Secure Networking with NAT Traversal for Enhanced Mobility

Lubomir Cvrk[1], Vit Vrba[1]

[1] Brno University of Technology, Dept. of Telecommunications, Purkynova 118, 61200 Brno, Czech Republic
{cvrk, vrba}@westcom.cz

Abstract. When a peer in a public network opens a connection to another one being behind a network address translator, it encounters the network address translation problem. So called "UDP hole punching" approach allows to open a public-to-private or private-to-private network connection. This article deals with this approach to propose new security architecture for IPv4 communication introducing so called "implicit security" concept. Main contributions are ability to interconnect to any host behind NAT using just a host's domain name, enhanced mobility, and encryption and authentication of all data transmitted through this connection right from a packet sender to a local receiver. Secure channel is established on-demand automatically and is independent on any application. No additional modification of current NAT, IPv4 or DNS is required.

1 Introduction

Communication in IPv4 networks encounters the network address translation (NAT) problem. When a peer needs to initiate communication with another one which is behind a NAT box, it is basically impossible.

But there are several approaches, how to obviate this problem. One of them uses so called TCP/UDP hole punching, introduced by Daniel Kegel in [3], fairly well described in [4]. This approach uses NAT characteristic behavior when a connection from a private to public network is established. A NAT box creates a "hole" – a rule that allows packets from the host in the public network pass into a private network destined to a host which has opened the connection. In case of the TCP protocol this rule exists until TCP connection is closed. In case of the UDP protocol the situation is a little bit different, because the UDP does not use the institute of a two sided connection. Packets sent by the UDP across a NAT box out to the public network force it to create appropriate address translation rule. But there is no sequence for closing this communication (it is not connected) and therefore there is no object or flag usable for the control of the translation rule existence. Hence NAT implementations use time-outs for incoming packets. The time-out is as short as 20 seconds for a reply, but this is not standardized constant, just experimental results (see [4]).

Application can cryptographically secure its data transmissions with several approaches. The first one, and probably mostly used, is security built into an application. This approach requires a good crypto library and developers with cryptographic skills. A user of a software system is then in mercy of its developer, whether she builds her system in such a way that it provides cryptographic protection of a transmitted data, or not. He has to look for the information about encryption to know how confident data could he enter.

Better approaches how to protect communication operate at lower layers than application level security providers. The SSL virtual private networking (VPN, [7]) or IPSec ([5]) are well known.

Both these VPN implementations encounter several disadvantages: VPN server must exist in the public network or at least NAT box must be configured to route VPN traffic to a VPN server in a private network.

After correct authentication to the VPN server all data transmitted over the Internet between the user's computer and the VPN server are encrypted, messages are reply-protected and authenticated. But when a user starts to communicate with some other node in the Internet, everything is as bad as without VPN protection. It is needed to remark that even if in the VPN, the communication with another local network host behind the VPN server (on the local network) is also insecure.

2 Implicit security model

Implicit security is inspired by the opportunistic encryption – the idea introduced by John Gilmore in the FreeS/WAN project [9]. Implicit security, the concept we introduce, establishes a secured channel whenever it is possible. When a destination host is behind NAT, no VPN is needed and the channel is decrypted by that host so the end-to-end communication security problem is solved. Nobody but the sender and the receiver reads the messages.

3 Architecture components

Architecture of the system works with IPv4 and requires three main components: 1) Connector server in the public network tightly cooperates with appropriate DNS server; 2) Connector client establishes connections using connector server (initiates the UDP hole punching process); 3) Packet processor is installed on every participating host.

3.1 Packet processor

The network packet processor is bumped in the protocol stack below the network level. It captures outgoing packets, asks the connector client to create a secure channel and passes packets into this channel.

The packet's IP payload is packed in the UDP protocol with appropriate port and destination setup. Incoming packet comes back to the specified UDP port, so packet processor detects it and unpacks. The restored original packet is then passed up to the user-space. This simple technique allows the channel to be very easily traversed across NAT boxes or firewalls, because the data of all protocols are packed and transmitted over one single UDP port.
When the packet processor detects outgoing packets, it automatically tries to establish secure connection. For this service it asks the connector client.

3.2 Connector client

The connector client logs in a connector server when it starts (usually with the operating system, it runs as a service), and waits for the packet processor's request to connect to another host.

3.3 Connector server

The connector server must be connected in the public network, so connector clients are able to connect from private networks.

The server handles the UDP hole punching process. For details of the UDP hole punching, see [3], [4].

The server also handles client's login process. This requires some cooperation with appropriate DNS server.

The server may keep a database with public keys and may be configured in the strict mode to accept only signed login requests.

4 Architecture overview

In the following figures these terms are used: "Detec" – we named this system "Detec" [ditek], the abbreviation of "Decentralized end-to-end communicator"; "Detec peer" or "Caller" is a host which requests communication with another host – "called host", which may be behind NAT.
Below is described the architecture from the caller's point of view.

4.1 Login to connector server

When a client starts the connector client, it tries to log in the connector server. IP address or domain name is known from the configuration of the client. This example is configured in such a way that client (Detec peer) is behind NAT in a private network.

The fig. 1 shows the messages sent between client and server:

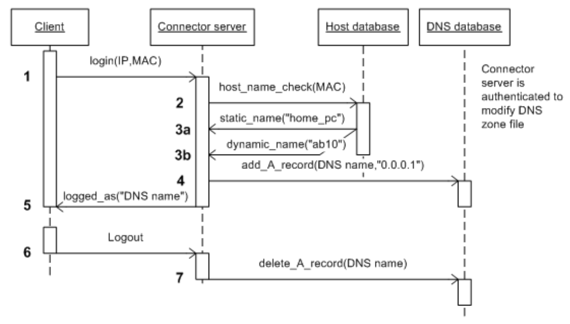


Fig. 1. Login process

First, the client sends the server a message with a login request, and as a parameter it passes the MAC address of its network interface and its local IP address (1).

The server checks the static host database (2) whether this client should be assigned a static (3a) DNS name or dynamic (3b). When a client is to receive a static DNS name, the login request message must be therefore signed and the server must know client's public key. In this case the server also authenticates itself to the client using public key cryptography.

The server then saves appropriate DNS name to the DNS server's database (4) and assigns host's DNS name to IP address 0.0.0.1. This IP address is not routable and is from the reserved address space 0.0.0.0/8 meaning "this network". Below is explained why.

The logout process is very similar to login and the server deletes client's A record from the DNS server database.

The DNS server should have a special zone (zones) reserved for the purpose of this system, i.e. third (fourth etc., depends on administrator) level domain and must be primary. In real life this is the DNS server handling public-IP-to-name mapping of the local network gateway's public interface. The connector server and the DNS may be installed on the same host (better for security).

4.2 Opening a connection

When the client is logged in the connector server it can open a connection. Every connection starts with the DNS query, represented by the function "`gethostby-name`". The result of this function is IP address of the host.

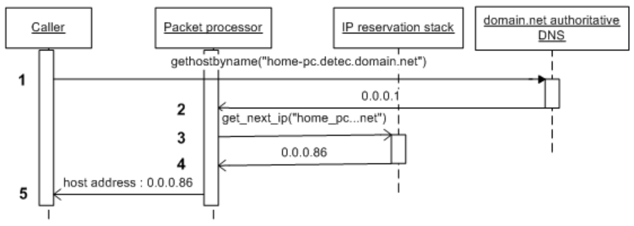

Caller
Packet processor
IP reservation stack
domain.net authoritative DNS
gethostbyname("home-pc.detec.domain.net")
1
0.0.0.1
2
get_next_ip("home_pc...net")
3
0.0.0.86
4
host address : 0.0.0.86
5

Fig. 2. Finding host's IP address

The fig. 2 shows the example how to get home_pc's address. First, the `gethost-byname` socket API function is called. This function generates DNS query to resolve the domain name "`home-pc.detec.domain.net`". The authoritative name server answers "0.0.0.1". The query was notified by the packet processor. When the response is received, the packet processor detects that IP is 0.0.0.1. This tells to the packet processor that some application from an upper layer needs to connect to a host being behind NAT. The packet processor accepts the DNS UDP packet but does not pass it to upper layers. Instead it takes a look in its "Detec IP reservation stack", where DNS name and appropriate assigned IP is stored. The stack answers 0.0.0.86. This means, that the stack saved the pair (`home-pc.detec.domain.net`, 0.0.0.86). Note that the stack stores global IP addresses too, where no modification is required. The packet processor modifies then the resolved DNS data and replaces "0.0.0.1" with "0.0.0.86" and passes the packet up. The global 0.0.0.1 IP address assigned to every Detec's host is on the local machine transformed to the first available Detec IP address. This address space provides more than 16 millions of single IP addresses. It is a very low probability that this space would ever exceed, because one host usually does not communicate with so many others roughly in the same time. Every established connection is automatically closed after the network inactivity time is out.

4.3 Data transmission and packet processing

Once the application receives the destination's IP address (0.0.0.86) it tries to open a communication channel (i.e. TCP). On the fig. 3 it is represented by the tcp_SYN (1). This packet is captured by the packet processor, because its destination IP (0.0.0.86) is in the IP reservations stack. Because the IP is from the network 0.0.0.0/8 the processor knows that a destination (called host) is behind NAT. That is why a UDP holes on both NAT boxes (caller's and called host's) must be opened.

4.3.1 Tunnel creation

Opening these holes is done this way: The packet processor asks the connector server for the called host's public IP address and a UDP port where a secured tunnel will be established (2a). It opens a UDP connection to the server (2b). Connector server sends a request to open a UDP connection (3a) to the called host and it opens it (3b). Server sends the message 3a through a TCP connection that the connector client has opened to the connector server at login process. The connector server is the called host's one, not the caller's. Its IP is translated from `detec.domain.net`.

Once both clients opened the UDP connection to the server it saves the UDP source port values (received from NAT boxes). Then it sends both sides the public IPs and the UDP ports to use (4a, 4b). When they receive this data they start sending UDP packets and the UDP hole is created on the both sides (5, 6). In the event of received packet from the opposite host (7) they both send `udp_hole_punching_done` messages to each other (8,9). If any of them does not receive this message in a configured time-out (in seconds), then the process state resets and begins again. If three attempts fail, then the application packet is dropped. In future versions this may need some interaction with a user to decide what to do.

The results of the UDP hole punching (NAT public address and the UDP port) are associated with appropriate 0.0.0.0/8 address in the internal connection database.

4.3.2 Tunneling

The original packet is processed and encapsulated into the UDP secure tunnel (11). On the called host's side it is processed by the packet processor and sent to appropriate application. The response is packed by the called host and sent to the caller. After message number processing, source IP verification and message authentication the packet is decrypted and passed to the upper layers, otherwise it is dropped.

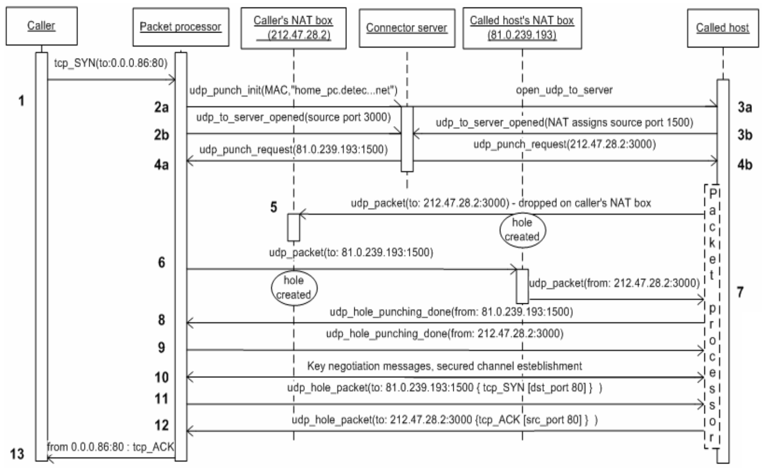

Caller
Packet processor
Caller's NAT box (212.47.28.2)
Connector server
Called host's NAT box (81.0.239.193)
Called host
tcp_SYN(to:0.0.0.86:80)
1
udp_punch_init(MAC,"home_pc.detec...net")
open_udp_to_server
2a
3a
udp_to_server_opened(source port 3000)
udp_to_server_opened(NAT assigns source port 1500)
2b
3b
udp_punch_request(81.0.239.193:1500)
udp_punch_request(212.47.28.2:3000)
4a
4b
5
udp_packet(to: 212.47.28.2:3000) - dropped on caller's NAT box
hole created
udp_packet(to: 81.0.239.193:1500)
6
hole created
udp_packet(from: 212.47.28.2:3000)
7
Packet processor
udp_hole_punching_done(from: 81.0.239.193:1500)
8
udp_hole_punching_done(from: 212.47.28.2:3000)
9
Key negotiation messages, secured channel esteblishment
10
udp_hole_packet(to: 81.0.239.193:1500 { tcp_SYN [dst_port 80] })
11
udp_hole_packet(to: 212.47.28.2:3000 {tcp_ACK [src_port 80] })
12
from 0.0.0.86:80 : tcp_ACK
13

Fig. 3. Data transmission

5 Packet processing

The packet processor encrypts and authenticates the outgoing packets and decrypts and verifies incoming packets. Outgoing packets are immediately sent over the UDP channel and incoming are received from the UDP channel.

This requires a protocol that supports encryption and authentication with reply protection. That is why we have designed the protocol SPUT (Secure protocol for UDP tunneling).

5.1 Secure protocol for UDP tunneling

This protocol defines some overhead fields required for successful decryption and reply protection.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        message number                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            reserved           |    command    |  next header  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                      Original IP payload                      |
~                        variable length                        ~
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
|                                                               |
|                  message authentication code                  |
|                          HMAC-SHA256                          |
|                                                               |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fig. 4. SPUT protocol definition

The field *message number* indicates the number of currently processed packet. When it exceeds 32 bits, the key negotiation process is started to negotiate new session key. The system keeps track of received messages and once it receives a message it can not accept a message with the same message number. A 32 bit value is used internally where each bit is a flag informing the packet processor whether the message with current message number was already received or not. This value provides a window for 32 messages, because they may come out of order. When a message is received bits in this flag are rotated left by appropriate number of bits and a flag is set to 1. Messages with the message number smaller than current message number decreased by 32 are dropped as well as messages already received. The field *command* is used to control the encryption / decryption process. Nowadays it uses only the value 1, which closes the connection. In this case an empty packet is sent. The *next header* field indicates the protocol number encapsulated in SPUT. The *original IP payload* is the field of variable length used for TCP or UDP (or any other) data from the original packet. It does not contain original IP header. The last field of the protocol is *MAC*. HMAC-SHA-256 is used for authentication computation, exactly as described in [8].

The fig. 5 shows encrypted and authenticated fields of the SPUT protocol. SPUT is designed to use encrypt-first-authenticate-then method of MAC computation.

```
|messageNumber{32}|reserved{16}|command{8}|nextHeader{8}|payload{?}|MAC{256}|
                  +------------------------------------------------+
                           E n c r y p t e d     AES-CTR-128
+------------------------------------------------------------------+
           A u t h e n t i c a t e d      HMAC-SHA-256
```

Fig. 5. Encrypted and authenticated fields

The SPUT protocol's overhead is 320 bits. Data are transported as the UDP payload whose overhead is 64 bits. That is why the system requires decrement of the maximum transmission unit by 384 bits = 48 bytes.

In case of multiple hosts behind NAT talking to one host (*H*) the tunnel and the payload destination ports must be remapped in such a way that *H* stores the original packet's source ports as future destination ports and substitutes the IP payload's port (if payload is TCP or UDP) for another port assigned by *H*. This allows *H* to identify more hosts hidden behind the public network interface of one NAT box.

5.2 Encryption process

Encryption process uses the AES-128 in CTR mode. The whole process requires computation of a key stream using the `message number` (and a secret key of course) as an input.

The key stream k is generated as follows:

$$k_j = E(K, j \| i \| s); j = 0.l/128$$

j is a number of current 128 bits long block of data to encrypt (8 bits). *i* is a SPUT message number (32 bits). *s* is a constant made of first (least significant) 88 bits of the secret session key. *l* is a number of bits of the data to encrypt. These values are concatenated to a 128 bits input block for AES encryption. k_j is 128 bits key-stream segment.

6 Experiments and results

We have tested several parts of the architecture. The most important information we had to find out was whether the packet processing overhead is acceptable. That is why the overall performance has been tested and compared to the implementation of IPSec.

6.1 Packet processor overall performance

We have implemented this architecture in C++ for the Microsoft Windows platform. This platform requires an NDIS kernel driver that hooks over the network interface and captures all traffic and passes it to the user space.

There is one disadvantage in out current implementation. It transfers captured packets from the kernel-mode to user-mode so we expected this to impact performance especially on the high speed lines. The measurements have proven this (see fig. 6).

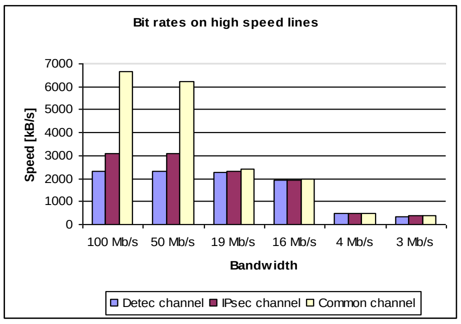


Fig. 6. Bit rates on high speed lines

The speed limit of this kind of implementation is 2293 kB/s – about 17.9Mb/s. This is acceptable for most use-cases because the system is designed for end-users not routers or gateways which must not use it.

In comparison with Microsoft's IPsec implementation in Microsoft Windows XP Professional our implementation is slower but not significantly. The speed limit of IPsec is around 3000 kB/s. IPsec is probably implemented in kernel mode and its code is optimized. Our code is just an initial version built with debug information and with no source code or compiler optimizations.

6.2 Performance on low bandwidths

On low bandwidths (1.5Mb/s and lower) the packet processor is as fast as IPsec and common (insecure) channel (see fig. 7). The fact of encryption and authentication is not perceptible since the bandwidth is 19 Mb/s or slower (see fig. 6).

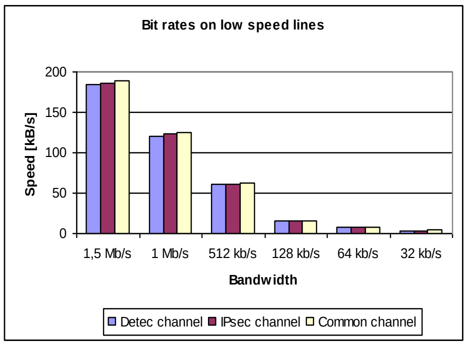


Fig. 7. Bit rates on low speed lines

6.3 Performance impact ratio on low speeds

We have compared the performance impact ratio of IPsec and Detec to the common (insecure) communication channel of the IP protocol - fig. 8.

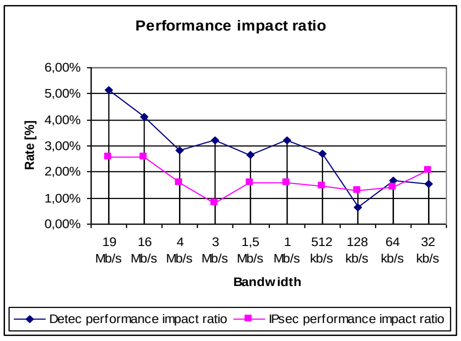


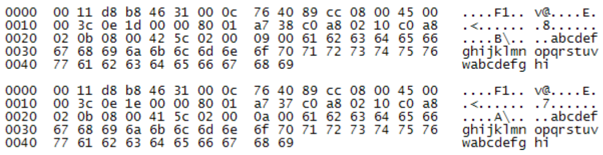
Fig. 8. Performance impact ratio

The rate means how many percents is particular encrypted connection slower than insecure one.

From the graph can be seen that on very low speeds (128kb/s and less) IPSec and Detec produce almost the same performance. This is due to the overhead of ESP protocol which is greater than that of SPUT protocol.

6.4 Packet analysis

For the reason of encryption checking we have analyzed the data transmitted through the network. The fig. 9 shows sequence of plain ICMP echo-request messages.

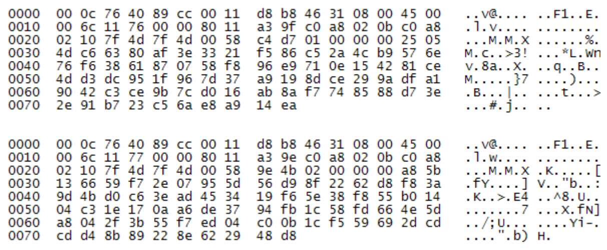


```
0000  00 11 d8 b8 46 31 00 0c  76 40 89 cc 08 00 45 00   ....F1.. v@....E.
0010  00 3c 0e 1d 00 00 80 01  a7 38 c0 a8 02 10 c0 a8   .<...... .8......
0020  02 0b 08 00 42 5c 02 00  09 00 61 62 63 64 65 66   ....B\.. ..abcdef
0030  67 68 69 6a 6b 6c 6d 6e  6f 70 71 72 73 74 75 76   ghijklmn opqrstuv
0040  77 61 62 63 64 65 66 67  68 69                     wabcdefg hi

0000  00 11 d8 b8 46 31 00 0c  76 40 89 cc 08 00 45 00   ....F1.. v@....E.
0010  00 3c 0e 1e 00 00 80 01  a7 37 c0 a8 02 10 c0 a8   .<...... .7......
0020  02 0b 08 00 41 5c 02 00  0a 00 61 62 63 64 65 66   ....A\.. ..abcdef
0030  67 68 69 6a 6b 6c 6d 6e  6f 70 71 72 73 74 75 76   ghijklmn opqrstuv
0040  77 61 62 63 64 65 66 67  68 69                     wabcdefg hi
```

Fig. 9. Sequence of ICMP messages

The fig. 10 shows these messages transported by the Detec system. They are transported within the UDP protocol port 32589, both encrypted by AES-128-CTR and authenticated using HMAC-SHA-256.



```
0000  00 0c 76 40 89 cc 00 11  d8 b8 46 31 08 00 45 00   ..v@.... ..F1..E.
0010  00 6c 11 76 00 00 80 11  a3 9f c0 a8 02 0b c0 a8   .l.v.... ........
0020  02 10 7f 4d 7f 4d 00 58  c4 d7 01 00 00 00 25 05   ...M.M.X ......%.
0030  4d c6 63 80 af 3e 33 21  f5 86 c5 2a 4c b9 57 6e   M.c..>3! ...*L.Wn
0040  76 f6 38 61 87 07 58 f8  96 e9 71 0e 15 42 81 ce   v.8a..X. ..q..B..
0050  4d d3 dc 95 1f 96 7d 37  a9 19 8d ce 29 9a df a1   M.....}7 ....)...
0060  90 42 c3 ce 9b 7c d0 16  ab 8a f7 74 85 88 d7 3e   .B...|.. ...t...>
0070  2e 91 b7 23 c5 6a e8 a9  14 ea                     ...#.j.. ..

0000  00 0c 76 40 89 cc 00 11  d8 b8 46 31 08 00 45 00   ..v@.... ..F1..E.
0010  00 6c 11 77 00 00 80 11  a3 9e c0 a8 02 0b c0 a8   .l.w.... ........
0020  02 10 7f 4d 7f 4d 00 58  9e 4b 02 00 00 00 a8 5b   ...M.M.X .K.....[
0030  13 66 59 f7 2e 07 95 5d  56 d9 8f 22 62 d8 f8 3a   .fY....] V.."b..:
0040  9d 4b d0 c6 3e ad 45 34  19 f6 5e 38 f8 55 b0 14   .K..>.E4 ..^8.U..
0050  04 c3 1e 17 0a a6 de 37  94 fb 1c 58 fd 66 4e 5d   .......7 ...X.fN]
0060  a8 04 2f 3b 55 f7 ed 04  c0 0b 1c f5 59 69 2d cd   ../;U... ....Yi-.
0070  cd d4 8b 89 22 8e 62 29  48 d8                     ....".b) H.
```

Fig. 10. Encrypted ICMP sequence

The same plaintext from the source messages is encrypted to two different cryptograms in encrypted messages.

7 Conclusion

This article has introduced the new secure communication architecture for IPv4 with NAT traversal. Key benefits it brings are ability to interconnect to any host behind NAT using just a host's domain name, and encryption and authentication of all data transmitted in this connection. Secure channel is established on-demand automatically and is independent on any application. Applications do not need to care of encryption of the data they transmit over the network.

The architecture is based on a packet processor which captures packets to encrypt, authenticates and sends over the UDP to the destination. The DNS database is used for the global naming service of clients behind NAT. It assigns these clients a special IP address so the packet processor can recognize that the host it needs to connect to is behind NAT and therefore the process requires the UDP hole punching to establish a LAN-LAN channel.

Performance of the packet processor has been measured and the results are showing it to be as fast as IPsec implementation in Microsoft Windows XP Professional. The future releases are expected to be faster than IPsec because of kernel-level implementation, lower protocol overhead and optimization of the source code.

Every user of this system is reachable by a single DNS name from any public or private network in the world which strongly simplifies his mobility.

Acknowledgement: *This article has been sponsored by the Ministry of Industry and Trade of the Czech Republic, project no. FT-TA3/011.*

8 References

[1] Fergusson, N., Schneier, B., *Practical Cryptography*, Wiley Publishing, Inc., Indianopolis USA, 2003

[2] A. J. Menzes, P. C. van Oorschot, S. A. Vanstone, *Handbook of applied cryptography*, CRC Press LLC, Florida, USA, 1997.

[3] D. Kegel, “NAT and Peer-to-peer networking”, *Web page*, http://alumnus.caltech.edu/~dank/peer-nat.html. 1999.

[4] B. Ford, P. Srisuresh, and D. Kegel, “Peer-to-Peer Communication Across Network Address Translators”, *Web page*, http://www.brynosaurus.com/pub/net/p2pnat/ 2005.

[5] S. Kent, R. Atkinson, “Security Architecture for the Internet Protocol”, *RFC 2401*, 1998.

[6] T. Dierks, C. Allen, “The TLS Protocol Version 1.0”, *RFC 2246*, 1999.

[7] OpenVPN project, http://openvpn.sourceforge.net

[8] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", *RFC 2104*, 1997.

[9] FreeS/WAN project, http://www.freeswan.org

[10] S. Kent, R. Atkinson, “IP Encapsulating Security Payload (ESP)”, *RFC 2406*, 1998.

[11] L. Cvrk, V. Zeman, D. Komosny, “H.323 Client-Independent Security Approach”. *Lecture Notes in Computer Science*, 2005.

[12] S. Kent, and R. Atkinson, “IP Encapsulating Security Payload (ESP)”, *RFC 2406*, 1998.