D. KRUTSKO S. SCHNEIDER A. SHUKLA

MAC **ATTACK**

MAC **ATTACK**

PROJECT DOCUMENTATION

github.com/dkrutsko

github.com/Harrold

github.com/AbsMechanik

Contents

INTRODUCTION 1
- Project Objectives 2
- Results Summary 2
- Report Outline 2

SYSTEMS DESIGN 3
- Design Strategies 5
  - Maskability 6
  - Accountability 7
  - Concealability 10
- Data Structures 13
  - Packet 13
  - Beacon 14
  - Message 14
- Class Diagram 15
- Class Structure 16
  - Address 16
  - Packet 17
  - Message 18
  - Identity 19
  - CRC32 20
  - Node 20
  - Onion Router 21

RESULTS 22
- Performance Based Metrics 23
- Security Based Metrics 24

CONCLUSION 29
- Future Development 30

REFERENCES 31

INTRODUCTION

Since the inception of “packet radio” networks (PRNET’s) in the early 1970’s, ad-hoc networking has gained popularity over the last few decades due to its unique nature of communication between wireless devices. Ad-hoc networking is a decentralized system of wireless communication, consisting of an autonomous system of nodes connected by wireless links without using any access points or base stations (infrastructure based networks). Each node participates in routing information for each other which allows for communication among nodes outside the wireless transmission range. Various types of routing protocols have been researched and implemented to provide maximum possible reliability and good throughput and route network traffic in the most efficient manner possible.

In the 21st century, ad-hoc networks are gaining even more momentum with the advent of vehicular communication technologies, virtual navigation, crisis management applications, defense applications and even education. Of these, vehicular communication and navigation and defensive applications are of keen interest due to the sensitive nature of the information being transmitted across the wireless spectrum. Like its counterparts, ad-hoc networks have also become prone to various forms of security threats in recent times. Due to the behaviour of communication between the nodes, it is extremely easy for an attacker to eavesdrop on a conversation between two or more parties. These types of vulnerabilities could potentially lead an attacker to perform a wide variety of attacks such as man-in-the-middle and/or impersonation against a target node and, consequently, compromise the integrity of the system. Therefore, it is essential to employ some form of security measures to the routing protocol itself to guard against such attacks and secure the integrity of the messages being transmitted.

PROJECT OBJECTIVES

The main objective of this endeavour was to design and implement an anonymous routing protocol which masks the identity of the users and conceal the messages being exchanged between them. To increase the legitimacy of the messages being exchanged and help establish trust between the nodes in the network it was deemed necessary to create a central, certification authority.

RESULTS SUMMARY

Based on data captured via Wireshark, we were able to successfully demonstrate the feasibility of our system. We also determined, theoretically, the shortcomings of using asymmetric cryptography and its potential ramifications on the performance. Despite these shortcomings, our simulations demonstrated that we were able to successfully mask the contents of our broadcasted message and also conceal the identity of our users. Additionally, while attacking the system is still hard, it was still vulnerable to the different traffic analysis strategies employed by attackers.

REPORT OUTLINE

This report describes the history behind the Onion Routing Protocol and its strategies for anonymously routing information across a network. This is followed by a detailed discussion of the design and implementation of the system and a comprehensive list of testing of the performance and security metrics. Finally, this report concludes with a discussion of potential applications and alternative implementations of the system followed by a list of references and cited works in journals, conferences and published papers.

SYSTEMS DESIGN

The Onion Routing Protocol, first presented in 1996, was designed to prevent any attacker from performing traffic analysis of a network. To help understand the need for such a protocol better, let us consider a simple analogy of a post office. Generally, a letter is placed in an envelope which is marked with the sender's and recipient's addresses. Naturally, the sender of the letter trusts that the post office does not peek inside the envelope (privacy reasons) and does not monitor the communication between it and the recipient. In the modern day context, this analogy can be applied to emails or any other form of electronic communication, thus, making it vital to protect the privacy of such messages. Granted that the communicating parties tend to reveal their identities to each other sometimes, there is no reason why the identities and the nature of the conversation between the aforementioned users should be revealed by the network to external parties. The protocol, therefore, addresses these two main problems in wireless security: eavesdropping and traffic analysis. Alternatively, this also led the researchers to devise a strategy to protect the identity of the users. This meant that not only were the contents of the message required to be protected but any identifying information attached to the message must also be protected and hidden from the external observer.

Subsequently, the researchers examined whether it was possible to devise a system whereby two parties can communicate with each other even if one or both of them do not want to be identified to the other and keep the contents of their communication private. The result of this ground-breaking research was the implementation of the first generation Onion Routing Protocol. The highlighting feature of this protocol was that even if an eavesdropper were to perform traffic analysis, the study of traffic patterns would not reveal much information about the paths of messages.

To initiate a communication, a simple plaintext message is encrypted (wrapped) with successive layers of encryption (onion) such that each layer can be decrypted (un-wrapped) by one intermediary (node or router) in a succession of intermediaries in the path taken by the onion routers (circuit). The message transmission is accomplished as follows:

- The sender picks nodes from a list which provide a path for transmitting the message.
- Using asymmetric cryptography, the sender encrypts the message with the public keys of the chosen nodes which are obtained from an advertised list.
- As the message passes through each node, a layer of encryption is removed by the receiving nodes (by using their private keys) until it reaches its destination.

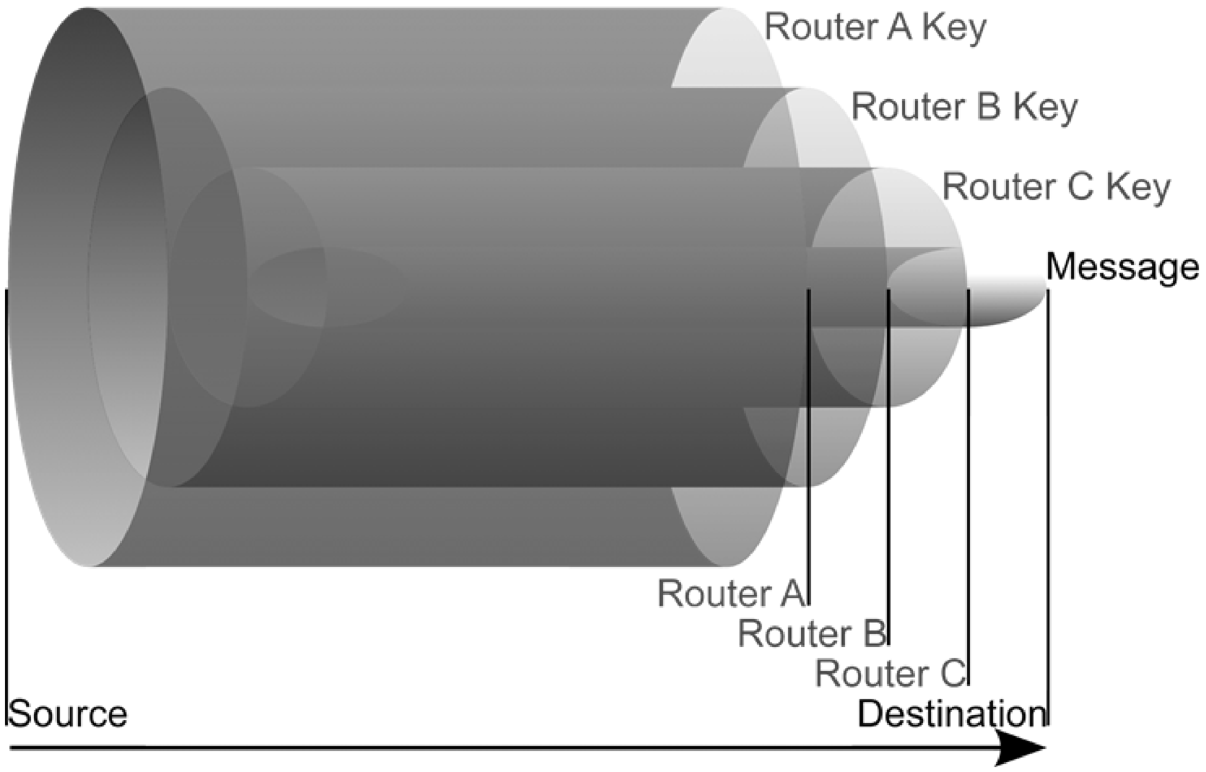


Figure 1: A typical onion packet [9]

Many modern applications such as web browsing and chat (MSN, IRC, etc.) apply this strategy to communicate both privately and anonymously. One of the most successful applications of this research is the Tor Project.

The Tor Project (also known as second/third generation Onion Routing Protocol) was initially designed, developed and deployed for the US Navy, but is now widely used by all branches of the military, law enforcement agencies, journalists and even normal people for daily use around the world. Given the potential for such a system, it was deemed important to design and develop such a system for a mobile ad-hoc network (MANET) due to its gaining popularity and widespread applications.

DESIGN STRATEGIES

The Zone Routing Protocol (ZRP) is a hybrid implementation of an ad-hoc network which integrates ideas from both reactive and proactive protocols. It consists of three sub-protocols, as outlined below:

- Neighbour Discover Protocol (NDP)
- Intrazone Routing Protocol (IARP)
- Interzone Routing Protocol (IERP)

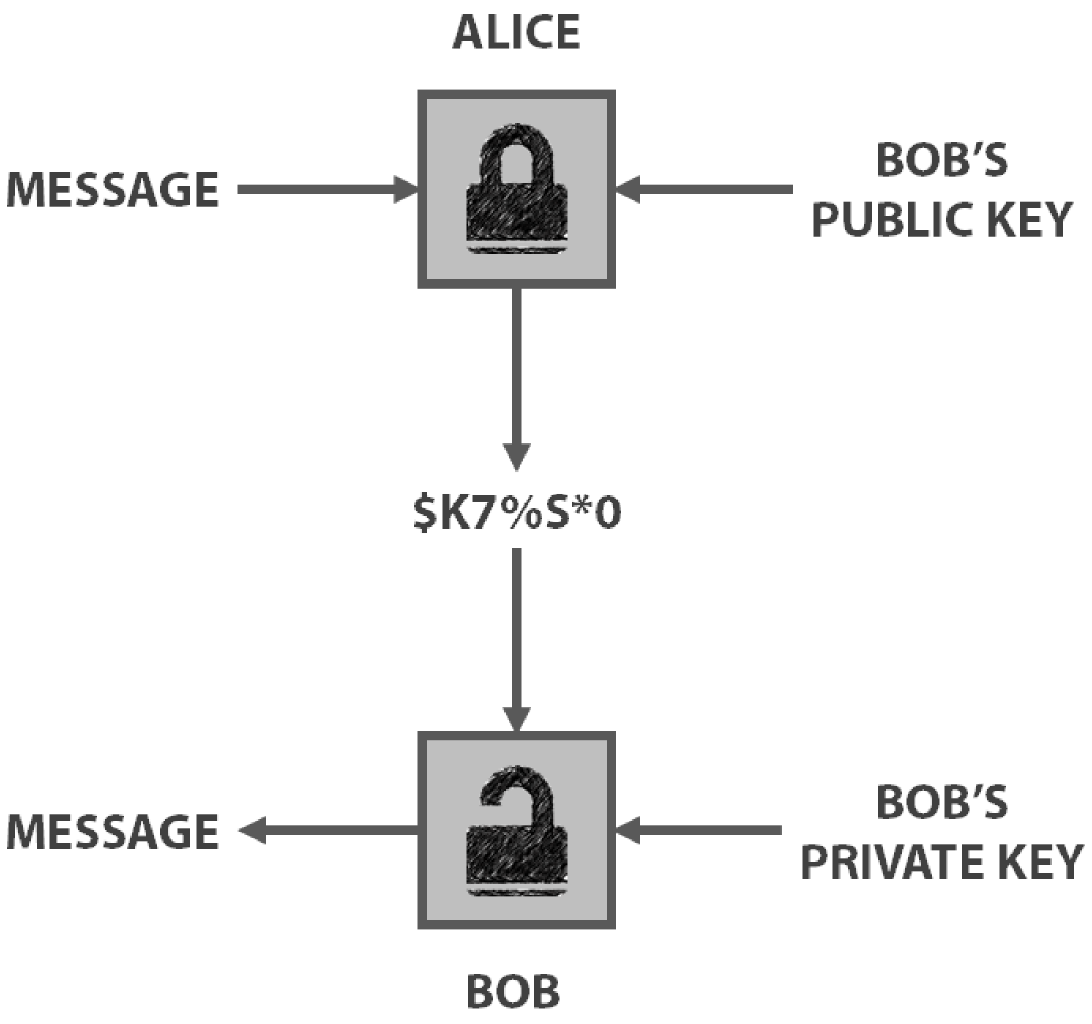
The NDP is used to discover one-hop neighbours and essentially stores the information about its neighbourhood, which contains a list of all the broadcasting nodes. By adopting this strategy, our protocol could maintain and update its list of neighbours while also enabling a user to pick a destination address and send a message across the network.

In addition to maintaining a list of neighbours, the protocol also needed to conceal the identities of the users themselves and transmit messages anonymously while ensuring that the transmitting node was certified by a central authority. Thus, we divided our implementation strategy into three phases:

1. Maskability
2. Accountability
3. Concealability

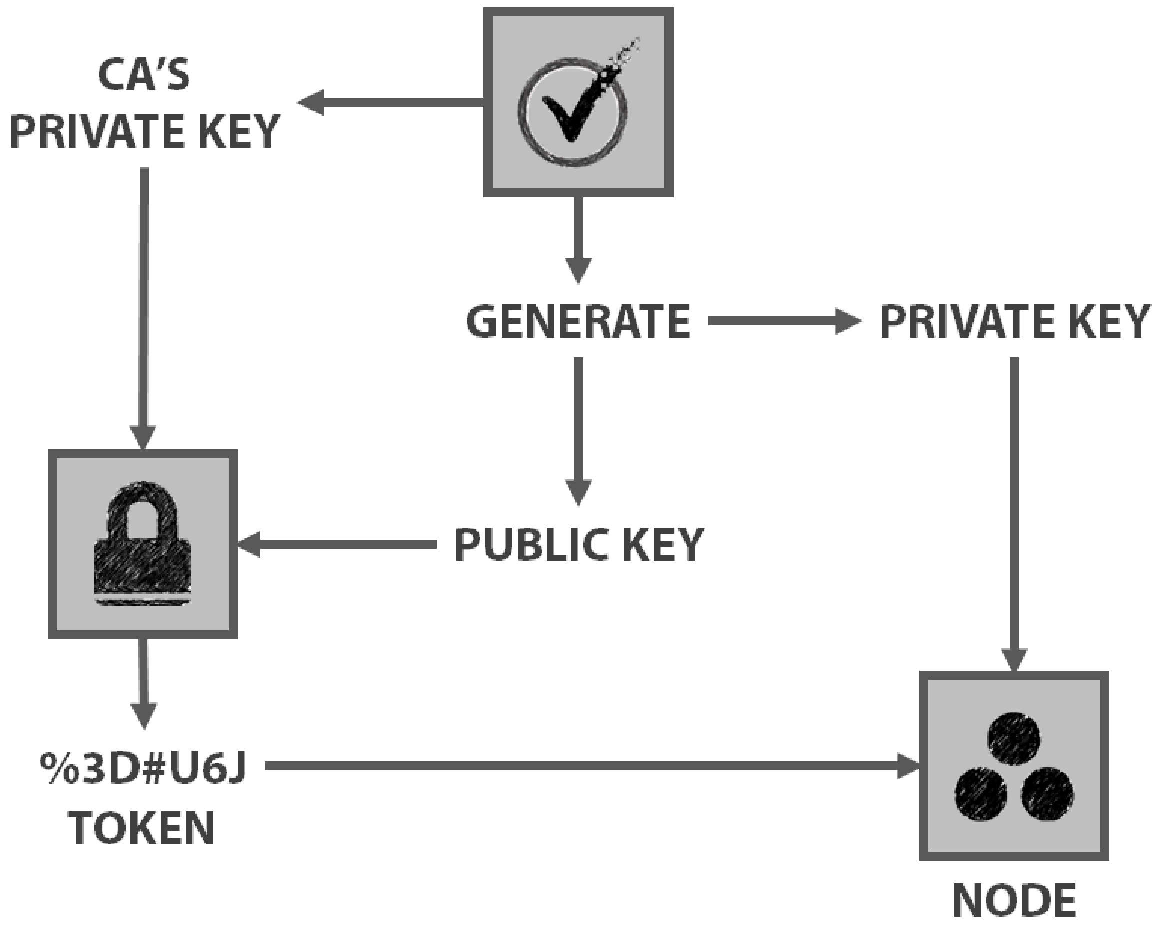
MASKABILITY

To mask the contents of the messages, public-key cryptography strategies were implemented (PKI). The figure below describes the process of a typical asymmetric encryption process:



ACCOUNTABILITY

To maintain and establish trust between the nodes within the network, we designed and implemented a certification authority (CA) system. Every single node has access to the CA’s public key which was later used by the nodes themselves to encrypt the messages being transmitted. The certification process works as depicted below:



In the first stage, the application generates the CA's key and is depicted below:

```
N  = A06D4257F75B937F66404B43AF49371F240EC7E32079574517F073B598A70EC3
D  = 72A07ED5F1C53A27BE7CD40A61B593AF03386BA1C37D810033DAA16ADC5259D1
P  = F2A9A482657D4CEA5F9B42D84782BE19
Q  = A93E86AC3795DAF7F7587567FE19173B
DP = A4B6BC00FAECCB79C77DB728E1998079
DQ = A367C74787F39C1346319649EDBFF3F9
QP = 3C4B4F872BA41D13BD0127371825167B
```

```
$ ./MacAttack Create 256 Auth
```

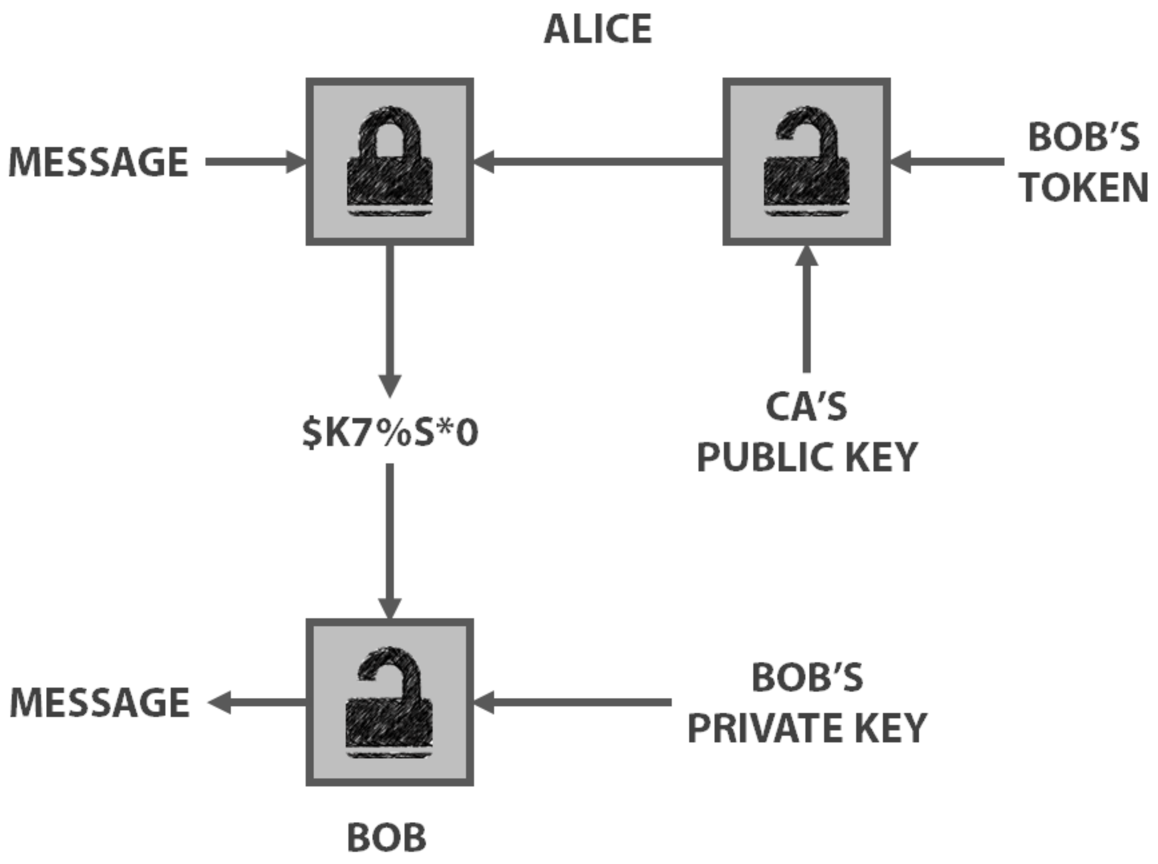
Once the CA's public key has been made available to the nodes, the only way a node can initiate a communication with another node is by signing its public key with the CA's public key. In other words, this ensures that the node is authorized to communicate in the network and its messages are deemed trustworthy. One thing to bear in mind is that all the nodes must use the same level of encryption to initiate a two-way conversation, i.e., a 256-bit encrypted node cannot initiate a conversation with a 128-bit node and vice-versa. This is due to the encryption-decryption process utilized by the RSA algorithm.

The second stage involves creating a node within the network and signing the node with the CA's public key. This procedure is depicted as follows:

```
N   = 994C34FBD514057D527A89657E04D40F
D   = 8C5DFC51EFBCC27CA08403263E8D5469
P   = C67F831CA46311B3
Q   = C5B4A981845C2E35
DP  = 40A7DA2A23358DD7
DQ  = 863CBE82D42E0AA9
QP  = 509855E232356618
N_A = A06D4257F75B937F66404B43AF49371F240EC7E32079574517F073B598A70EC3
N_N = 2CC290FB41E302A190046FE4901139E92EA6A8622DD77537E53365868D09E94C
```

```
$ ./MacAttack Create 128 Node
$ ./MacAttack Sign   Auth Node
```

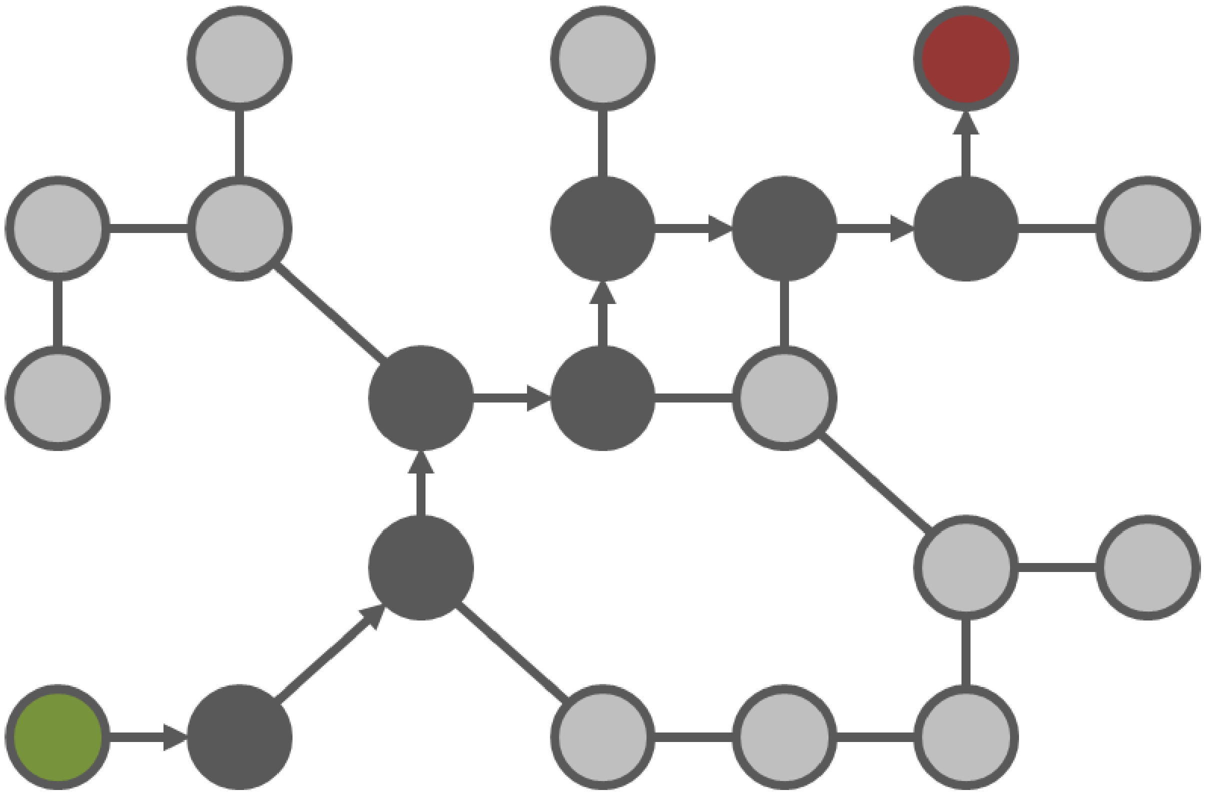
Finally, once all the nodes have been signed and authorized, they are free to communicate with each other:



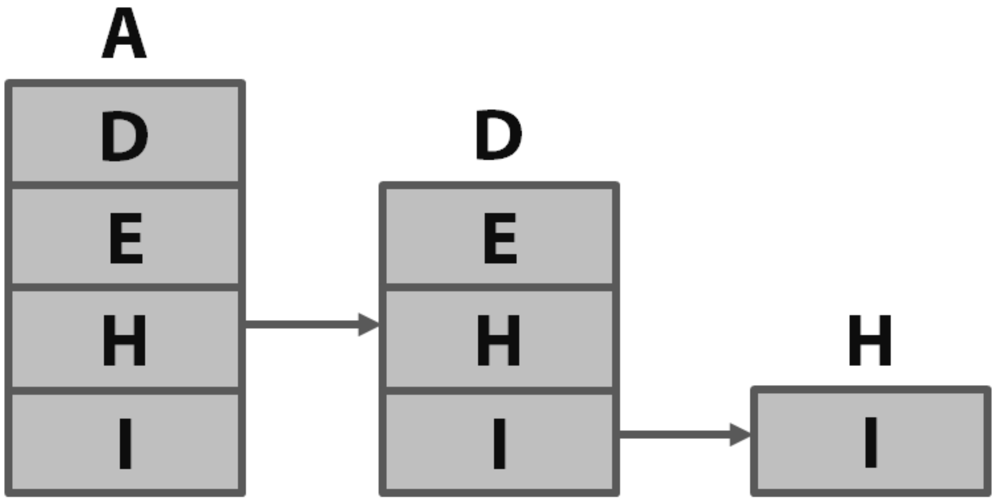
CONCEALABILITY

The main purpose of this criterion is to obscure the path taken by nodes to transmit the message. Ideally, this feature helps discourage attackers from gaining any useful information by performing traffic analysis. Moreover, it also makes eaves-dropping an infeasible solution to determine the nature of communication between multiple parties.

To better understand the nature of this problem, let us consider a simple scenario. Let us assume that Alice wants to send a message to Bob. Under normal circumstances, Alice can transmit a message to Bob by using any standard routing protocol which will simply broadcast its message to its neighbours until it reaches its target (multi-hopping):



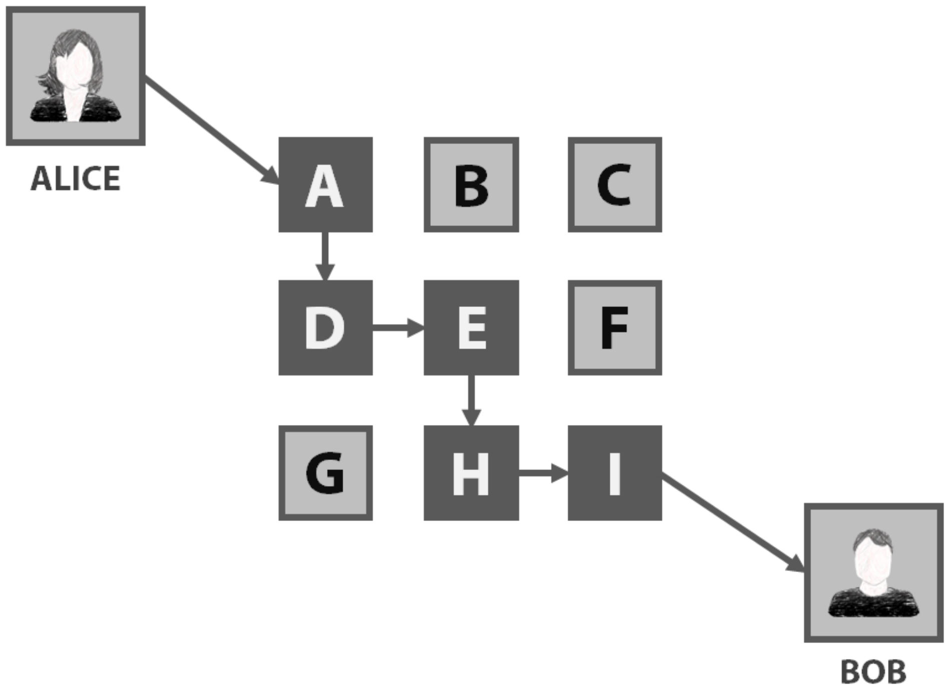
To address this problem, we decided to encrypt the message with multiple layers of encryption. In any given network, every authorized node can simply display a list of active nodes in the zone (NDP), which is all the information it needs to transmit a message to any existing node in the network. Ideally, the node then builds a “circuit” to transmit its message to its destination, while encrypting the message (in order) with the public keys of the nodes in the “circuit”. This process is described as depicted below:



The encryption is performed by the following nested loop algorithm as depicted below:

```
$ Send [Destination] [Message ...]

Find Destination Address in the Table
    Adjust Message to be of Same length as Key
    Encrypt Message with Destination Key
    Loop Though Destination Address Path
        Find Address in Address Path
            Encrypt Message with Address Key
```

Once the message has been encrypted, Alice can communicate freely with Bob, anonymously. Our approach also conceals the identity of the users themselves (unless they choose to reveal their identities):



DATA STRUCTURES

Various data structures are using throughout the protocol in order to make it efficient to send data from one node to another. This section will discuss three major data structures including the packet declaration, structure of a beacon and message. Implementation details are described in the next section.

PACKET

The first data structure is the packet. There is only one packet type in the protocol which handles both beacons and messages. The way to differentiate between the two is to look at the IPType, if it equals 0x3950 then the packet is a beacon. Beacon packets contain the public key of the source node in Message while also including the path of that beacon in the AddressPath. Beacon packets have no hashes.

Messages have an IPType of 0x3960. The AddressPath is empty however the hashes list contains the hash results for decrypted messages. This is so that decryption can be checked for whether or not the message was successfully decrypted.

```
Address      Target
Address      Source
UInt16       IPType
UInt32       MessageLength
UInt32       AddressLength
UInt32       HashesLength
UInt8*       Message
Address*     AddressPath
UInt32*      Hashes
```

BEACON

Below is a sample beacon packet as described above. When AddressPath is empty that means that the nodes are direct neighbors. The hashes list is not used.

```
Target          = Broadcast
Source          = Source Address
IPType          = 0x3950
MessageLength   = Key Size
AddressLength   = 0
HashesLength    = 0
Message         = Public Key (N)
AddressPath     = null
Hashes          = null
```

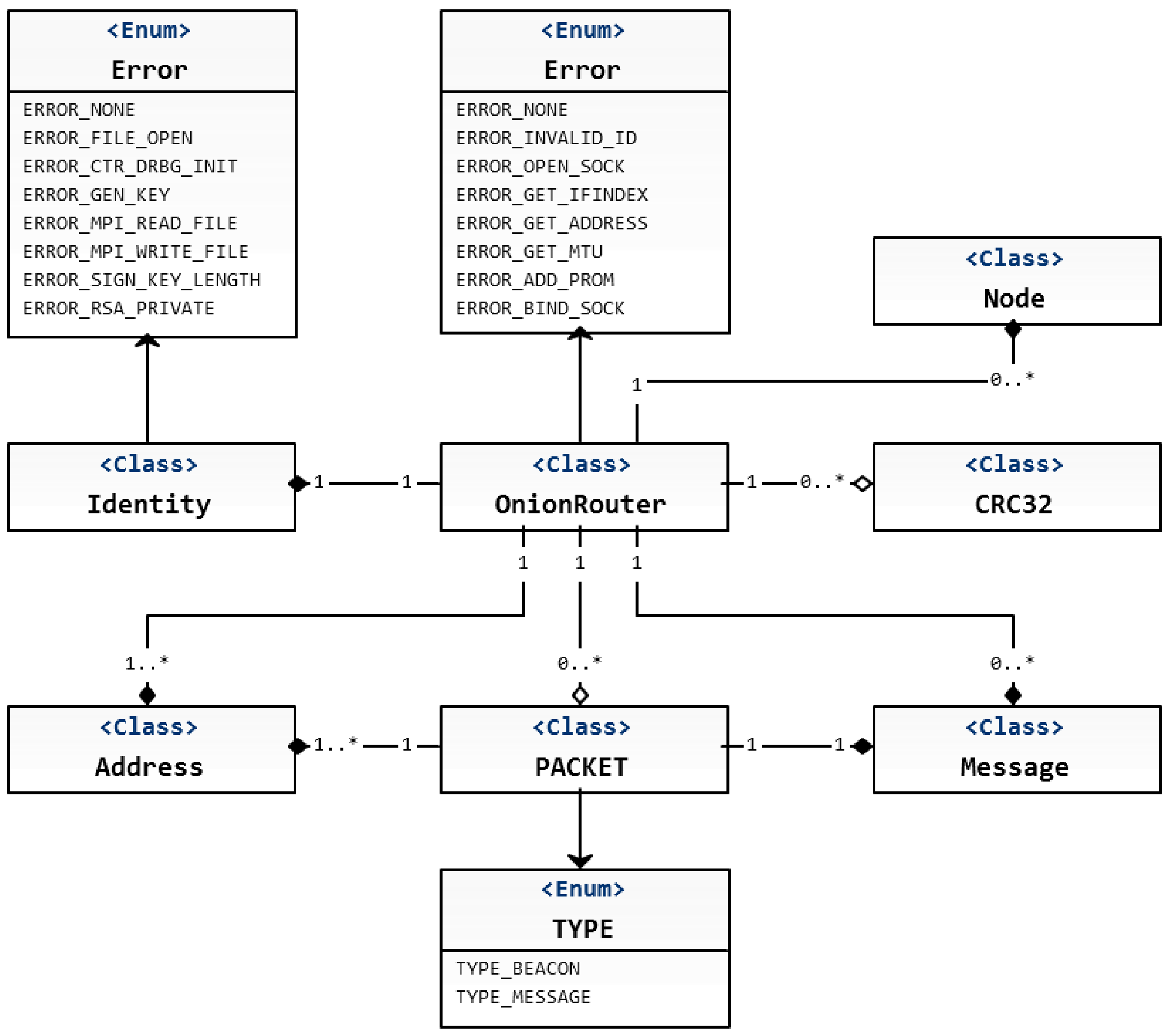
MESSAGE

Below is a sample message packet as described above. AddressPath is not used. Hashes contains the list of hashes, each rebroadcast removes a hash from the top.

```
Target          = Broadcast
Source          = Source Address
IPType          = 0x3960
MessageLength   = Message Size
AddressLength   = 0
HashesLength    = Hashes Length
Message         = Encrypted Message
AddressPath     = null
Hashes          = null
```

CLASS DIAGRAM

This section describes the structure of the overall system as well as the interaction between classes. The data is presented in a simplified fashion to better emphasise relationships throughout the system. Please refer to the next section in order to get more detailed information about the internal workings of individual classes.

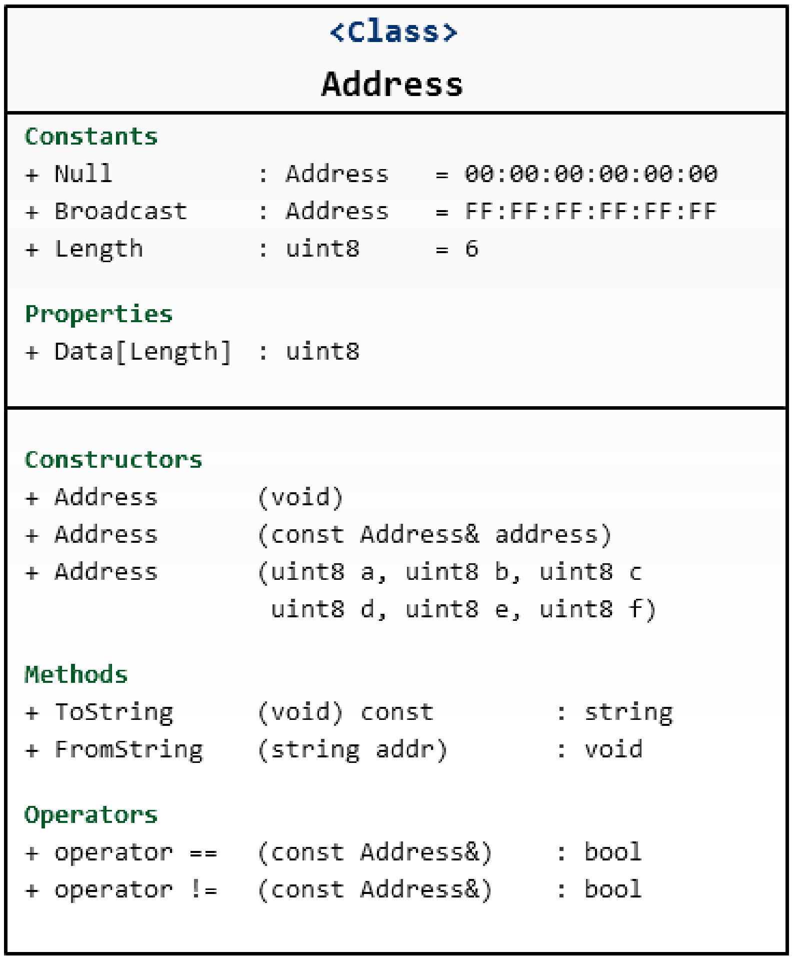


CLASS STRUCTURE

This section describes the internal workings of classes detailed in the previous section. Each class is split up into one or more subgroups depending on the type of data being portrayed. Some information is omitted as it does not add to the overall representation of the class.

ADDRESS

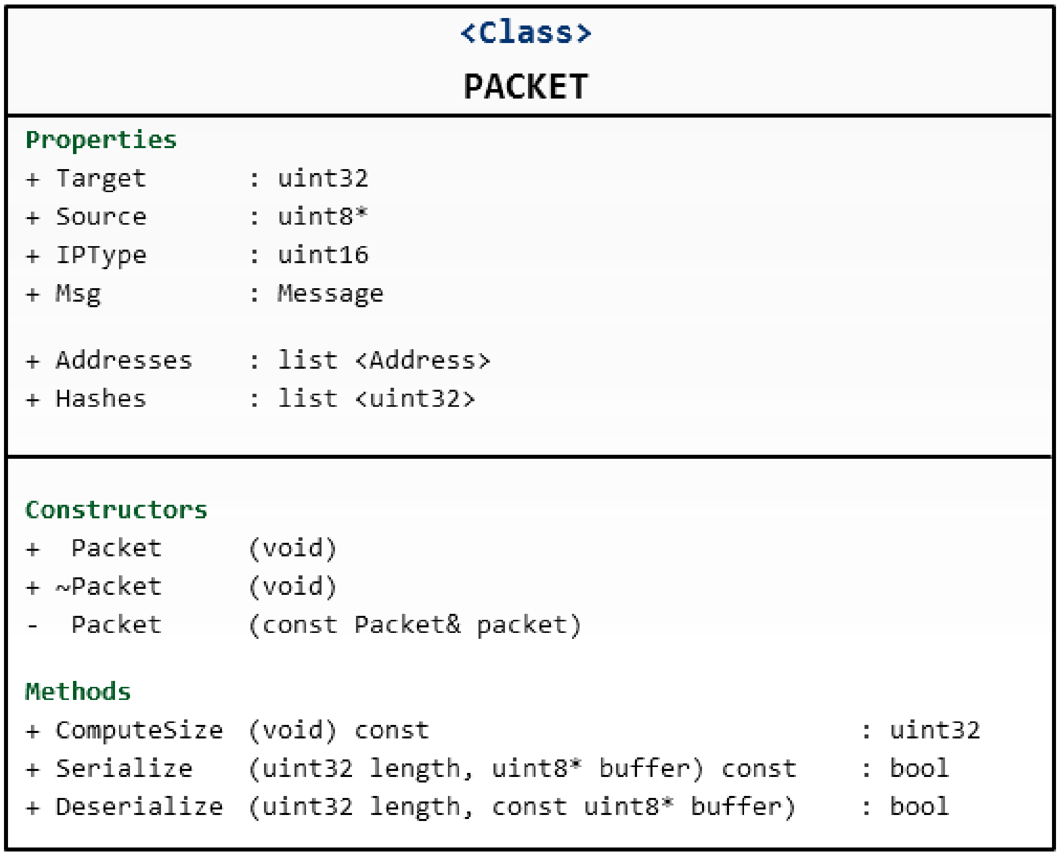
This class represents a single address. An address is typically designed to symbolize a devices' mac address, this class is used throughout the program to denote source and destination mac addresses and is always six bytes long.



```
<Class>
Address

Constants
+ Null          : Address    = 00:00:00:00:00:00
+ Broadcast     : Address    = FF:FF:FF:FF:FF:FF
+ Length        : uint8      = 6

Properties
+ Data[Length]  : uint8

Constructors
+ Address       (void)
+ Address       (const Address& address)
+ Address       (uint8 a, uint8 b, uint8 c
                 uint8 d, uint8 e, uint8 f)

Methods
+ ToString      (void) const        : string
+ FromString    (string addr)       : void

Operators
+ operator ==   (const Address&)    : bool
+ operator !=   (const Address&)    : bool
```

PACKET

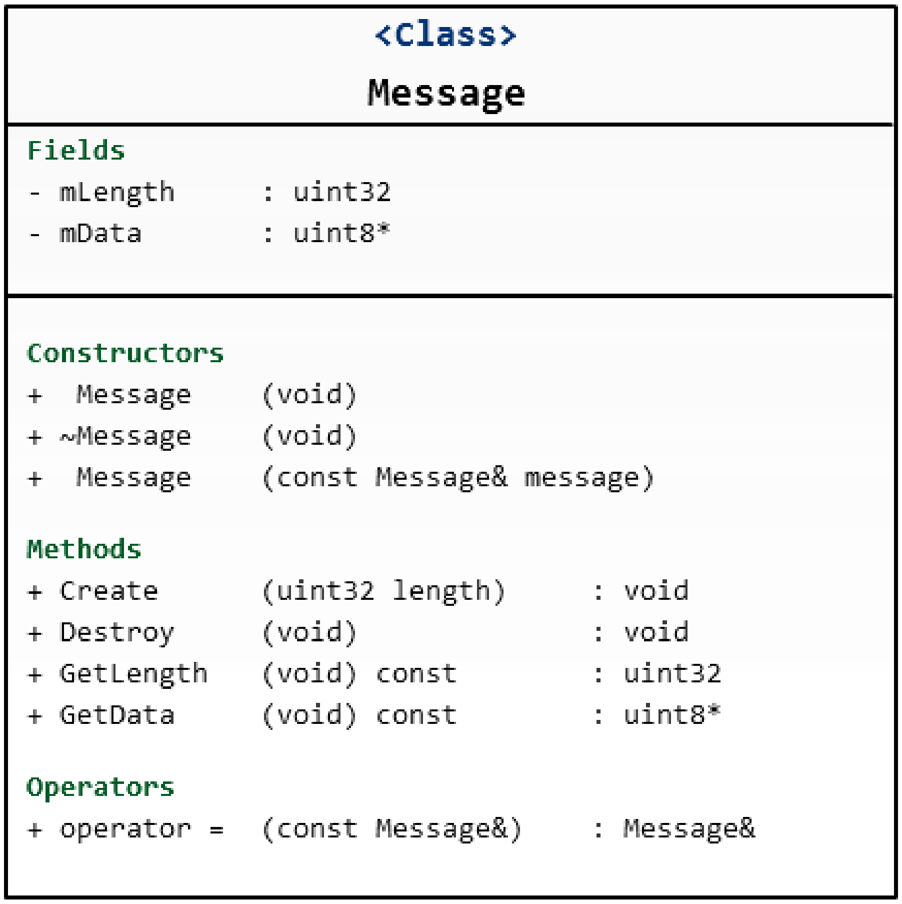
This class represents a single packet. Each instance of this class inputs the data which needs to be sent and is then responsible for serializing it in the correct format based on the data structure discussed in a previous section. This class can also deserialize data which has been previously serialized.



```
<Class>
PACKET

Properties
+ Target       : uint32
+ Source       : uint8*
+ IPType       : uint16
+ Msg          : Message

+ Addresses    : list <Address>
+ Hashes       : list <uint32>

Constructors
+  Packet      (void)
+ ~Packet      (void)
-  Packet      (const Packet& packet)

Methods
+ ComputeSize  (void) const                           : uint32
+ Serialize    (uint32 length, uint8* buffer) const   : bool
+ Deserialize  (uint32 length, const uint8* buffer)   : bool
```

MESSAGE

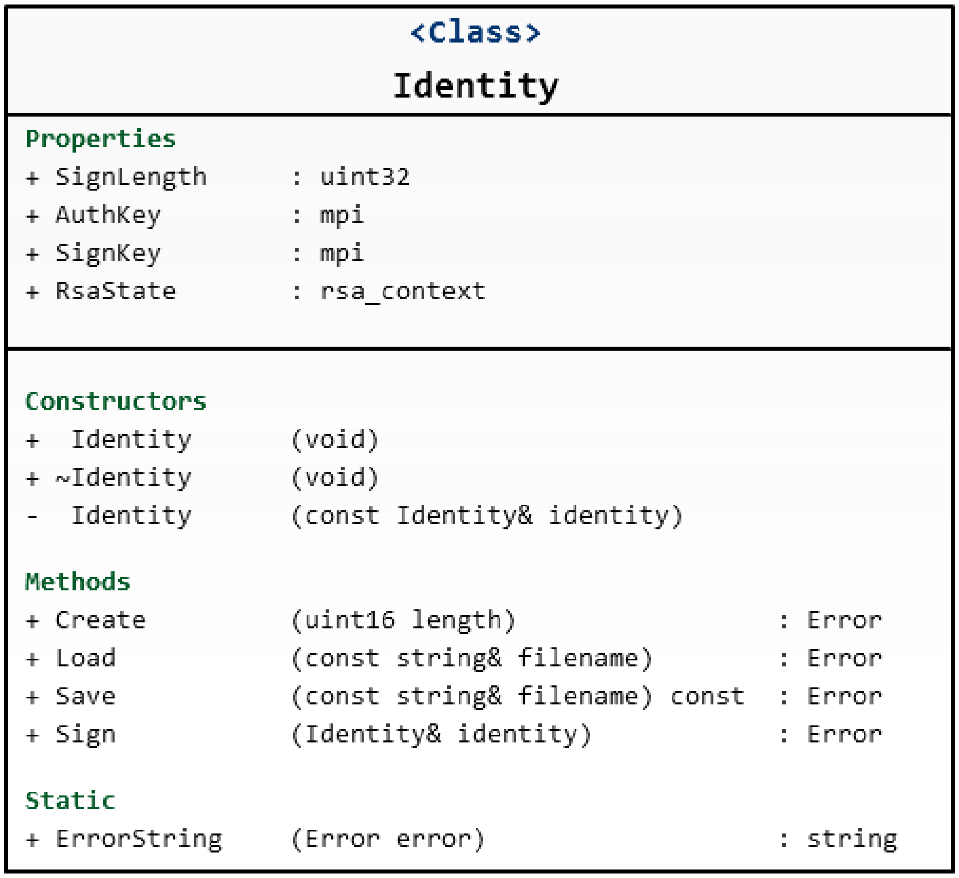
This class represents a single message. A message can be encrypted and sent to other users on the network where is will be decrypted and processed. No message is sent if a node is sending out a beacon packet.



```
<Class>
Message

Fields
- mLength      : uint32
- mData        : uint8*

Constructors
+  Message     (void)
+ ~Message     (void)
+  Message     (const Message& message)

Methods
+ Create       (uint32 length)      : void
+ Destroy      (void)               : void
+ GetLength    (void) const         : uint32
+ GetData      (void) const         : uint8*

Operators
+ operator =   (const Message&)     : Message&
```

IDENTITY

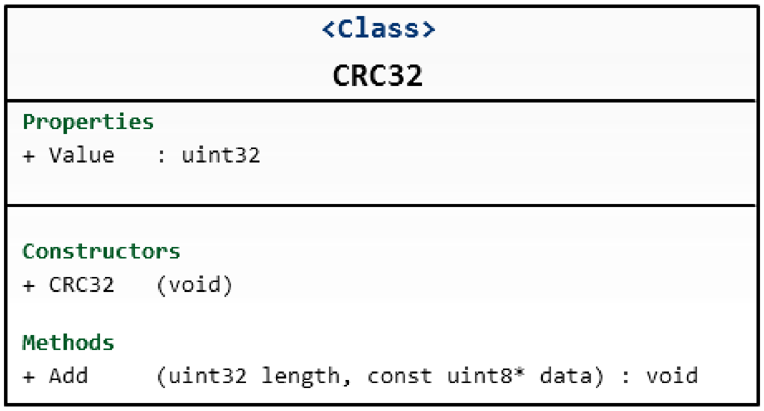
This class represents a single identity. In order to sign onto the network, a node must have an identity signed by an authority and cannot be self-signed. This class provides the methods for creating, saving, loading and signing identities.



```
<Class>
Identity

Properties
+ SignLength      : uint32
+ AuthKey         : mpi
+ SignKey         : mpi
+ RsaState        : rsa_context

Constructors
+  Identity       (void)
+ ~Identity       (void)
-  Identity       (const Identity& identity)

Methods
+ Create          (uint16 length)                : Error
+ Load            (const string& filename)       : Error
+ Save            (const string& filename) const : Error
+ Sign            (Identity& identity)           : Error

Static
+ ErrorString     (Error error)                  : string
```

CRC32

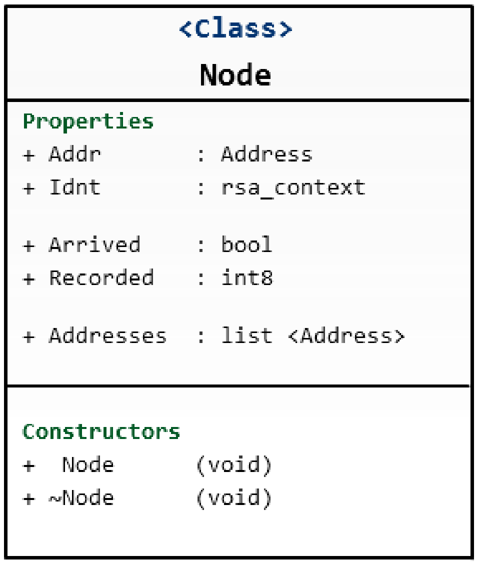
This is a utility class designed to compute the hash value of a specified message using the CRC-32 hashing algorithm. Each instance of this class is standalone and is capable of storing a running hash of one or more messages. This class is used to compute message hash values before they get encrypted and is used to verify whether decryption was successful.



```
<Class>
CRC32

Properties
+ Value    : uint32

Constructors
+ CRC32    (void)

Methods
+ Add      (uint32 length, const uint8* data) : void
```

NODE

This class is part of OnionRouter and represents a single node in the network. It stores all the necessary information about a node including its source address, NDP data and the path it took to reach its destination. This class also stores the public key of the source node so that messages can be encrypted.



```
<Class>
Node

Properties
+ Addr         : Address
+ Idnt         : rsa_context

+ Arrived      : bool
+ Recorded     : int8

+ Addresses    : list <Address>

Constructors
+  Node        (void)
+ ~Node        (void)
```

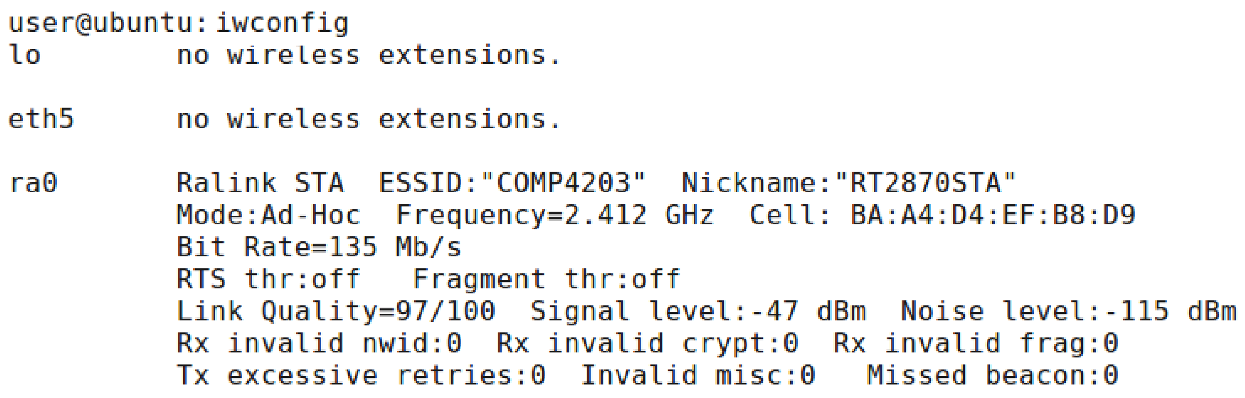
ONION ROUTER

This class represents the core component of the Mac Attack onion routing protocol. Each instance of this class is a node in the network capable of sending and receiving beacons and messages. A single machine can represent multiple nodes. This class handles the entire networking backend.

```
                              <Class>
                            OnionRouter
Properties
+ Network           : list <Node*>

Fields
- mAuthority        : rsa_context

- mIdentity         : Identity*
- mAddress          : Address

- mMTU              : int32
- mSocketID         : int32
- mIfIndex          : int32

- mDest             : sockaddr_ll
- mDestLength       : uint8

- mSendThread       : pthread_t
- mRecvThread       : pthread_t
- mMutex            : pthread_mutex_t
- mActive           : volatile bool

- mMessages         : list <Message*>
- mIgnore           : list <Address*>

Constructors
+  OnionRouter      (void)
+ ~OnionRouter      (void)

Methods
+ Create            (const string& interface, Identity* identity)   : Error
+ Destroy           (void)                                          : void

+ Start             (void)                                          : void
+ Stop              (void)                                          : void
+ IsActive          (void) const                                    : void

+ Send              (const Address& address, const Message& msg)    : bool
+ Recieve           (void)                                          : Message*
+ Flush             (void)                                          : void

+ Lock              (void)                                          : void
+ Unlock            (void)                                          : void

+ ReadIgnoreList    (const string& filename)                        : void

Static
+ ErrorString       (Error error)                                   : string
```

RESULTS

Based on our implementation strategies, we decided to use both performance based and security based metrics to analyse and test the feasibility of our system. For our simulations, we made use of the machines located in HP 4175 (Herzberg Laboratories) located at Carleton University. Each workstation was equipped with a wireless D-Link adapter, which can be used to create a smaller scale ad hoc network, and a Linux VM.



```
user@ubuntu: iwconfig
lo        no wireless extensions.

eth5      no wireless extensions.

ra0       Ralink STA  ESSID:"COMP4203"  Nickname:"RT2870STA"
          Mode:Ad-Hoc  Frequency=2.412 GHz  Cell: BA:A4:D4:EF:B8:D9
          Bit Rate=135 Mb/s
          RTS thr:off   Fragment thr:off
          Link Quality=97/100  Signal level:-47 dBm  Noise level:-115 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:0
```

We also made extensive use of Wireshark in order to monitor and capture the flow of packets being transmitted and to analyze the structure of each packet to determine if any useful information could be obtained

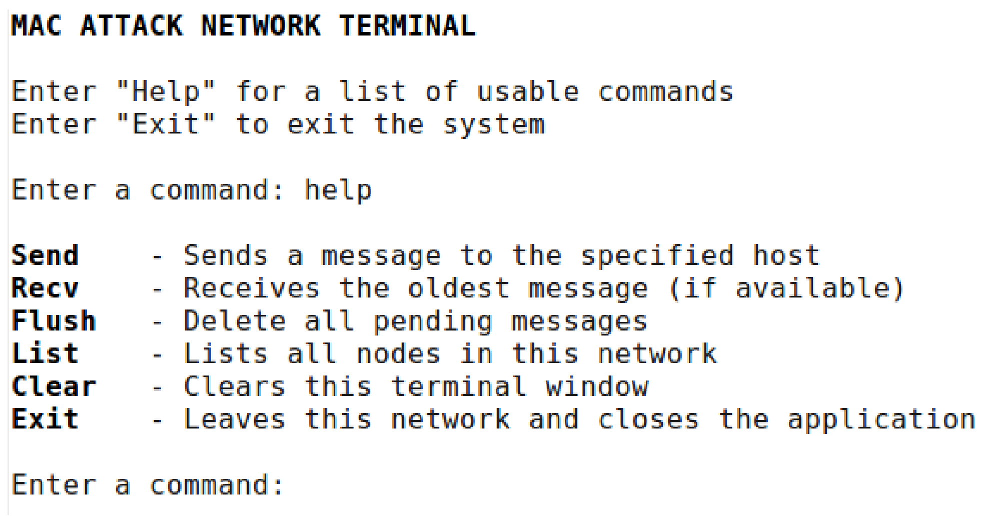
PERFORMANCE BASED METRICS

Our packet essentially consists of a simple message wrapped in successive layers of encryption. RSA encryption and decryption can be a tedious process and may influence the performance of the network. To successfully apply multiple layers of encryption to our message, our "encryption layering" algorithm made use of nested loops. This unfortunately yielded an O (N^3) algorithm, which is deemed poor by our performance requirements. During a live demonstration, there was little noticeable difference in performance as we chose a direct path to transmit a message. This problem would most certainly be noticeable if we were to employ a multi-path route. In addition to this, the extra overhead incurred due to the encryption and decryption at various stages, we hypothesize that this would most certainly have a severe impact on the performance within the network. Wireshark, unfortunately, was unable to provide us with any useful data to analyze the performance of packet delivery. However during our simulations, we did not notice any issues with performance since the application was deployed on an extremely small scale capacity and the lab environment only contained a handful of nodes for establishing our routing protocol.

We were, however, able to procure enough data by using Wireshark. By inspecting each, individual packet and analyzing the information presented by Wireshark, we could perform a small scale traffic analysis to try and determine the viability of the protocol itself. This naturally, leads us to perform a security metrics analysis with the help of packet analyzers built into Wireshark.

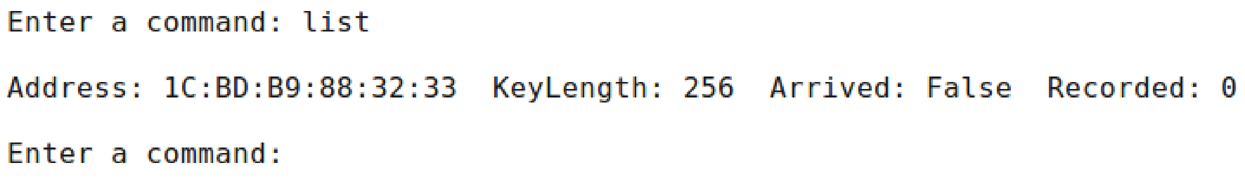
SECURITY BASED METRICS

Instead of performing an actual attack, due to limited resources, we decided to capture and analyze packets of information to see what information it revealed. The picture below demonstrates our application in action. Once a node has been created and certified, it can participate in a conversation with another node via our application:



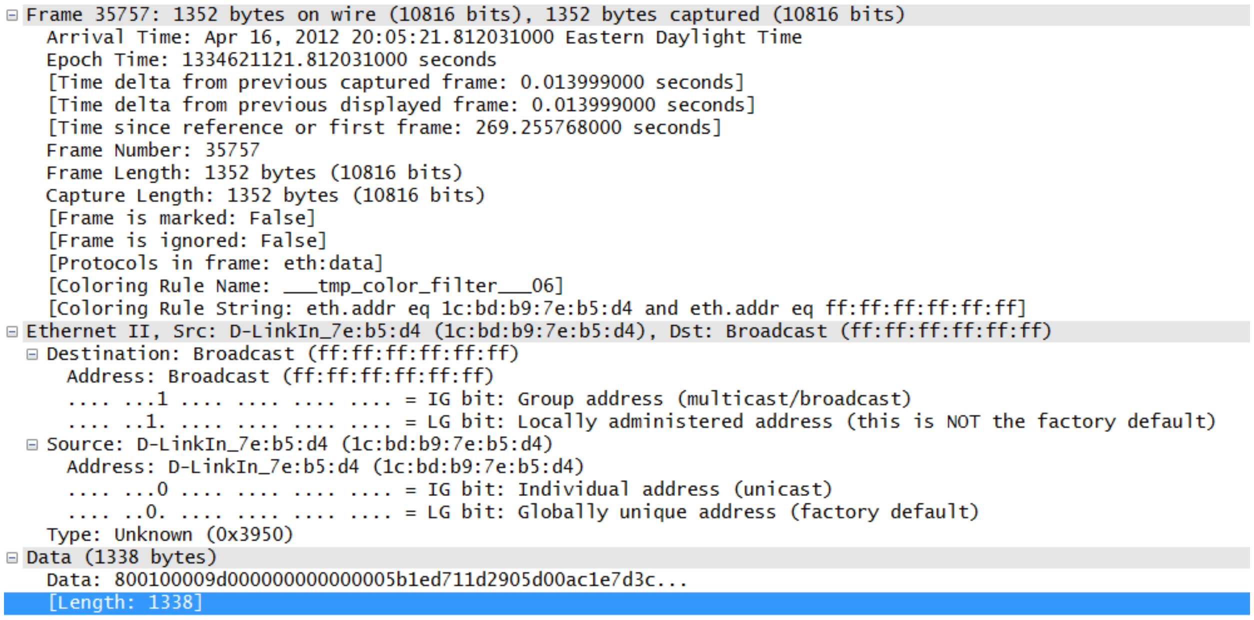
```
MAC ATTACK NETWORK TERMINAL

Enter "Help" for a list of usable commands
Enter "Exit" to exit the system

Enter a command: help

Send    - Sends a message to the specified host
Recv    - Receives the oldest message (if available)
Flush   - Delete all pending messages
List    - Lists all nodes in this network
Clear   - Clears this terminal window
Exit    - Leaves this network and closes the application

Enter a command:
```

As soon as a node joins the network, it starts broadcasting a beacon to its neighbours as is required by the NDP protocol. The built-in NDP protocol gathers and displays a list of active nodes within the network.



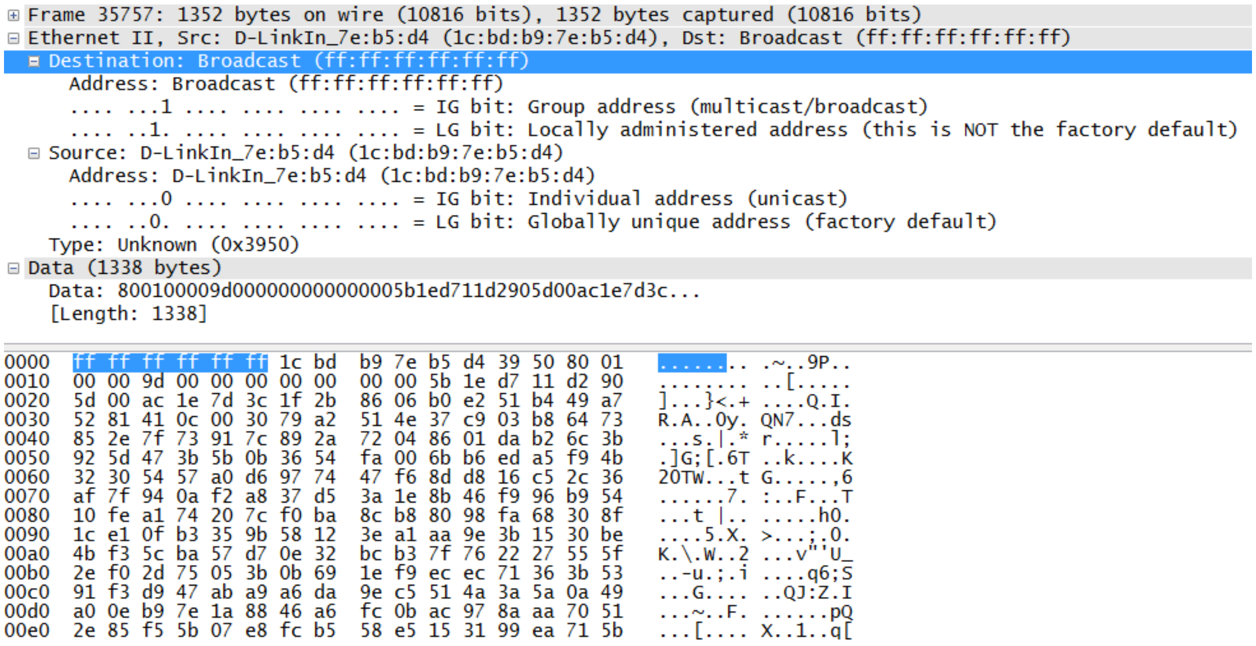
```
Enter a command: list

Address: 1C:BD:B9:88:32:33  KeyLength: 256  Arrived: False  Recorded: 0

Enter a command:
```

Below is a representation of what a typical broadcast looks like in Wireshark:



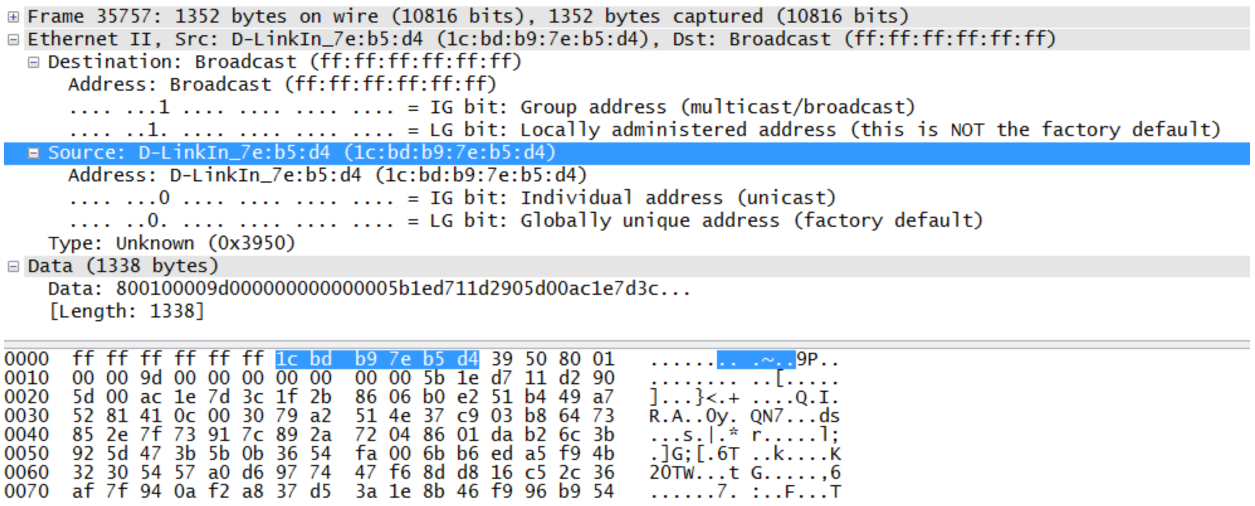
```
⊟ Frame 35757: 1352 bytes on wire (10816 bits), 1352 bytes captured (10816 bits)
    Arrival Time: Apr 16, 2012 20:05:21.812031000 Eastern Daylight Time
    Epoch Time: 1334621121.812031000 seconds
    [Time delta from previous captured frame: 0.013999000 seconds]
    [Time delta from previous displayed frame: 0.013999000 seconds]
    [Time since reference or first frame: 269.255768000 seconds]
    Frame Number: 35757
    Frame Length: 1352 bytes (10816 bits)
    Capture Length: 1352 bytes (10816 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:data]
    [Coloring Rule Name: ___tmp_color_filter___06]
    [Coloring Rule String: eth.addr eq 1c:bd:b9:7e:b5:d4 and eth.addr eq ff:ff:ff:ff:ff:ff]
⊟ Ethernet II, Src: D-LinkIn_7e:b5:d4 (1c:bd:b9:7e:b5:d4), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  ⊟ Destination: Broadcast (ff:ff:ff:ff:ff:ff)
      Address: Broadcast (ff:ff:ff:ff:ff:ff)
      .... ...1 .... .... .... .... = IG bit: Group address (multicast/broadcast)
      .... ..1. .... .... .... .... = LG bit: Locally administered address (this is NOT the factory default)
  ⊟ Source: D-LinkIn_7e:b5:d4 (1c:bd:b9:7e:b5:d4)
      Address: D-LinkIn_7e:b5:d4 (1c:bd:b9:7e:b5:d4)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
    Type: Unknown (0x3950)
⊟ Data (1338 bytes)
    Data: 800100009d000000000000005b1ed711d2905d00ac1e7d3c...
    [Length: 1338]
```

The captured packet reveals the destination address, source address, data length and message type. The destination of a broadcast packet is depicted below:



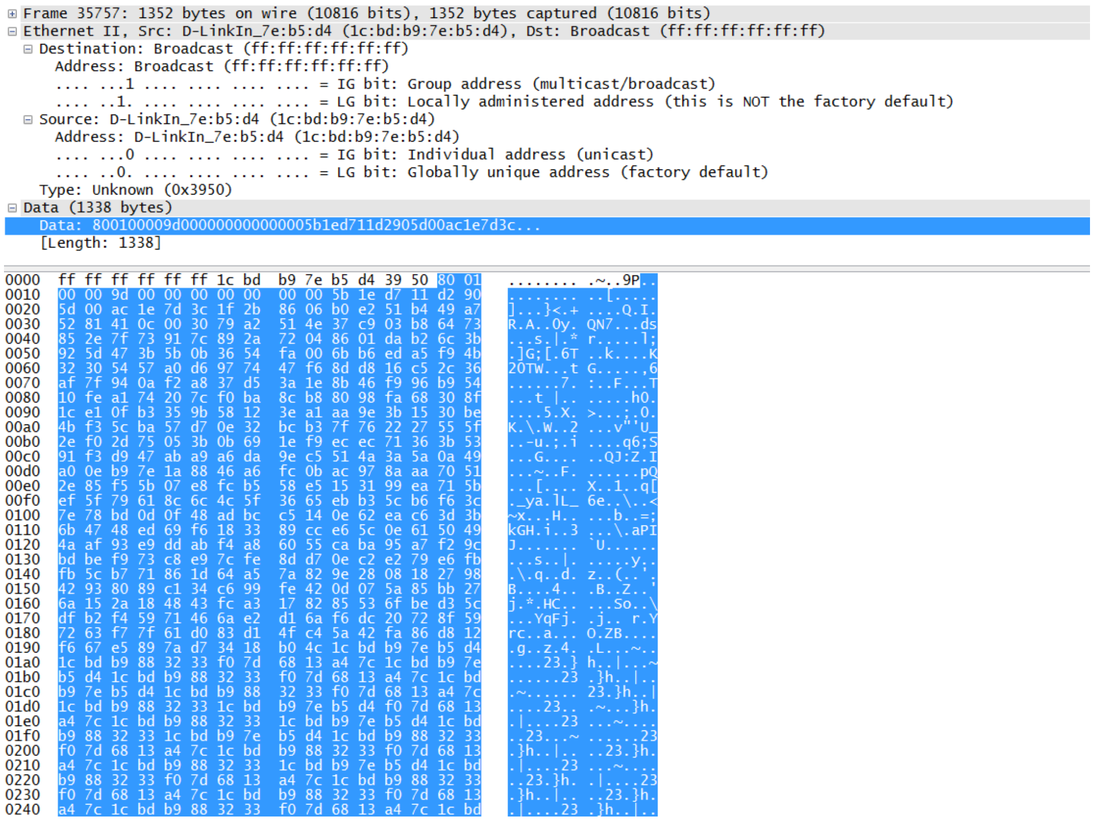
```
⊞ Frame 35757: 1352 bytes on wire (10816 bits), 1352 bytes captured (10816 bits)
⊟ Ethernet II, Src: D-LinkIn_7e:b5:d4 (1c:bd:b9:7e:b5:d4), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  ⊟ Destination: Broadcast (ff:ff:ff:ff:ff:ff)
      Address: Broadcast (ff:ff:ff:ff:ff:ff)
      .... ...1 .... .... .... .... = IG bit: Group address (multicast/broadcast)
      .... ..1. .... .... .... .... = LG bit: Locally administered address (this is NOT the factory default)
  ⊟ Source: D-LinkIn_7e:b5:d4 (1c:bd:b9:7e:b5:d4)
      Address: D-LinkIn_7e:b5:d4 (1c:bd:b9:7e:b5:d4)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
    Type: Unknown (0x3950)
⊟ Data (1338 bytes)
    Data: 800100009d000000000000005b1ed711d2905d00ac1e7d3c...
    [Length: 1338]

0000  ff ff ff ff ff ff 1c bd  b9 7e b5 d4 39 50 80 01   ........ .~..9P..
0010  00 00 9d 00 00 00 00 00  00 00 5b 1e d7 11 d2 90   ........ ..[.....
0020  5d 00 ac 1e 7d 3c 1f 2b  86 06 b0 e2 51 b4 49 a7   ]...}<.+ ....Q.I.
0030  52 81 41 0c 00 30 79 a2  51 4e 37 c9 03 b8 64 73   R.A..0y. QN7...ds
0040  85 2e 7f 73 91 7c 89 2a  72 04 86 01 da b2 6c 3b   ...s.|.* r.....l;
0050  92 5d 47 3b 5b 0b 36 54  fa 00 6b b6 ed a5 f9 4b   .]G;[.6T ..k....K
0060  32 30 54 57 a0 d6 97 74  47 f6 8d d8 16 c5 2c 36   20TW...t G.....,6
0070  af 7f 94 0a f2 a8 37 d5  3a 1e 8b 46 f9 96 b9 54   ......7. :..F...T
0080  10 fe a1 74 20 7c f0 ba  8c b8 80 98 fa 68 30 8f   ...t |.. .....h0.
0090  1c e1 0f b3 35 9b 58 12  3e a1 aa 9e 3b 15 30 be   ....5.X. >...;.0.
00a0  4b f3 5c ba 57 d7 0e 32  bc b3 7f 76 22 27 55 5f   K.\.W..2 ...v"'U_
00b0  2e f0 2d 75 05 3b 0b 69  1e f9 ec ec 71 36 3b 53   ..-u.;.i ....q6;S
00c0  91 f3 d9 47 ab a9 a6 da  9e c5 51 4a 3a 5a 0a 49   ...G.... ..QJ:Z.I
00d0  a0 0e b9 7e 1a 88 46 a6  fc 0b ac 97 8a aa 70 51   ...~..F. ......pQ
00e0  2e 85 f5 5b 07 e8 fc b5  58 e5 15 31 99 ea 71 5b   ...[.... X..1..q[
```

Since the message is broadcast by using NDP, every packet also displays the source address. The source address displays the Mac Address of the host/sender who was broadcasting the message.



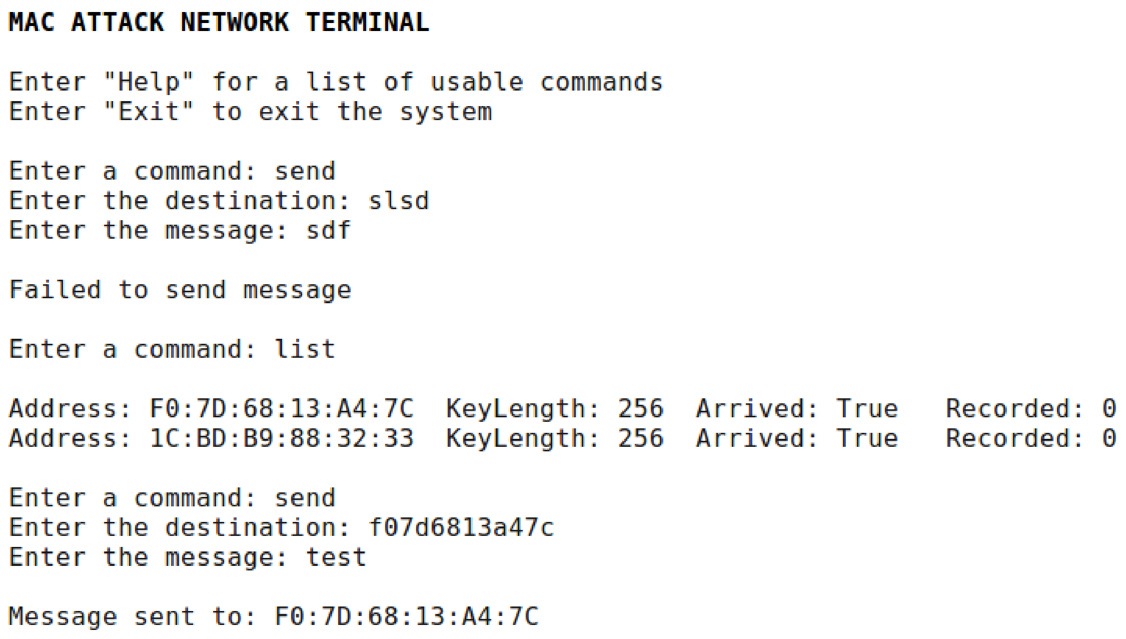
```
⊞ Frame 35757: 1352 bytes on wire (10816 bits), 1352 bytes captured (10816 bits)
⊟ Ethernet II, Src: D-LinkIn_7e:b5:d4 (1c:bd:b9:7e:b5:d4), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  ⊟ Destination: Broadcast (ff:ff:ff:ff:ff:ff)
      Address: Broadcast (ff:ff:ff:ff:ff:ff)
      .... ...1 .... .... .... .... = IG bit: Group address (multicast/broadcast)
      .... ..1. .... .... .... .... = LG bit: Locally administered address (this is NOT the factory default)
  ⊟ Source: D-LinkIn_7e:b5:d4 (1c:bd:b9:7e:b5:d4)
      Address: D-LinkIn_7e:b5:d4 (1c:bd:b9:7e:b5:d4)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
    Type: Unknown (0x3950)
⊟ Data (1338 bytes)
    Data: 800100009d000000000000005b1ed711d2905d00ac1e7d3c...
    [Length: 1338]

0000  ff ff ff ff ff ff 1c bd  b9 7e b5 d4 39 50 80 01   ........ .~..9P..
0010  00 00 9d 00 00 00 00 00  00 00 5b 1e d7 11 d2 90   ........ ..[.....
0020  5d 00 ac 1e 7d 3c 1f 2b  86 06 b0 e2 51 b4 49 a7   ]...}<.+ ....Q.I.
0030  52 81 41 0c 00 30 79 a2  51 4e 37 c9 03 b8 64 73   R.A..0y. QN7...ds
0040  85 2e 7f 73 91 7c 89 2a  72 04 86 01 da b2 6c 3b   ...s.|.* r.....l;
0050  92 5d 47 3b 5b 0b 36 54  fa 00 6b b6 ed a5 f9 4b   .]G;[.6T ..k....K
0060  32 30 54 57 a0 d6 97 74  47 f6 8d d8 16 c5 2c 36   20TW...t G.....,6
0070  af 7f 94 0a f2 a8 37 d5  3a 1e 8b 46 f9 96 b9 54   ......7. :..F...T
```

Lastly, the node also broadcasts its public key to its neighbours. It can be used by any node within the network to encrypt a message if it is part of the AddressPath.



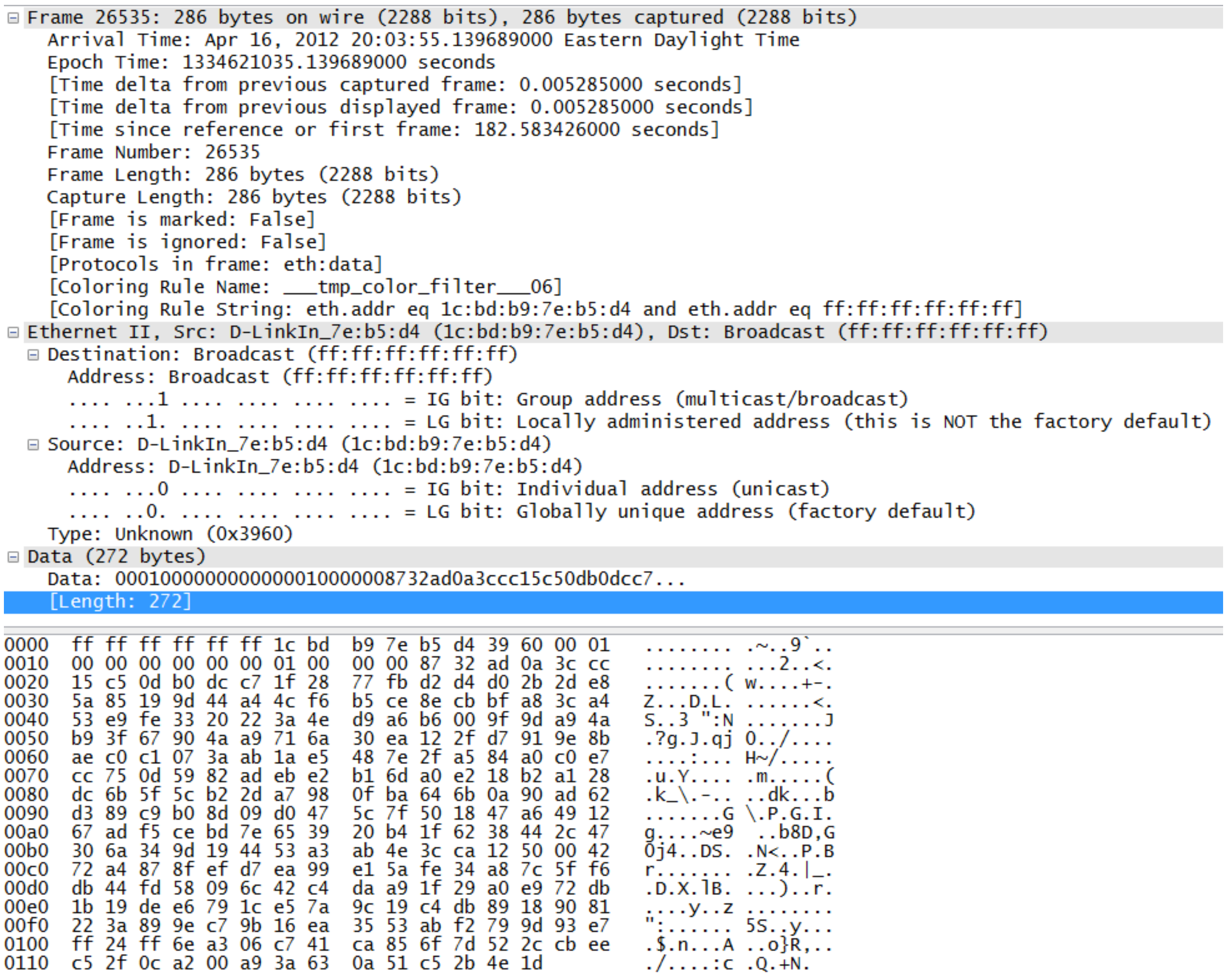
```
⊞ Frame 35757: 1352 bytes on wire (10816 bits), 1352 bytes captured (10816 bits)
⊟ Ethernet II, Src: D-LinkIn_7e:b5:d4 (1c:bd:b9:7e:b5:d4), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  ⊟ Destination: Broadcast (ff:ff:ff:ff:ff:ff)
      Address: Broadcast (ff:ff:ff:ff:ff:ff)
      .... ...1 .... .... .... .... = IG bit: Group address (multicast/broadcast)
      .... ..1. .... .... .... .... = LG bit: Locally administered address (this is NOT the factory default)
  ⊟ Source: D-LinkIn_7e:b5:d4 (1c:bd:b9:7e:b5:d4)
      Address: D-LinkIn_7e:b5:d4 (1c:bd:b9:7e:b5:d4)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
    Type: Unknown (0x3950)
⊟ Data (1338 bytes)
    Data: 800100009d000000000000005b1ed711d2905d00ac1e7d3c...
    [Length: 1338]

0000  ff ff ff ff ff ff 1c bd  b9 7e b5 d4 39 50 80 01   ........ .~..9P..
0010  00 00 9d 00 00 00 00 00  00 00 5b 1e d7 11 d2 90   ........ ..[.....
0020  5d 00 ac 1e 7d 3c 1f 2b  86 06 b0 e2 51 b4 49 a7   ]...}<.+ ....Q.I.
0030  52 81 41 0c 00 30 79 a2  51 4e 37 c9 03 b8 64 73   R.A..0y. QN7...ds
0040  85 2e 7f 73 91 7c 89 2a  72 04 86 01 da b2 6c 3b   ...s.|.* r.....l;
0050  92 5d 47 3b 5b 0b 36 54  fa 00 6b b6 ed a5 f9 4b   .]G;[.6T ..k....K
0060  32 30 54 57 a0 d6 97 74  47 f6 8d d8 16 c5 2c 36   20TW...t G.....,6
0070  af 7f 94 0a f2 a8 37 d5  3a 1e 8b 46 f9 96 b9 54   ......7. :..F...T
0080  10 fe a1 74 20 7c f0 ba  8c b8 80 98 fa 68 30 8f   ...t |.. .....h0.
0090  1c e1 0f b3 35 9b 58 12  3e a1 aa 9e 3b 15 30 be   ....5.X. >...;.0.
00a0  4b f3 5c ba 57 d7 0e 32  bc b3 7f 76 22 27 55 5f   K.\.W..2 ...v"'U_
00b0  2e f0 2d 75 05 3b 0b 69  1e f9 ec ec 71 36 3b 53   ..-u.;.i ....q6;S
00c0  91 f3 d9 47 ab a9 a6 da  9e c5 51 4a 3a 5a 0a 49   ...G.... ..QJ:Z.I
00d0  a0 0e b9 7e 1a 88 46 a6  fc 0b ac 97 8a aa 70 51   ...~..F. ......pQ
00e0  2e 85 f5 5b 07 e8 fc b5  58 e5 15 31 99 ea 71 5b   ...[.... X..1..q[
00f0  ef 5f 79 61 8c 6c 4c 5f  36 65 eb b3 5c b6 f6 3c   ._ya.lL_ 6e..\..<
0100  7e 78 bd 0d 0f 48 ad bc  c5 14 0e 62 ea c6 3d 3b   ~x...H.. ...b..=;
0110  6b 47 48 ed 69 f6 18 33  89 cc e6 5c 0e 61 50 49   kGH.i..3 ...\.aPI
0120  4a af 93 e9 dd ab f4 a8  60 55 ca ba 95 a7 f2 9c   J....... `U......
0130  bd be f9 73 c8 e9 7c fe  8d d7 0e c2 e2 79 e6 fb   ...s..|. .....y..
0140  fb 5c b7 71 86 1d 64 a5  7a 82 9e 28 08 18 27 98   .\.q..d. z..(..'.
0150  42 93 80 89 c1 34 c6 99  fe 42 0d 07 5a 85 bb 27   B....4.. .B..Z..'
0160  6a 15 2a 18 48 43 fc a3  17 82 85 53 6f be d3 5c   j.*.HC.. ...So..\
0170  df b2 f4 59 71 46 6a e2  d1 6a f6 dc 20 72 8f 59   ...YqFj. .j.. r.Y
0180  72 63 f7 7f 61 d0 83 d1  4f c4 5a 42 fa 86 d8 12   rc..a... O.ZB....
0190  f6 67 e5 89 7a d7 34 18  b0 4c 1c bd b9 7e b5 d4   .g..z.4. .L...~..
01a0  1c bd b9 88 32 33 f0 7d  68 13 a4 7c 1c bd b9 7e   ....23.} h..|...~
01b0  b5 d4 1c bd b9 88 32 33  f0 7d 68 13 a4 7c 1c bd   ......23 .}h..|..
01c0  b9 7e b5 d4 1c bd b9 88  32 33 f0 7d 68 13 a4 7c   .~...... 23.}h..|
01d0  1c bd b9 88 32 33 1c bd  b9 7e b5 d4 f0 7d 68 13   ....23.. .~...}h.
01e0  a4 7c 1c bd b9 88 32 33  1c bd b9 7e b5 d4 1c bd   .|....23 ...~....
01f0  b9 88 32 33 1c bd b9 7e  b5 d4 1c bd b9 88 32 33   ..23...~ ......23
0200  f0 7d 68 13 a4 7c 1c bd  b9 88 32 33 f0 7d 68 13   .}h..|.. ..23.}h.
0210  a4 7c 1c bd b9 88 32 33  1c bd b9 7e b5 d4 1c bd   .|....23 ...~....
0220  b9 88 32 33 f0 7d 68 13  a4 7c 1c bd b9 88 32 33   ..23.}h. .|....23
0230  f0 7d 68 13 a4 7c 1c bd  b9 88 32 33 f0 7d 68 13   .}h..|.. ..23.}h.
0240  a4 7c 1c bd b9 88 32 33  f0 7d 68 13 a4 7c 1c bd   .|....23 .}h..|..
```

Meanwhile, a user attempts to send a message to a known destination address. Upon successful completion of task, the application lists the address to which the message was delivered to with a message indicating successful or unsuccessful transmission. Correspondingly this information is only available to the sender of the message and hidden from the view of an external observer.



```
MAC ATTACK NETWORK TERMINAL

Enter "Help" for a list of usable commands
Enter "Exit" to exit the system

Enter a command: send
Enter the destination: slsd
Enter the message: sdf

Failed to send message

Enter a command: list

Address: F0:7D:68:13:A4:7C  KeyLength: 256  Arrived: True   Recorded: 0
Address: 1C:BD:B9:88:32:33  KeyLength: 256  Arrived: True   Recorded: 0

Enter a command: send
Enter the destination: f07d6813a47c
Enter the message: test

Message sent to: F0:7D:68:13:A4:7C
```

Once a message has been transmitted, an external observer can attempt to analyse the transmitted information by inspecting the packet below. Though it is pretty clear that the destination address is obscured to the external observer, even though our application tells us otherwise; since the message is also encrypted, an eavesdropper cannot perform any man in the middle attacks to steal any crucial piece of information to gain access to the network. Essentially, this means that both the message and the destination have been made anonymous.



```
⊟ Frame 26535: 286 bytes on wire (2288 bits), 286 bytes captured (2288 bits)
    Arrival Time: Apr 16, 2012 20:03:55.139689000 Eastern Daylight Time
    Epoch Time: 1334621035.139689000 seconds
    [Time delta from previous captured frame: 0.005285000 seconds]
    [Time delta from previous displayed frame: 0.005285000 seconds]
    [Time since reference or first frame: 182.583426000 seconds]
    Frame Number: 26535
    Frame Length: 286 bytes (2288 bits)
    Capture Length: 286 bytes (2288 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:data]
    [Coloring Rule Name: ___tmp_color_filter___06]
    [Coloring Rule String: eth.addr eq 1c:bd:b9:7e:b5:d4 and eth.addr eq ff:ff:ff:ff:ff:ff]
⊟ Ethernet II, Src: D-LinkIn_7e:b5:d4 (1c:bd:b9:7e:b5:d4), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  ⊟ Destination: Broadcast (ff:ff:ff:ff:ff:ff)
      Address: Broadcast (ff:ff:ff:ff:ff:ff)
      .... ...1 .... .... .... .... = IG bit: Group address (multicast/broadcast)
      .... ..1. .... .... .... .... = LG bit: Locally administered address (this is NOT the factory default)
  ⊟ Source: D-LinkIn_7e:b5:d4 (1c:bd:b9:7e:b5:d4)
      Address: D-LinkIn_7e:b5:d4 (1c:bd:b9:7e:b5:d4)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
    Type: Unknown (0x3960)
⊟ Data (272 bytes)
    Data: 000100000000000001000000873 2ad0a3ccc15c50db0dcc7...
    [Length: 272]

0000  ff ff ff ff ff ff 1c bd  b9 7e b5 d4 39 60 00 01   ........ .~..9`..
0010  00 00 00 00 00 00 01 00  00 00 87 32 ad 0a 3c cc   ........ ...2..<.
0020  15 c5 0d b0 dc c7 1f 28  77 fb d2 d4 d0 2b 2d e8   .......( w....+-.
0030  5a 85 19 9d 44 a4 4c f6  b5 ce 8e cb bf a8 3c a4   Z...D.L. ......<.
0040  53 e9 fe 33 20 22 3a 4e  d9 a6 b6 00 9f 9d a9 4a   S..3 ":N .......J
0050  b9 3f 67 90 4a a9 71 6a  30 ea 12 2f d7 91 9e 8b   .?g.J.qj 0../....
0060  ae c0 c1 07 3a ab 1a e5  48 7e 2f a5 84 a0 c0 e7   ....:... H~/.....
0070  cc 75 0d 59 82 ad eb e2  b1 6d a0 e2 18 b2 a1 28   .u.Y.... .m.....(
0080  dc 6b 5f 5c b2 2d a7 98  0f ba 64 6b 0a 90 ad 62   .k_\.-.. ..dk...b
0090  d3 89 c9 b0 8d 09 d0 47  5c 7f 50 18 47 a6 49 12   .......G \.P.G.I.
00a0  67 ad f5 ce bd 7e 65 39  20 b4 1f 62 38 44 2c 47   g....~e9  ..b8D,G
00b0  30 6a 34 9d 19 44 53 a3  ab 4e 3c ca 12 50 00 42   0j4..DS. .N<..P.B
00c0  72 a4 87 8f ef d7 ea 99  e1 5a fe 34 a8 7c 5f f6   r....... .Z.4.|_.
00d0  db 44 fd 58 09 6c 42 c4  da a9 1f 29 a0 e9 72 db   .D.X.lB. ...)..r.
00e0  1b 19 de e6 79 1c e5 7a  9c 19 c4 db 89 18 90 81   ....y..z ........
00f0  22 3a 89 9e c7 9b 16 ea  35 53 ab f2 79 9d 93 e7   ":...... 5S..y...
0100  ff 24 ff 6e a3 06 c7 41  ca 85 6f 7d 52 2c cb ee   .$.n...A ..o}R,..
0110  c5 2f 0c a2 00 a9 3a 63  0a 51 c5 2b 4e 1d         ./....:c .Q.+N.
```

CONCLUSION

From our simulations and demonstrations, our routing protocol was able to mask the contents of a transmitted packet, conceal the identity of the users (senders and users) while ensuring that the messages were only sent via authorized nodes. However, the application was plagued by a number of issues during the implementation phase. The complexity of implementing such a protocol is extremely high due to the multiple layers of encryption involved. This proved to be an additional challenge to our routing strategy as RSA algorithms are time consuming and hence, having an impact on the reliability of the network while trying to pick a route.

The Tor network solves this problem by deploying dedicated Tor Routers/Nodes around the world which act as relay nodes and help the Tor client pick a random path to the destination address. Since the former approach is more feasible for an infrastructure based network, such an approach is not feasible for a mobile ad hoc network due to the dynamic nature and fluctuating topology of the network infrastructure.

Onion Routing (in general), however, is not invulnerable to such attacks. If an attacker has enough data and has the right set of skills and an advanced understanding of statistical analysis techniques, it is possible to analyze usage patterns and hence, make an educated guess about the routing of messages. Likewise, it is definitely possible to perform "exit node sniffing" attacks at either end of the nodes (sender and/or receiver). This form of attack is extremely dangerous since an exit node has complete access to the content being transmitted between the sender and the receiver. Generally, this problem can be made much harder by employing end-to-end encryption.

FUTURE DEVELOPMENT

Since our application was plagued with performance issues, we have devised a few strategies to further improve the performance of our routing. One was to make use of HashMaps and the other idea was to use AES (symmetric-key algorithms) for encrypting messages.

Hash functions are extremely fast and easy to compute, while also making it infeasible to modify a message. In addition to this, no two messages will ever have the same hash value (ideally) which is a useful property given the nature of this application. Additionally, AES algorithm is a well-known specification for encrypting data and one of the most popular algorithms employed in symmetric cryptography. Since symmetric-key algorithms are used for both the encryption and decryption process, it would be well worth exploring the use of AES encryption and hash maps and compare the performance between the two systems.

Furthermore, since all known attacks against AES encryption are currently computationally infeasible, it makes it an attractive option to consider if one decides to implement an anonymous routing protocol.

REFERENCES

[1] Tor project: Overview. (2012, February 26). Retrieved from: www.torproject.org/about/overview.html.en

[2] "Tor: The Second-Generation Onion Router", in Proceedings of the 13th USENIX Security Symposium, August 2004.

[3] David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. Hiding Routing Information, Workshop on Information Hiding, Cambridge, UK, May, 1996.

[4] Yih-Chun Hu, Adrian Perrig, "A Survey of Secure Wireless Ad Hoc Routing," IEEE Security and Privacy, pp. 28-39, May-June, 2004

[5] Barbeau, M. (2010, January 11). The zone routing protocol and neighbour discovery protocol. Retrieved from: people.scs.carleton.ca/~claurend/Courses/COMP4203/W12/Resources/NDP.pdf

[6] Z.J. Haas and M.R. Pearlman, ZRP a hybrid framework for routing in ad hoc networks, Ad Hoc Networking (Charles E. Perkins, ed.), Addison-Wesley, 2001, pp. 221-253.

[7] Laurendeau, C. (2006, April). Wireless networks and protocols: Lecture review. Retrieved from: people.scs.carleton.ca/~claurend/Courses/COMP4203/W12/Resources/ChristineLectRev.pdf

[8] Dingledine, R., Mathewson, N., & Syverson, P. (2004, August). Tor: The second-generation onion router. Paper presented at Proceedings of the 13th USENIX Security Symposium, San Diego.

[9] Neal, Harrison (Wikimedia, March 2008) SVG Diagram of the Onion Routing Principal, Online [World Wide Web] Available From: en.wikipedia.org/wiki/File:Onion_diagram.svg