Using End-to-end Data Encryption to Secure SIP Device Configuration

Xudong Chen
School of Software Engineering
Beijing Jiaotong University, 100044
Beijing, China
Email: chenxd@bjtu.edu.cn

Wenjun Fan
Departamento de Ingeniería de Sistemas Telemáticos
ETS de Ingenieros de Telecomunicación
Universidad Politécnica de Madrid, 28040
Madrid, Spain
Email: efan@dit.upm.es

***Abstract*—Security between the endpoints is an important requirement for network transmission. One existing solution of End-to-End security is based on the certificate of the third-party enterprise such as the authorization from ISP. And the other methods usually established a virtual private tunnel between the two endpoints. Considering the SIP device configuration basically used tftp protocol to transmit the configuration file, we proposed a free End-to-End security scheme using Application Gateway and data encryption to secure this transmission process. In this paper, Application Gateway used FastCGI programming to protect the HTTP server from attacking. On the other hand, the file encryption approach applied symmetrical encryption to ensure that only the authentic client could share the data from the server.**

Keywords-End-to-End security; Application Gateway; FastCGI; SIP Device Configuration

I. INTRODUCTION

End-to-end security relies on protocols and mechanisms that are exclusively used on the endpoints with a connection. The most typical example is an HTTPS connection (based, for example, on Transport Layer Security (TLS) [1]) to a Web server; IP Security (IPsec) [2] can also be used for end-to-end security, which was initially proposed as a default connection mechanism for IPv6.

In the traditional definition, the end-to-end security usually starts from the client side and then ends on the server side. However, this definition should be adjusted because of the multitude of parallel applications running on an operating system and the virtualization technology. The operating system can establish a security connection using session on application level. It also can be terminated in the front end, which is on behalf of numerous servers, as a case in many TLS [1] deployments.

Admittedly, an endpoint is an entity that communicates with another entity on the network. This definition, albeit vague, is enough for discussion at hand to understand why the network has a role to play in the security.

However, in this article, the definition of an endpoint is a server or a client. End-to-end security including sever-server, client-server, or client-client ensures secure communication. Several solutions had been proposed, such as eDonkey for the peer-to-peer [3] [4] that is a kind of client-client way, and VPN for client-server security used by many enterprises for special purpose.

VPN uses IPsec, SSL, TLS or Socks5 to implement E2E security. However, VPN must establish a virtual and private tunnel between two endpoints, and the user has to get the certificate from the third-party enterprise such as ISP.

IPSec works at the network layer in TCP/IP protocol stack. If the developer wants to filter the IP packets, they have to revise the kernel of the operating system. We can get the source code for free from the open source operating system like Linux; Under Windows, it is impossible to get the source from Microsoft, we just can use the Drive Development Kits (DDK) to capture the IP packet. Thus, the platform difference limits the wide use of IPSec.

Tunneling is a method used to circumvent firewall restrictions by disguising forbidden traffic such as P2P, remote computing and telnet - as ordinary Web surfing content. Tunneling could also be used by various software and even Trojan horses to transmit unexpected traffic to a remote server. Tunneling is possible because of the fact that organizations must allow HTTP, and usually also FTP and SMTP traffic transmit through the firewall in order to conduct routine business and allow their users surf the web, download files and use email.

Some studies had proposed several methods to build a reliable network access on the public network to transmit the data through the gateway, such as HTTP Tunnel [5], a method and system for providing a persistent HTTP tunnel for a connection-oriented protocol between a client and a Web server. Basically, there are two ways to implement HTTP Tunnel: (1) HTTPS, using SSL [6] (Secure Socket Layer) 443 port; (2) applying HTTP 80 port.

In the first method, with the 443 port, a TCP connection can be established, which implements bidirectional data transmission. According to SSL protocol, all data passing through the 443 port is encrypted, which the firewall can't block, so all packets can pass through the firewall transparently. Although the 443 port can be achieved through NAT and firewall penetration, but based on some security causes, some

978-1-4799-1282-7/13/$31.00 ©2013 IEEE

business systems will close the 443 port. Thus, hackers can make use of the fact that a firewall can't decrypt the data passing through the port 443 to invade or attack the systems.

In the second method, depends on HTTP protocol, the HTTP tunnel through port 80 can protect the IP packet passing through firewall and NAT, avoiding being filtered. But to establish a special connection of the HTTP tunnel will cost many resources, such as the network bandwidth. And in some cases, the efficiency of communication with this method is also a problem.

eSafe Gateway [7] with AppliFilter can identify and block HTTP-tunneling activities. AppliFilter monitors gateway traffic in real-time, making sure for instance, and only standard HTTP packets are allowed passing through port 80. However, nasty user also can fabricate standard HTTP packets to send the uninspected traffic to a remote server. So, eSafe can't solve the network-based security.

Manuel Crotti et al. [10], detecting HTTP tunnels with Statistical Mechanisms, propose the application of a statistically-based traffic classification technique to solve this problem. By the analysis of parameters– arrival time, size and order of the packets crossing a gateway, they showed that it was possible to detect with high accuracy whether an observed flow is carrying a legitimate HTTP session, or the flow is being used to another tunnel protocol. The authors described how this technique can be used effectively to enhance Application Level Gateways and firewalls, helping to better apply network security policies but can't insure the end-to-end security.

Internet Server API (ISAPI) filters improved the internet servers with custom features such as enhanced logging of HTTP requests, custom encryption and compression schemes, or new authentication methods. The filter application locates between the client side and the HTTP server side. Depending on the options that the filter application requests notification for, it can make several server actions. Filters can be set to receive notification when corresponding events occur, including reading raw data from the client, processing the headers, managing communications over a secure port using Personal Communications Technology (PCT) or Secure Sockets Layer (SSL), or handling other stages in the processing of the HTTP request.

The principal difference between the CGI programming model and the ISAPI programming model is that CGI creates a unique process for every request, while ISAPI does not. With CGI, every time an HTTP server receives a request, it must initiate a new process, and maintain the processes, which is very resource intensive. This inherent limitation in CGI has made it difficult to develop responsive applications on the Internet. However, the ISAPI only can be used in IIS, so we still use CGI because of its platform independent features. In our scheme, Application Gateway only opens the port 80, and the data output from the server has to be encypted by the FastCGI application. And only the legal client who knows the key could decrypt the ciphertext.

Granular computing (GrC) is an emerging computing paradigm of information processing. Data processing is an aspect of information processing and the data security is an important issue. In this paper, we proposed a scheme to protect the information of SIP device configuration by using end-to-end data encryption. The article is organized as follows: In Section 2, we briefly describe the technical background; in Section 3, we propose the implementation of free end-to-end security; in Section 4, the experiment shows the result; in the Section 5, we make a conclusion.

II. TECHNICAL BACKGROUND

A. Application Gateway

An Application Gateway is a type of firewall. All internal computers establish a connection with the proxy server. The proxy server performs all communications with the Internet. External computers only see the IP address of the proxy server and never communicate directly with the internal clients. The Application Gateway examines the packets more thoroughly than a circuit-level gateway before making packet-forwarding decisions. Obviously, it is more secure, but uses more memory and processor resources. Also known as application proxy or application-level proxy, an Application Gateway is an application program that runs on a firewall system between two networks. When a client program establishes a connection to a destination service, it connects to an Application Gateway, or proxy. The client then negotiates with the proxy server in order to communicate with the destination service. In fact, the proxy establishes the connection to the destination behind the firewall and acts on behalf of the client, hiding and protecting individual computers on the network behind the firewall. This creates two connections: one locates between the client and the proxy server and the other locates between the proxy server and the destination. Once the connection established, the proxy makes all packet-forwarding decisions. Since all communication is conducted through the proxy server, computers behind the firewall are protected.

However, this is secure method of firewall protection, Application Gateways require significant memory and processor resources compared to other firewall technologies, such as stateful inspection.

B. FastCGI

FastCGI is viewed as a new implementation of CGI, designed to overcome CGI's performance problems. The major implementation differences are:

(1)FastCGI processes are persistent. After finishing a request, they wait for a new request instead of exiting. Thus, the fundamental difference between FastCGI and CGI is that FastCGI applications are long-lived, which

means that the Web Server needs to rendezvous with a running application, rather than starting the application in order to explicitly communicate with it.

(2)The application library provides replacements for the C language standard I/O (stdio) routines such as printf() and gets(). The library converts references to environment variables, standard input, standard output, and standard error to the FastCGI protocol. Instead of using operating system environment variables and pipes, the FastCGI protocol multiplexes the environment information, standard input, output, and error on a single full-duplex connection. This allows FastCGI programs to run on remote machines, using TCP connections between the Web server and the FastCGI application.

FastCGI communicates the same information as CGI in a different way. Because FastCGI is CGI, and like CGI runs applications in separate processes, it suffers none of the server API problems listed above.

Since a FastCGI application does not always run on the same node as the HTTP server, we support two implementations of the connection: a *stream pipe* [8], for communications on the same machine, and TCP streams, for communication when the client and the server are on different machines.

The two security issues with remote FastCGI connections are authentication and privacy. FastCGI applications should only accept connections from Web servers that they trust (the application library includes support for IP address validation). Future versions of the protocol will include support for applications authenticating Web servers, as well as support for running remote connections over secure transport protocols such as SSL or PCT.

C. *Encryption*

Encryption is the process in which data (plaintext) is translated into something that appears to be random and meaningless (ciphertext). Decryption is the process in which the ciphertext is converted back to plaintext.

Encryption can be divided into hard encryption, soft encryption, and pseudo encryption

The so-called hard encryption is that each data bit of the original file does a bit transformation by the key. The output of the encryption is the garbled code of which people cannot make sense. Hard encryption encrypts the bit data of the file, completely independent above the operating system. So the ciphertext results from hard encryption can be decrypted by any other kind of operating system. Decryption is the inverse operation of the encryption. The bigger the size of the file is, the longer duration the encryption and decryption will take. Use of hard encryption is the fundamental method of ensuring data security.

Soft encryption does not process the data of the file transformation. It just applies some acrobatics and makes the confidential data can be accessed. But the data does present with plaintext form. For example, some software modifies the registry of Windows and makes the files with hidden attributes not visible.

Pseudo encryption is a kind of soft encryption, but it is more dangerous than soft encryption. It alters the disk sectors, utilizes system vulnerability, or adopts other baleful means to hide the data. Tiding the disk fragment or rebuilding the operating system will finish this kind of encrypted data easily.

It is difficult to determine how good an encryption algorithm is. Algorithm that seems promising sometimes turns out to be very easy to break, given the proper attack. When selecting an encryption algorithm, it is probably a good idea to choose one that has been around for a while, and has successfully resisted all attacks.

III. IMPLEMENTATION

A. *Application Gateway with FastCGI Programming*

The main advantage of Application Gateway is that the programmer doesn't need to change the operating system, and just create the Application Gateway. In many cases, if it is impossible to access the system source code, the programmers can't modify the operating system. However, programmers can use conventional programming tools to build Application Gateway, and the gateway does not require any modifications to the underlying protocol software. What's more, as soon as the site owns the gateway, then it will be able to apply standards programs of client and server.

For most situations, various institutions in order to address security issues chose the Application Gateway. For example, in order to establish a remote connection, any user on the host of the institutions had to call the client software. The software is initially connected to the Application Gateway, and the Application Gateway then orients the users to connect to the desired destination host.

For our context, SIP device configuration usually applies tftp protocol to transmit the configuration file that includes much private information. Thus, we proposed this idea to secure the process of SIP device configuration. Compared with other approaches, our implementation is much more light and resource efficiency.

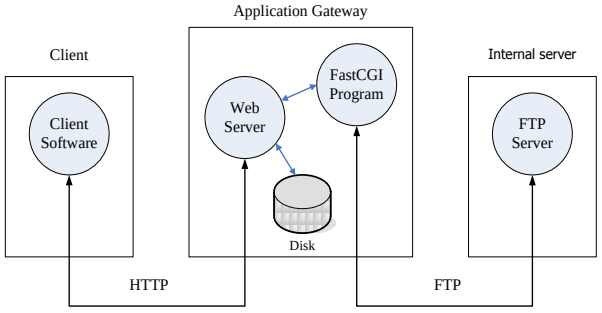

Figure 1. Application Gateway for Security

Our idea is illustrated in Fig. 1. The Application Gateway is a media of connection between the client and the destination server. In this situation, the server only opens the port 80 to provide service rather than open other port like 21. To achieve this kind of application gateway, we need four facilities: an HTTP and FTP access computers, a regular web server, Application Gateway program, and the mechanism to send file requests to the gateway program from a Web server. Fortunately, most web servers have the necessary facilities. The web server using FastCGI technology makes the server to be able to call the program.

The web server contains a FastCGI program [9] must deal with two types of requests. Upon receiving a request from client to open a regular web page or to get a file, the server obtains a copy of this page or file from the disk, and sends it to the client. Upon receipt of a request for information under FastCGI program management, the server call FastCGI program, and pass the request to it, and then wait for program execution. FastCGI program establishes a FTP connection, and accesses a copy of the file, and then the web server returns the result to the client.

From the client's perspective, there is no difference between conventional web pages and documents generated by the FastCGI program. In both cases, clients send a request first and then accept the document. Finally, the document is passed to the customer.

Thus, in our case, the SIP device send a request for configuring itself to the web server, and then the FastCGI program process this request, later it gets the configuration file from the ftp server or the disk, at last it replies the configuration file to the SIP device.

The client could request the FastCGI using the URL like this:
http://ServerName/cgi-bin/encryption.fcgi?FileName

"encryption.fcgi" is the FastCGI program in our system.

B. *Client Development*

We use the MFC program to develop the client software. The MFC class CHttpConnection manages the connection to an HTTP server. HTTP is one of three Internet server protocols implemented by the MFC WinInet classes.

Table I shows the steps in a typical HTTP client application.

TABLE I. HTTP CLIENT APPLICATION

Goal	Actions
Begin an HTTP session.	Create a CInternetSession object.
Connect to an HTTP server.	Use CInternetSession::GetHttpConnection.
Open an HTTP request.	Use CHttpConnection::OpenRequest.
Send an HTTP request.	Use CHttpFile::AddRequestHeaders and CHttpFile::SendRequest.
Read from the file.	Use CHttpFile.
Handle exceptions.	Use the CInternetException class.
End the HTTP session.	Dispose of the CInternetSession object.

Using the HTTP client application, the user could access the FastCGI application.

C. *Encryption/Decryption Process*

In this paper, we used Symmetric-key algorithm (or we call it Reciprocal Cipher). Once a message has been encrypted, it can be stored on nonsecure media or transmitted over a nonsecure network and still remain secret. Later, the message can be decrypted into its original form. This process is shown in the following illustration, Fig. 2.

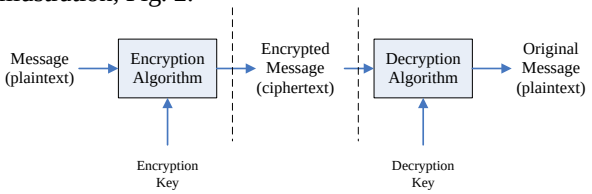

Figure 2. Encryption/Decryption Process

Data encryption and decryption are simple processes. When a message is encrypted, an encryption key is used. This is comparable to a key that is used to lock a padlock. To decrypt the message, a decryption key must be used. The encryption and decryption keys are often, but not always, the same key. In this article, we adopt the symmetric encryption principle, thus the both encryption and decryption keys are the same.

It is very important to keep the keys safe and transmit them securely to other users. This will be discussed further in the Security Strategy. However, the challenge is how to properly restrict access to the decryption key and the encryption algorithm, because anyone who possesses them will be able to decrypt all messages that were encrypted with the corresponding encryption key.

A symmetric encryption key (also known here as a session key) is used during both the encryption and decryption processes. In order to decrypt a particular piece of ciphertext, you must possess the key that was used to encrypt the data. Essentially, a session key consists of a random number, of approximately 40 to 2,000 bits in length. The longer the key that is used, the more difficult it is to decrypt a piece of ciphertext without possessing the key.

The goal of each encryption algorithm is to make it as difficult as possible to decrypt the generated ciphertext without using the key. If a really good encryption algorithm is used, then there is no technique better than methodically trying every possible key. Even for a key size of just 40 bits, this works out to 2^{40} (just over 1 trillion) possible keys.

In our system, one of the hard encryption algorithm pseudo-code is just like this:

```
1: char key[BUF], ch;
2: int i = 0;
2: fgets(key, sizeof(key), keyFileStream);
3: inFileStream = fopen(FileName, "rb");
4:          while(!feof(fileStream)){
5:                    ch=fgetc(fileStream);
6:                    fputc(key[i++]^ch, outFileStream);
7:                    if(key[i]=='\0'){
8:                              i=0;
9:                    }
10:         }
```

In the implement, the key size is 8 bits for encrypting and decrypting the configuration file. Obviously, you can enlarge the size of the key if you want to improve the secrecy of the process. One problem we must take consideration in our system is that the C language has the platform difference. In other words, for example, the result of function fputs() is a little different between Windows OS and Linux OS.

D. Security Strategy

We have mentioned that it is important to ensure the key security and transmit it safely to clients. In our context, our aim is to transmit the SIP device configuration file to the user, and one example of the SIP device configuration file is like this:

```
User Passwd=
ETH1 PPPOE User=
ETH1 PPPOE Passwd=
ETH1 IP Address=
ETH1 Subnet Mask=
Gateway Address=
HostName=
Domain=
1st DNS=202.106.0.20
2nd DNS=208.67.220.220
1st NTP Server=ch.pool.ntp.org
2nd NTP Server=216.218.192.202
ETH0 IP Address Mode=Static IP
DHCP Lease Time=864000
ETH0 IP Address=192.168.18.1
ETH0 Subnet Mask=255.255.255.0
Number Of Client IP Address=
DHCP Start IP Address=192.168.18.100
DHCP End IP Address=192.168.18.222
STUN Enable=no
STUN Test Enable=no
STUN Server=
Phone1 Encrypt Algorithm=NONE
Phone1 Encrypt Key=
Phone2 Encrypt Algorithm=NONE
Phone2 Encrypt Key=
Resync Rule=
Resync Periodic=3600
```

When the client asks for the configuration file from the server, the client has to send its serial ID encrypted by an initial password to the server side. We should know that the client equipment was sold by the server-side provider, thus this serial number and initial password was saved in the database of the server. When the server receives the encrypted serial ID, it can decrypt the serial ID by the initial password. Later, the server exams the serial number of the device of the client. If the serial number is authorized, the server will send the encrypted configuration file to the client. And the client can decrypt the configuration file with the decrypt algorithm and the decrypt key. Here, the key for encrypting and decrypting the SIP device configuration file is preset in the SIP device according to the algorithm that is related with the serial of the SIP device.

The algorithm and the key for encrypting and decrypting the SIP session data are from the parameters in the SIP device configuration file. To select which key and algorithm depends on the programmer.

Therefore, using this security strategy, the key for the client to decrypt the SIP device configuration file is preset, and we don't need to transmit it on the network. Thus, there is no potential security hazard that the hacker can hijack the key from network, only if someone can get the key that had been burned on the SIP device. Thus, the key and algorithm for encrypting and decrypting the SIP session data are safe, too. Only if hacker can get the key and algorithm that encrypted the SIP device configurations file.

IV. VERIFICATION

In this section, we describe the system test environment and the result of the tests.

A. Test environment

Table II shows the system configuration parameter of the two endpoints.

TABLE II. SYSTEM CONFIGURATION PARAMETER

Client System	
Operating System	Windows XP
CPU	Intel Core i5 460M 2.53G 2cores
Client development kit	Microsoft VC 6.0
Server System	
Operating System	Linux Fedora 10
CPU	Intel Core i5 460M 2.53G 2cores
Web Server	Apache httpd-2.0.64
Apache httpd FastCGI module	mod_fastcgi-2.4.6
FastCGI Development Kit for C language	fcgi-2.4.0

B. Result

The performance is a requirement to keep the algorithm feasible. This experiment demonstrated the performance of the proposed approach. Time consumption of File encryption or decryption was

shown in Figure 3. It is obvious that with the increasing size of the file needed to be encryption or decryption, the system takes more and more time to process the file.

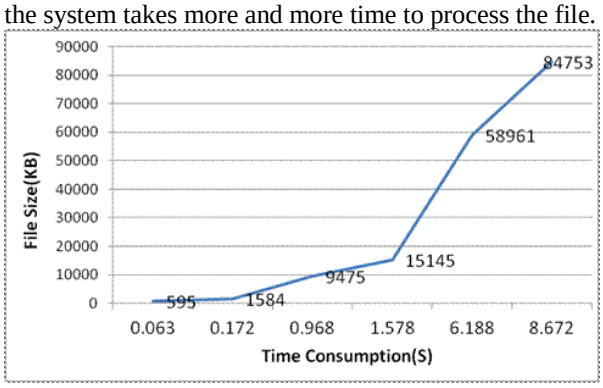

Figure 3. Encryption /decryption time cost

We can find that, if the file size is less than 1MB, and the system takes time less than 1s. As we all know, for most file data, the size is not very large. However, if the file is a streaming media file, such as a FLV format file, and the size is 84,753KB in the Fig. 3, the time cost is about 8s. It seems a little slow. However, our aim is to secure the transmit procedure of the SIP device configuration file. The size of this kind of file is basically less than 1MB. Thus, using our scheme the performance is reliable and efficiency. Besides, we never build any channel on the network, thus this approach won't impact on the bandwidth, and won't affect the network resource.

In this paper, we used the sequential (or centralized) information processing. Last but not least, if our system has a GPU, we could use the CUDA [11] programming to accelerate the speed of encryption and decryption process evidently, which is parallel (or decentralized) information processing. W. Fan, X.D. Chen and X.F. Li [12] had implemented a method using the CUDA to parallel encrypt the data based on using RSA.

V. CONCLUSION

In this paper, we proposed a novel scheme to implement the security in endpoints, especially server-client structure. The Application Gateway using port 80 protects the Web server from malicious access of the nasty users, and uses the FastCGI programming to access the client desiring destination. In addition, in order to ensure the data transition security, we use the symmetrical encryption to process the data at the sever end. When clients receive the data from the server, it uses the same encryption algorithm to decrypt the data.

Absolutely, End-to-end security protocols and solutions are an essential cornerstone in network security. In today's networks scenario, however, it is unrealistic to assume that end-to-end security solution is adequate. First, the cryptographic protocols that are used to provide secure communication are already often targeted by diverse attacks. The paper [13] proposed an anomaly-based detection system by sniffing the encrypted traffic, extract features to construct normal usage behavior profiles. Due to the deviations from these normal profiles, the system can detect the suspicious activities. Second, there are countless examples where network-based security is also essential, and where the end-to-end security solutions are not only useless, but also present an extra problem [14]. Thus, in order to have robust end-to-end security, it is essential to combine with network-based security solutions and also necessary to make a combination with intrusion detecting system. This is one of our future works.

In addition, inspired by the paper [15], we are also supposed to consider whether our approach could be used to secure the media stream in peer-to-peer systems. This is another future work for us to extend the application of our scheme.

REFERENCES

[1] T. Dierks, et al., "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246, August 2008.

[2] S. Kent, et al., "Security Architecture for the Internet Protocol," RFC 4301, December 2005.

[3] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-peer computing. Technical Report HPL-2002-57, HP Labs, March, 2002

[4] A. Oram, editor. Peer-to-peer: harnessing the power of disruptive technologies. O'Reilly & Associates, Inc., March, 2001.

[5] Rodger D. Erickson, St. Louis, MO(US); Ronald D. Sanders, Spokane, WA(US). METHOD AND SYSTEM FOR PROVIDING A PERSISTENT HTTP TUNNEL, Method and system for providing a persistent HTTP tunnel, United States Patent Application Publication, Pub.No.:US 2002/0156901 A1, Pub. Date: Oct. 24, 2002

[6] Kipp E.B. Hickman "The SSL Protocol," Nov. 29th, 1994

[7] New Threats Requiring. Gateway-level. Application Filtering, White paper, eSafe, September 26, 2006

[8] W. Richard Stevens, UNIX Network Programming, Prentice-Hall, Section 7.9, 1990

[9] http://www.fastcgi.com/devkit/doc/fastcgi-prog-guide/cover.htm, April 10th, 2011

[10] Crotti, M.; Dusi, M.; Gringoli, F.; Salgarelli, L., "Detecting HTTP Tunnels with Statistical Mechanisms," *Communications, 2007. ICC '07. IEEE International Conference on* , vol., no., pp.6162,6168, 24-28 June 2007

[11] NVIDIA Corporation. NVIDIA Compute Unified Device Architecture Programming Guide Version 2,April 2009

[12] Wenjun Fan; Xudong Chen; Xuefeng Li, "Parallelization of RSA Algorithm Based on Compute Unified Device Architecture," *Grid and Cooperative Computing (GCC), 2010 9th International Conference on* , vol., no., pp.174,178, 1-5 Nov. 2010.

[13] Fadlullah, Z.M.; Taleb, T.; Vasilakos, A.V.; Guizani, M.; Kato, N., "DTRAB: Combating Against Attacks on Encrypted Protocols Through Traffic-Feature Analysis," *Networking, IEEE/ACM Transactions on* , vol.18, no.4, pp.1234,1247, Aug. 2010

[14] Michael H. Behringer, "End-to-End Security," The Internet Protocol Journal, Volume 12, No.3, September 2009

[15] Zhijie Shen, et al, "Peer-to-Peer Media Streaming: Insights and New Developments," Proceedings of the IEEE 99(12): 2089-2109 (2011)