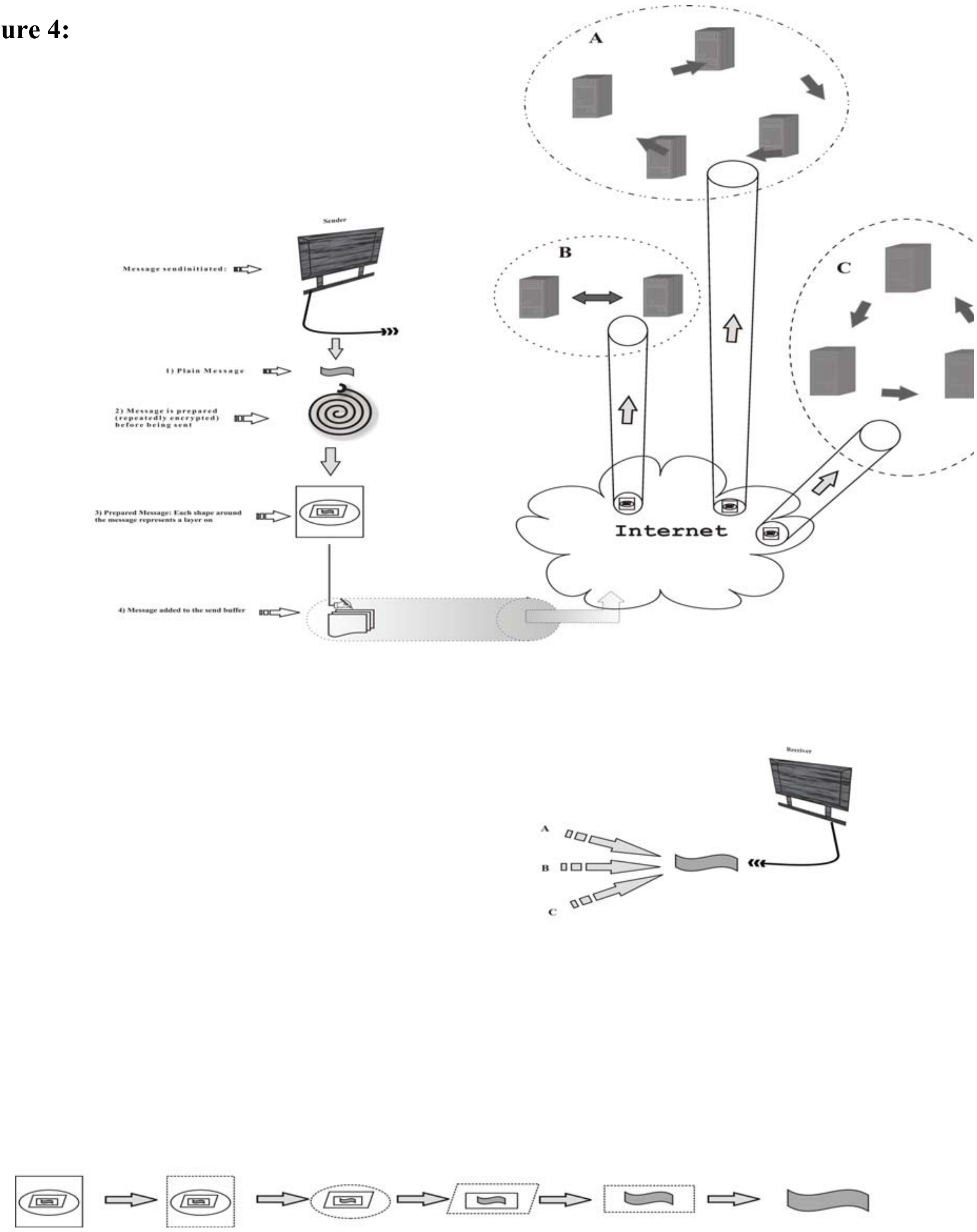

orig. source
Number of hops to be used: 16 (randomly generated)
Message encrypted by all of it route's router's public key
knows that the "advertised-router" for dest. handle (nick) "king" is "9"
source proxy
1
2
3
4
5
6
7
8
9
10
11
12
Router 9: Knows the translation from king to destination ip (209.54.65.69)
Ultimate destination (nick: king) 209.54.65.69

Figure 3:


Message data as would be seen at the last ring-participant

A fake message is generated by ring-participants if there is less than a minimum threshold of message in the communication channels in an anonymity ring. It could also be generated by a ring-participant if finds invalid data in a received message.
A fake message could be generated by any non ring-participant by sending a message to self, sending partial messages (content message without a trigger message etc) or by sending invalid data to a ring-participant.

Figure 4:


A
Sender
Message sendinitiated:
B
C
1) Plain Message
2) Message is prepared (repeatedly encrypted) before being sent
3) Prepared Message: Each shape around the message represents a layer on
Internet
4) Message added to the send buffer
Receiver
A
B
C