Constructing Private Service with CRYP2CHAT Application

Andrey Kiryantsev <reyzor2142@gmail.com>,
Irina Stefanova, <aistvt@mail.ru>,
Volga Region State University of Telecommunications and Informatics, 77 Moskovskoe sh., Samara, 443090, Russian Federation

Annotation. The article contains the description of a private service with the client-side data encryption and data decryption. Owing to the Onion Router (TOR) technology, anonymous network connection protected from interception becomes possible. Users in TOR network may remain anonymous while visiting websites, uploading materials, sending messages and working with other applications that use TCP protocol. Traffic security is ensured by the distributed network of onion routers. The focus of the article is on the direct client-to-client connection. Nowadays messengers – programs for on-line messages exchange – place metadata on the central server without encryption, which provides an opportunity to learn (if required) the information about the common users, time of their communication, the number of messages they send within a session. To solve the problem the authors offer CRYP2CHAT program for client-side encryption. Sending messages through TOR network is performed by asymmetric encryption, e.g. by RSA method that enables other encryption algorithms as well. The article provides the algorithm for work of the programs. The authors describe the methods of protection from some network attacks, such as MITM and the experiment of prototype work. They check clean access server and use self-destruction of messages after the session end. Additionally, the authors consider some potential dangers of an external character that can violate confidential communication data, for instance, change of the application code, password attack or private key theft. The article illustrates the way the Onion Router technology works. It allows to protect from MITM attacks, to remain anonymous and to proxy. Moreover, there is a comparative analysis of Cryp2Chat qualitative characteristics and its analog.

***Keywords*:** cryptography; encryption; encoding; MITM-attack; end2end encryption; node.js; cryprico; java script

DOI: 10.15514/ISPRAS-2015-27(3)-19

For citation: Kiryantsev Andrey, Stefanova Irina. Constructing Private Service with CRYP2CHAT Application. *Trudy ISP RAN/Proc. ISP RAS, vol. 27, issue 3, 2015*, pp. 279-290. DOI: 10.15514/ISPRAS-2015-27(3)-19.

1. Introduction

Modern society is characterized by the exchange and buffering information in electronic form. While processing the information, we may need to react immediately on constantly emerging problems with data protection and security of data centers.

The problem is now becoming more urgent considering declarations and current publications by Edward Snowden, the former system administrator for the Central Intelligence Agency. He reports on the fact that the National Security Agency (NSA) operates global surveillance programs with the cooperation of telecommunication companies and European governments through the existing communication networks.

Nowadays to exchange information on-line special programs – messengers – are used. They are particularly useful for transmission of text messages, sound signals, images, video and games as well as for organization of teleconferences by coding messages of on-line users. Messengers usually operate in coordination with a server, and they are defined as client-side programs with their own rules of work and peculiarities in operating, e.g. ICQ, Skype. The main drawback of these programs is that while using them, we leave metadata on a hosting server as non-encrypted data flow, which provides an opportunity to learn (if required) the information about the common users, time of their communication, the number of messages they send within a session.

2. Description of CRYP2CHAT program

To eliminate the defect we develop a model to run a program that allows coding the data on the client side with the help of Cryp2Chat Application

Currently existing Internet messengers fail to perform the following functions:

- to check for MITM (Men in the middle)-attacks;
- to provide a ‘clean’ (data free) server;
- to destruct messages automatically after the session is over.

MITM-attack is the most wide-spread way to attack for stealing the data of some users. This type of attack presupposes that the attackers are able to read and alter messages of a sender and a receiver as they wish. Additionally, neither a sender nor a receiver sees any hints of the attacker to be in the channel. It is the matter of no importance if SSL cryptographic protocol is applied or not. The attacker hooks into a channel between users and interferes actively with the communication protocol. He/ she may delete, falsify data or provide the false ones.

The term ‘clean’ server implies that the communication between two users leaves no information on the server. In this case the server functions as a repeater and simply translates the encrypted message between the clients. After the session is over, the access to the data of the on-line chart is lost without any opportunity for return.

The described problems with messengers could be solved if we use a new application – Cryp2Chat.

Cryp2Chat application has been developed to minimize the drawbacks of the Internet messengers, i.e. it leaves no metadata on the central server. The client is the only person who can decode the incoming message. The client possesses data de-encryption key, and the key does not go further.

The program operation procedure is the following (fig. 1). A server receives a list of network user's contacts. A data encryption key is generated on the side of a sender. Further the public key is sent to the server and, finally, to a receiver. The private part of a key remains on the user's (sender's) side.

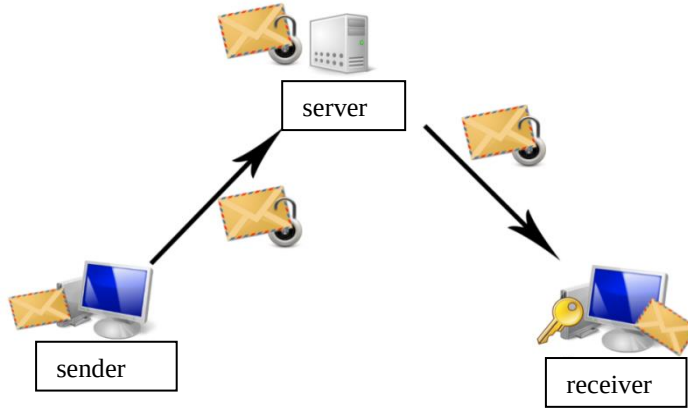

Fig. 1. An Example of Cryp2Chat Application Running

When a user (a receiver) sends back a message, the operation is realized within three main stages:

1. He/she receives a public key of a receiver from the server;
2. The message is encrypted by a public key;
3. The cryptographed message is sent to the server.

RSA method is employed for encryption; the key length includes 1024 bit. However, the possibility to use other algorithms of encryption is also provided.

The server created as a prototype of this application is written in Node.js programming language (advanced JavaScript) on the basis of Socket.IO library.

Cryp2Chat application is an original service designed to exchange rapidly-changing messages. It supports End2End encryption.

To enable the program to use proxy servers (to protect the client's computer from some network attacks) and to increase the reliability of a channel, we offer the use of network of TOR (The Onion Router). On the computer of a client a proxy server connected to the network of TOR starts its work [1]. It involves a multilevel encryption. The process of message transmission in a network is schematically presented on fig. 2.

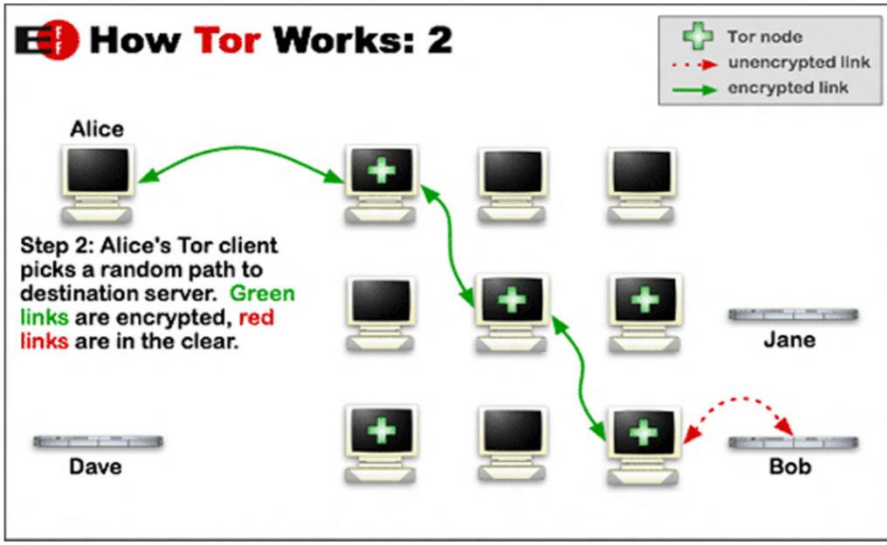

Fig.2. Schematic Presentation of TOR Work

Before transmitting the data packet to the server, it goes through three random computers. Before being sent, the package is encrypted by three keys: for each of the three computers respectively. In addition, the TOR network can provide anonymity for servers.

Network users start TOR multi-level (“onion”) proxy server on their machine. It connects to the TOR servers, periodically forming a chain through the TOR network that uses a multi-level encryption. Every packet entering the system passes through three different proxy servers - server *nodes* that are randomly selected. Before being sent, the package is sequentially encrypted by three keys: first, in the third node, then in the second node, and, finally, in the first node. When the first node receives a packet, it decrypts the "upper" layer encryption (similar to how we clean the onion) and gets the information where to send the packet to. The second and the third servers do the same. At the same time, the software multi-level ("onion") proxy server provides a SOCKS-interface.

SOCKS (SOCKet Secure) are the programs, running on the SOCKS-based interface. Their work could be configured through the TOR network. The TOR network creates multiplexed traffic and sends data through a virtual chain of the TOR network, thus, providing anonymous web surfing.

Inside the TOR network the traffic is forwarded from one router to another, and finally it reaches the exit point from which the pure (unencrypted) data package comes to the original recipient address (server). The traffic from the receiver is sent back to the exit point of the TOR network [2].

The server prototype of this application is written in Node.js (advanced JavaScript) with the help of the library for web sockets - Socket.IO.

Node.js is a programming platform founded on V8 database engine that translates JavaScript into the machine code. In this way it transforms JavaScript from the higly-specialised language into the common language for users. The client part is realized on Html and JavaScript with the help of Cryprico library.

Node.js has not been chosen by chance. This is one of the few servers that work quickly and productively with a single-threaded code. For instance, being the programming language it does not need to create a new thread to transmit a stream of query parameters and to interpret the code.

Node.js is the aggregate of the V8 database engine used in Google Chrome and in the abstraction to access the file system and similar server modules.

To shift away from the standard web 2.0 scheme of data transmission we used Web-Sockets and their implementation for node.js servers in the form of Socket.IO library. It should be mentioned that Web-Socket is a Protocol intended for exchanging messages between the browser and the web server in real time.

At the same time, the Socket.IO library provides a good level of abstraction above the sockets that are implemented in JavaScript. With its help you can easily pass objects to the server and from the server, without serializing them.

The structure of the server part is the following: the server accepts the message. If it is a command, the server performs certain actions. If it is simply a message, the server sends it to the client.

The JavaScript language, which is used in the prototype, is currently the most common cross-platform language. It is commonly used as an embedded language for program access to the application objects. The JavaScript language is widely used in browsers as a scripting language to add interactivity to the web-pages.

The JavaScript language may be distinguished by its main architectural features: dynamic typing, weak typing, automatic memory management, prototype programming, and functions as the first class objects.

The only requirement for JavaScript work (and it is present by default in all operating systems) is the availability of the browser. It does not need to be rewritten when migrating from one operating system to another. We write the script and run it in the place where there is a browser on an electronic device.

Over the last decade JavaScript turned from the applied language for checking how the blanks are filled, into a language that can provide the programmer a powerful tool to tackle any kind of problems. The JavaScript library is constantly updated with new scripts and styles.

Now there are many add-in settings for JavaScript as its possibilities are constantly growing, but the syntax and its architecture is not changed. A simple example is CoffeeScirpt language, which allows you to write more compact code compared to JavaScript. It helps to solve some architectural omissions such as the lack of OOP (object oriented programming), collbecki (CallBacks) – callback and syntactic ‘sugar’ (code lines that improve the way the program looks like). All this makes the language more convenient for the programmer.

3. Prototype work

As an example, we may consider the fragments of scripts in Cryp2Chat prototype. Below there is a fragment of the script that implements the simultaneous exchange of encryption keys between clients:

```
socket.on('key1',function(data)
{
        keys[0] = data.key;
}
);
socket.on('key2',function(data){
        keys[1] = data.key;
        chat.emit('key', { key1: keys[1], key2: keys[0], stats: "ok"});
}
);
```

When the client sends his/her first client key ‘key1’, it is immediately saved. However, while sending the second client key ‘key2’, the handshake happens. The handshake process is asynchronous exchange of public keys to encrypt data between two clients.

In Cryp2Chat prototype the transmission of the incoming message is presented through the following scrip:

```
socket.on('msg', function(data)
{
        socket.broadcast.to(socket.room).emit('receive', {msg: data.msg, user: data.user, img: data.img});
}
);
```

Next, when the server receives an incoming message, the server sends it to the second client with the help of the socket.io library.

The public RSA key is generated in the following lines of script:

```
var myRSAkey = cryptico.generateRSAKey(PassPhrase, 512);
var PublicKeyString = cryptico.publicKeyString(myRSAkey);
```

The decryption of the cryptogram and its presentation in the client side is represented by the lines of the script:

```
var msgs = cryptico.encrypt(textarea.val(),roomKey);
socket.emit ('msg',{msg: msgs, user: name, img: img});
```

The client is the only one who can decrypt the transmitted message, as the private key never leaves the client side. The connection is made directly from client to client.

```
socket.on('key',function(data)
{
        console.log(data);
        console.log(yourName.val());
        console.log(hisName.val());
```

```
        if(myId == 1)
{
console.log("roomKey" + roomKey);
        roomKey = data.key1;
}
 else
        {
        console.log("roomKey" + roomKey);
        roomKey = data.key2;
        }
}
);
```

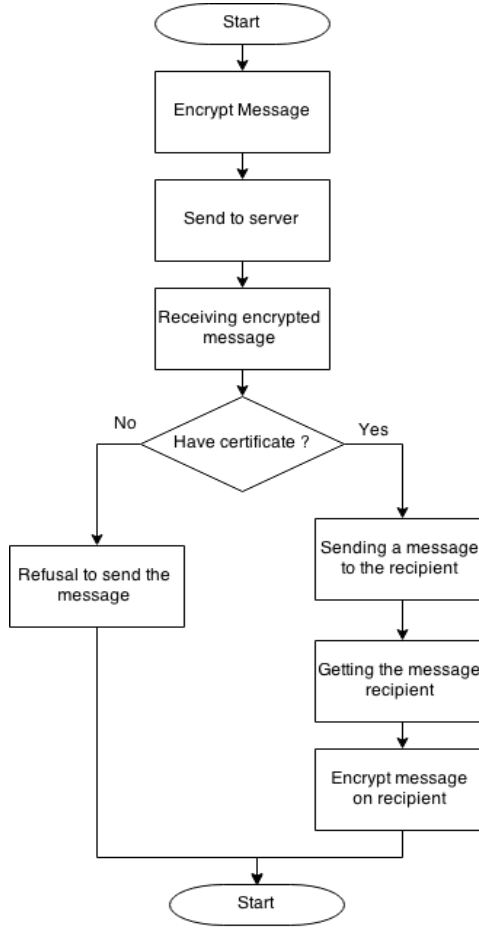

Fig. 3. Generalized Algorithm of Cryp2Chat Application

The script describes the client-side function that implements handshake. The generalized algorithm of the Cryp2Chat application is illustrated in Fig. 3.

As shown in the flowchart, from the moment of receiving the encrypted message and till the moment the message is sent to the recipient, the server undertakes the only action – certificate (i.e. license) verification. All the other steps associated with encryption, key generation, the transmission of the cryptogram to the recipient and decrypting of the cryptogram by the recipient, occur at the clients and in their browsers.

4. Experiment procedure

Experimental study of the application was conducted on a typical mobile phone, where Cryp2Chat program was installed. Mobile phone is Nexus 5 with the processor speed 2260 MHz and with the operating system Android 4.4.4. This operating system supports novelties related to the safe operation in the browser. When a user opens an application, it verifies the certificate on the sender's device. In case of a successful verification the sender chooses a receiver. In case the connection is completed, the receiver's public key and a signature are taken from the browser local database, or they are requested from the server.

Next the program encrypts the message and the sender's signature key. The message is sent to the server, and it verifies this signature on the basis of the contacts list. If the sender's signature exists in the server database, the latter immediately transmits the message to the recipient. In case of an incoming message the signature of the recipient is verified and it is decrypted with a secret decryption key.

The experimental results with Cryp2Chat prototype are shown on Figures 4 - 7.

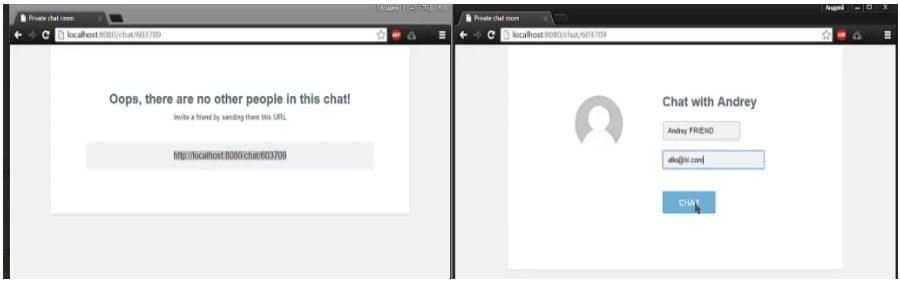

Fig. 4. Introducing the Users

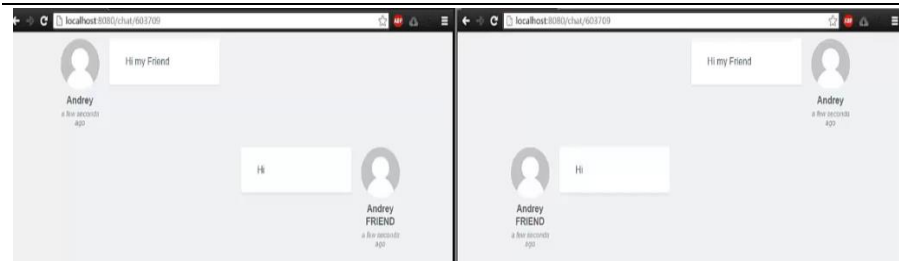

Fig. 5. Exchange with Test Messages

Fig. 6. Console of the First Client

On the console of the first client and on the server console one could see only the encrypted string. This way the information is transmitted to the server (Fig. 7). Additionally, the recipient - the second client - is the only one who possesses the key to decrypt it.

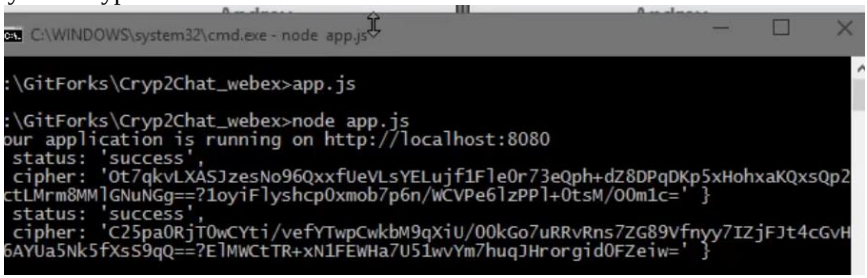

Fig. 7. Console of the Server

Further we conducted an experiment for a group of 20 users. Especially for this purpose we launched the site in the cloud Azure that hosts Cryp2Chat application - http://cryp2chat.azurewebsites.net/. Based on the experiment we have had the following results:

- high speed of response from the client's side as well as from the server side;
- a sufficiently high contact capacity of the program, as all 20 users managed to establish contacts with their subscribers simultaneously.

Fig. 8 is a table of qualitative indicators of Cryp2Chat application along with its analogues. In the table the following conventional symbols are employed:

	cryp2chat	SnapChat	WhatsApp	Telegram
End2End encrypting	V	V	V	X
Support of crypto signs	V	X	X	V
Business version	V	V	*	X
Oriented onto the Russian Federation and CIS market	V	X	X	V
Audio & Video translation	V	X	V	V
Crossplatform	V	X	X	V
Self-destruction of messages	V	X	X	V
The ability to work without a server	V	X	X	X

Fig. 8. Cryp2Chat Application and its Analogues

v – activated functional features of the program,

x – inactivated functional features of the program,

* – business version. There exists a business version, but it is patented under a different name and it might be a slightly different product.

Figure 8 illustrates the following advantages of Cryp2Chat application:

1) The application corresponds to all the parameters;

2) It provides a cross-platform messaging and self-destruction of messages;

3) It uses translator servers, i.e. working on peer2peer scheme.

5. Potential dangers

While designing the application three possible potential dangers were considered:

1. Brute force. Kaspersky blog has been used to assess the possibility of selecting passwords [3]. The program has shown that the selection of the password with a key of about 50 characters length, including special characters, will take more than 100,000 years. Even on a powerful botnet Conficker a password will be sorted out for ten thousand centuries.

2. Key theft. It is impossible for two reasons:

− If it is *android* application, the "sandbox" - a tightly controlled set of resources for the execution of the guest program - will not give to another application access to the files with a password,

− If it is *web* application, the call to a variable is impossible, as a pointer to an element is deleted, and it is only the inner code that can refer to this variable.

3. The application code cannot be changed because:

− If it is web application, then the downloaded code is stored when you start the application for the first time and it cannot be downloaded when you run,

− If it is the native application, changes in a code from the server side does not lead to a change of the client application code.

The transfer of potentially dangerous information (acts of terrorism, drug sales) is prevented because control data exchange is carried out with the use of an electronic signature. While registering the user generates a signature. This is a RSA key that is passed to the server, stored there and never changed.

When sending a message, the server checks the signature and if this signature is missing on the server, this message is not sent. Also, the signature may be withdrawn from server storage due to violation of the license agreement or similar cases. Thus, it is possible also to control the transmission of messages. Though we do not know what is encrypted in the message, we may deny the user in the network communication services.

6. Conclusion

In the future, we plan to rewrite the project from scratch and to implement it as a complete business solution with further access to the market. Additionally we plan to develop graphical password and voice authentication function. In addition, the plan is to transfer video, audio and other files.

References

[1] Tor – The Onion Router. Wikipedia, the free encyclopedia/ URL: https://ru.wikipedia.org/wiki/Tor#.D0.90. [08.08.2014]
[2] Tor: Overview URL: www.torproject.org/about/overview.html.en
[3] Blog.kaspersky URL: blog.kaspersky.com/password-check.

Создание приватного сервиса с использованием приложения CRYP2CHAT

Андрей Кирьянцев <reyzor2142@gmail.com>,
Ирина Стефанова <aistvt@mail.ru>
Поволжский государственный университет телекоммуникаций и информатики ПГУТИ, 443090, Россия, г. Самара, Московское шоссе, д. 77

Аннотация. Статья содержит описание приватного сервиса с шифрование и расшифрованием данных на стороне клиента с поддержкой технологии The Onion Router (TOR), которая позволяет устанавливать анонимное сетевое соединение, защищенное от прослушивания. С помощью сети TOR пользователи могут сохранять анонимность при посещении веб-сайтов, публикации материалов, отправке сообщений и при работе с другими приложениями, использующими протокол TCP. Безопасность

трафика обеспечивается за счёт использования распределённой сети серверов (onion routers). В статье описано прямое соединение - клиент к клиенту. Современные мессенжеры – программы для обмена сообщений в реальном времени оставляют метаданные на центральном сервере в незашифрованном виде, что позволяет узнать информацию об абонентах, времени и количестве сообщений в сессии. Авторами предлагается программа шифрования данных на клиентской стороне CRYP2CHAT, которая устраняет этот недостаток. Отправление сообщения через сеть TOR осуществляется с использованием асимметричного шифрования сообщения, например, методом RSA с возможностью использования и других алгоритмов шифрования. В статье приведен алгоритм работы программы, описаны способы защиты от некоторых сетевых атак по типу MITM, проверка наличия «чистого» сервера, самоуничтожение сообщения после закрытия сессии, а так же эксперимент работы прототипа. Рассмотрены потенциальные опасности внешнего характера в виде подмены серверного кода, подбора пароля и кражи приватного ключа, которые могут повлиять на конфиденциальность передачи данных. Так же описан пример работы технологии The Onion Router, которая позволяет добиться защиты от MITM, анонимности и проксификации. Кроме того, в статье приводится сравнение качественных показателей Cryp2Chat с его аналогами.

Ключевые слова *– криптография, шифрование, end2end шифрование, node.js, cryprico, java script, MITM-атака*

DOI: 10.15514/ISPRAS-2015-27(3)-19

Для цитирования: Кирьянцев Андрей, Стефанова Ирина. Создание приватного сервиса с использованием приложения CRYP2CHAT. Труды ИСП РАН, том 27, вып. 3, 2015 г., стр. 279-290 (на английском языке). DOI: 10.15514/ISPRAS-2015-27(3)-19.

Список литературы

[1] Tor – The Onion Router. Wikipedia, the free encyclopedia / URL: https://ru.wikipedia.org/wiki/Tor#.D0.90. [08.08.2014]
[2] Tor: Overview URL: www.torproject.org/about/overview.html.en
[3] Blog.kaspersky URL: blog.kaspersky.com/password-check.