Software implementation details of the secure data communication protocols stack based on the dynamic network topology

E A Kushko and N Y Parotkin
Institute of Informatics and Telecommunications, Reshetnev Siberian State University of Science and Technology, Krasnoyarsk, Russia

E-mail: evgeny.kushko@gmail.com

Abstract. Secure data communication protocols stack based on the dynamic network topology is a set of protocols designed for secure data communication over a network. It is based on the principles of moving target technology along with cryptography, avalanche routing and group broadcasting. The data transfer protocol implemented in the protocols stack allows to transfer data not only hiding the data itself, but also its sender and receiver. The paper covers the software implementation details of the secure data communication protocols stack.

1. Introduction
According to the information security incidents statistics despite the fact that significant funds are invested in the development of information systems protection, attackers successfully attack these systems. The shortcomings and imperfections of the protocols and security measures used make it possible to intercept and analyze network traffic transmitted over the network. As a result, an attacker can extract confidential information, as well as find new attack vectors. Due to the fact that the analysis and interception of traffic are difficult to detect [1], in the case of overcoming the network perimeter, the attacker has almost unlimited amount of time. There are several approaches to counteract the interception and analysis of network traffic, but they have some disadvantages, such as identification of the type of traffic [2], identification of users [3], proven attacks [4] and others.

To address the shortcomings of existing solutions to counteract interception and traffic analysis, we developed our own solution - a secure data communication protocols stack based on the dynamic network topology. In this work, we’ve sought to achieve the following protocols stack characteristics:

- data hiding;
- hiding communication channels between nodes;
- inability to use existing attack methods and vulnerabilities;
- compliance with the general security requirements;
- flexible integration with existing hardware and software infrastructure;
- ease and low cost of use;
- decentralized system configuration;
- self-recovery;
- self-adaptation;
guaranteed bandwidth and communication quality.

2. Functional description of the protocols stack
We’ve considered algorithms underlying the stack of secure data communication protocols as well as comparison with similar technologies in another work [5]. A secure data communication protocols stack based on the dynamic network topology implements the principles of moving target technology [6]. Network nodes participating in the secure data communication, move through the groups of secure data communication and avalanche transmit encrypted data to these groups by means of group broadcasting. The data packet transmitted this way does not contain the explicit addresses of the receiving and sending nodes, so it is impossible to establish specific links between the nodes. In addition, nodes are connected to multiple secure communication groups at the same time, and the data packet has multiple delivery paths to the receiver. Avalanche routing and handshake method ensure guaranteed delivery of data to the receiver. Each node relays data packets resulting in a communication channel between the two nodes are hidden among identical channels.

In the first step, a node joins the secure data communication initialization group, forms the list of active participants and selects 2-3 groups from the pool of all available groups and periodically changes them according to a special algorithm, the pool is also dynamic, decentralized and formed by the algorithm. In the current implementation, the algorithm is a function of the time and number of the secure data communication participants.

The protocols stack provides two types of message: service and application. Service messages are responsible for transmission of service information between nodes, for example, for notification of connection to a secure network segment, for exchange of identifiers and public keys. The protocols stack provides several operation codes, such as: connect, disconnect, binary data, text, error, and successful data processing. The node generates a data packet and sends it to the connected secure data communication groups.

The application data packet is formed as follows. The packet header contains:

- packet ID in the open (2 bytes);
- type of message in the open (1 byte);
- operation code encrypted with the recipient's public key (1 byte);
- recipient ID encrypted with the recipient's public key (2 bytes);
- sender ID encrypted with the recipient's public key (2 bytes);
- timestamp encrypted with the recipient's public key (4 bytes);
- message length, encrypted with the recipient's public key (2 bytes);
- reserved field;
- applied information encryption key, encrypted with the recipient's public key;
- applied information encrypted with the application information encryption key.

The service message contains the same set of data, except for the application information encryption key and the application information block, and it is transmitted in open form. The encryption algorithm of the packet headers is RSA with a key length of 1024 bits and the application information encryption algorithm if AES with a key length of 256 bits.

All participants of the secure data communication, connected to the groups where the packet was sent, relay the received message and try to decrypt the packet header. If the participant of the secure data communication decrypts the packet successfully and finds its ID in the header, then he considers that the packet is intended for him. Otherwise, the node discards the packet. The node in any case relays the packet, regardless of whether the packet is intended for it or not, if the node has not previously processed the packet. The node that has successfully received the packet generates a successful delivery response. If the sending node does not receive a successful response after the time has elapsed, it sends the data packet again.
3. Protocols stack software implementation details

The authors have developed a software implementation of the secure data communication protocols stack based on the dynamic network topology. This software implementation is a plug-in library that can be used both in a user project to implement secure data communication and as a standalone application with a graphical interface.

Developing the application, the authors have used the Qt 5.10 framework for C++ development, as well as the open source cryptographic library OpenSSL version 1.1.0h. Qt allows to create cross-platform and portable software, and provides a set of system-independent libraries to work with different types of data, with the network, with streams and many other libraries. The OpenSSL library supports most encryption standards and is available on most UNIX-like operating systems as well as on Windows operating systems. The authors successfully compiled the application and tested it under the x86-64 architecture running Windows 10 and Ubuntu 14.04, as well as under the ARM architecture running the Raspbian operating system.

The program is written in an object-oriented programming paradigm using the Model-View-Controller design pattern. The model, in this case, is the data of the application, the view is a kind of display of the model to the user, and the controller provides an interface for interaction with the software solution. In this work, we sought to reduce the engagement and increase the connectivity of the software solution so that the code was well structured and could be reused. Figure 1 shows the architecture of the software solution.

![Software solution architecture](image)

**Figure 1.** Software solution architecture.

The software solution contains the following main classes:

- MtdAdapter
- MtdAdapterSettings
- MtdInformation
- MtdModel
- MtdService
- MtdProtocol
- MtdProtocolMessage
The code implements the described above functional description. MtdAdapter is a class responsible for implementing the transport part of the protocol. It contains a UDP socket and mechanisms for interacting with it. Depending on the mode of operation, 2 or 3 objects of this class are created. MtdAdapterSettings includes parameters to control the transport part of the secure communication protocols stack, such as the local port, IP address of the secure communication group, and other parameters.

MtdProtocol is responsible for processing incoming and outgoing data packets. This class uses the interface of the MtdCryptography class to generate keys, encrypt outgoing and incoming data packets, decrypt them, and pass them to the MtdService and MtdInformation classes for further processing. MtdProtocolMessage converts an array of bytes into a data packet representation of the secure communication protocols stack. MtdService and MtdInformation are classes responsible for generating and processing service and information messages, respectively. MtdCryptography is a class that is responsible for the cryptographic functions of the secure data communication protocols stack based on the dynamic network topology.

MtdSync is responsible for synchronizing the change of multicast groups of secure data communication, as well as for calculating the dynamic parameters of network adapters. MtdSettings includes parameters for managing the secure communication protocols stack as a whole, such as encryption key lengths, secure communication group change interval, number of network adapters, allowing you to study the most efficient protocols stack parameters. Mtd is a controller class that provides the application programming interface as well as creating all the necessary entities. In addition, the program code contains some helper classes, such as graphical user interface classes and others.

The program is multithreaded, but data packets are processed in single-threaded mode. The following entities work in separate threads: MtdProtocol, two or three MtdAdapter, and MtdSync. The rest of the entities work in the main application thread. The interaction between the threads is carried out by the built-in Qt slots and signals [7].

The program initially connects to a secure network segment, generates a list of secure communication participants and a list of their public encryption keys by an object of the MtdService class, passing objects of the MtdProtocolMessage class to the MtdProtocol object, which forms an array of bytes for transmission to the MtdAdapter network adapter connected to the secure data communication initialization group. The next step is for the MtdSync class object to configure the network adapter settings and reconfigure them at the time intervals specified by the MtdSettings class. After that, the user can exchange data through the graphical interface or through the software interface provided by the Mtd class.

4. Data transfer performance
This software solution has been used to determine data transfer performance. Table 1 shows the transfer overhead costs when using the secure data communication protocols stack based on the dynamic network topology.

| Message type | Operation type                        | Header size | Total size | Transfer costs |
|--------------|---------------------------------------|-------------|------------|----------------|
| Service      | Connection to a secure network segment| 1136 bits   | 1136 bits  | 100%           |
| Service      | Responding to a request to connect to a| 1136 bits   | 1136 bits  | 100%           |

Table 1. Transfer costs.
According to RFC 2544 [8], the following data transmission performance parameters were obtained: bandwidth (table 2), latency (table 3), frame loss, and data packet processing with a minimum interval. There was no frame loss at all frame sizes. Data packet processing with a minimum interval is 5 MS for all frame sizes. The authors built an experimental stand in the form of a peer-to-peer network based on the D-Link DSR-1000AC router with a variable number of intermediate nodes between the sender and the receiver of data.

Table 2. Bandwidth.

| Frame size (bits) | Bandwidth (kbit/s) |
|------------------|---------------------|
|                  | Number of repeaters between nodes |
|                  | No repeaters | One repeater | Two repeaters |
| 64               | 39          | 22          | 9            |
| 128              | 83          | 41          | 18           |
| 256              | 155         | 76          | 36           |
| 512              | 308         | 162         | 84           |
| 1024             | 590         | 252         | 118          |
| 1518             | 654         | 314         | 153          |
| 2048             | 1170        | 622         | 315          |

Table 3. Latency.

| Frame size (bits) | Latency (ms) |
|------------------|--------------|
|                  | Number of repeaters between nodes |
|                  | No repeaters | One repeater | Two repeaters |
| 64               | 10          | 21          | 40           |
| 128              | 11          | 21          | 42           |
| 256              | 11          | 22          | 42           |
| 512              | 12          | 25          | 43           |
| 1024             | 13          | 27          | 44           |
| 1518             | 13          | 27          | 45           |
| 2048             | 14          | 29          | 46           |

5. Conclusion

As a result of the software solution implementation, it was possible to determine the performance parameters of data transmission over the network. Unfortunately, the current performance parameters do not allow to transfer a large amount of data. The system has several bottlenecks: single-threaded processing of data packets, configuration of inefficient network parameters and inefficient pool of protected data communication groups, generation of too much side traffic. The developed software solution allows to solve problems of the secure data communication protocols stack more effectively.

References

[1] Qadeer M A, Zahid M, Iqbal A and Siddiqui M R 2010 Network Traffic Analysis and Intrusion Detection Using Packet Sniffer ICCSN '10 2nd Int. Conf. (Singapoure: IEEE) pp 313–7

[2] Alshammari R and Nur Zincir-Heywood A 2009 Machine Learning based encrypted traffic classification: identifying SSH and Skype In Proc. of the IEEE Symp. on Computational
Intelligence for Security and Defense Applications (Ottawa: IEEE) pp 1–8

[3] Yanbing M and Xiaolei X 2017 Locating tor's hidden service clients based on protocol feature IEEE 2nd Information Technology Networking Electronic and Automation Control Conf. pp 282–5

[4] Ren J and Jie W 2010 Survey on anonymous communications in computer networks Computer Communications 33 420–31

[5] Kushko E A and Parotkin N Y 2019 The research of technologies for secure data communication in dynamic networks Proc. of the Int. Conf. Dynamics of Systems, Mechanisms and Machines (Omsk: IEEE)

[6] Styugin M A, Parotkin N Y and Zolotarev V V 2018 An Approach for Constructing Indistinguishable Information Systems Proc. of the Int. Conf. Quality Management, Transport and Information Security, Information Technologies (St. Petersburg: Russia)

[7] Lazar G and Penea R 2016 Mastering Qt 5 (Birmingham: Packt Publishing)

[8] Bradner S and McQuaid J 1999 Benchmarking Methodology for Network Interconnect Devices Request for Comments 2544