Global Information Assurance Certification Paper

Encryption
Methods for Key Distribution and Algorithms

Nicolás Severino
June 3rd, 2003
SANS GSEC Practical Assignment

Encryption: Methods for Key Distribution and Algorithms

Nicolás A. Severino
June 3rd, 2003
SANS GSEC Practical Assignment v.1.4b

Abstract

Encryption is crucial to ensure the distribution of information securely in today's connected world. The level of expertise needed to perpetrate attacks is decreasing, the information and tools available to generate these attacks is growing, and the combination of these two factors can certainly jeopardize the integrity of the information if it is being distributed enciphered.

This document reviews the different aspects of encryption, goes through the different implementations of key distribution analyzing pros and cons, we will describe the different algorithms and it relation to the level of security they provide, and what does the key length represent. Then, well analyze a real life implementation and finish explaining why is so important not to choose only one method.

Introduction

There was a time in which the enterprise client was a bunch of dumb character based terminals connected to a mainframe. Then, the PC revolution came and the "powerful" x386 running DOS or –in early adopter companies– Microsoft Windows (pick your version of preference as soon as it is below 3.1) ruled the enterprise space. Novell was the beating heart of this era, which changed again in the late nineties with Windows NT 4.0.

Nowadays, the enterprise information systems are fully connected to the Internet, have dozens of applications installed and handle critical and sensitive information for the company at the desktop level. This adds a significant degree of complexity when securing the perimeter and raise the inevitable question: Is it possible to monitor and control all the data going in and out through the communication channel? Is that channel secure enough? Is the information safe from being compromised? From Julius Caesar Cipher [1] to the modern AES, security specialists have been pursuing stronger and faster encryption methods. The information that is being protected can be used against individuals or groups and Industrial espionage is also a concern among highly competitive business. As a result of the evolution of the technology, more intrusive the new threats are and then more compromised the confidentiality, integrity and availability gets. And this has a direct impact in the administration, deployment and maintenance of the enterprise security.

Encryption Export Regulations

Before we jump into the different algorithms and methods of encryption, I do not want to forget to mention that there are specific regulations on encryption export outside the U.S. This is regulated by the Bureau of Exporting Affairs [2] and basically this organization determines if a product that encrypts information can be sold outside the U.S. (if yes they also define where), and this depends not only on the level of encryption but also on the functionality and nature of the hardware/software that would be actually shipped. Until 1996, anything stronger than 40-bit encryption not approved for export. Now, it is possible to export 56-bit encryption, and 128-bit with some restrictions (as it is the new digital standard for encryption).

Methods of Encrypting Data

When thinking about encryption of data, we think about a cumbersome process of transforming the information into several characters that must not make any sense for the occasional reader: the potential interceptor of that information. As this transformation process can be implemented using software, the next challenge is to create the process in such a way that the encryption/decryption does not consume much resource, but is strong enough to avoid code breaking. One example of this is the 'PKZIP®' utility, which offers both compression AND data encryption [3], which is accomplished by using an encryption key to alter the compressed data. The key is not stored in the file and is needed for decryption of the information.

Pursuing speed and performance, we encounter the translation tables as a quick method as it does not need any heavy calculation other than checking the 'translation table', and/or the resulting 'translated' value from within the table is then written into the output stream. In fact, the instruction 'XLAT' was included in 80x86 CPU's. The 'XLAT' instruction changes information between tables at the hardware level [4]. The main risk with this method is that once that the table has been discovered then the whole set of encryption messages done (now and in the past) can be decrypted. In order to avoid this the method can become more complex by adding a second (or a third or more) tables to encrypt/decrypt. If the use of this tables is based on the premise of utilization in a ‘pseudo-random’ order then the code breaking gets relatively more difficult. A good example of this method is the computer of the movie 2001: A Space Odyssey, whose name was HAL. If you use a translation table that has an offset of one letter (taking as a reference the English alphabet) then you will find out that “HAL” means “IBM”. As you can see (and despite the fact that this is a simple and naïve example) this method is far from secure, but is important to mention it as it was used for centuries until new methods came to the table.

Adding complexity is 'data repositioning'. It takes more processing time than 'translation table' but it is more secure (specially when used in combination with other methods). The way this works is by 'reorganizing' the order of bytes and written. On the other end the inverse process is done at destination [5]. As an example, you could take the sentence "This text will be encrypted" and the outcome could be "Tx rhtbyi epsw tieetlndelc". Again, this method (although more secure than translation tables) is still far from secure.

Last (but not least) we have word/byte rotation and XOR bit masking, a method that rotates words following variable direction and duration for that rotation [6]. It uses a cyclic redundancy check as a checksum in this method. What this method does is take the original information and start a cycle of reorganizing the words and bytes in such a way that after each rotation the output matrix is different than the input making sure that the operation is reversible. Basically this is done once and again in so called "rounds" and as we will see through this paper, not only different ciphers use a different number of rounds but also the same cipher could have multiple numbers of rounds depending on the key length being used. A good example of this is AES, which iterates from 9 to 13 times (depending on the key length) changing the bytes in order to generate the resulting encrypted message.

Public and Symmetric Encryption Methods

As of today, encryption methods are based in keys. The key is the password used to encrypt/decrypt the message. And there are two different implementations for this: Symmetric and Public Encryption. The first one is a good option when key distribution is not an issue, as the same key is used for encryption and decryption operations. Then, Public Encryption came to solve the problem of key distribution [7], based on the assumption that it is possible to cipher using the one key (i.e. the public key), but that is not possible to decrypt using that same key (i.e. need to use the private). But let's take a closer look to each one of these methods:

Symmetric Key

As its name states, Symmetric Key encryption (also referred to as conventional cryptography) [8] uses the same key to encrypt and decrypt information. As a result, the success of the coding process relies on the encryption cipher ability to use stronger keys at a higher speed. In fact, if we analyze the different algorithms that use symmetric keys, we will find that there is a point in which there are iterations (also called rounds) that permute once and again the information with the objective of generate the actual encryption. Additionally, all the users that need to share the information need to have the same key. Then it could be a little bit of a problem to distribute in a safe way all the keys needed, as if only one key gets compromised, all the other ones need to be replaced.

The strength of this method relies in how securely we could maintain the key: The longer not always means the safer. And it is important to notice that despite the fact that we can

generate a key of the length that we want, it does not assure 100% that it would not be broken: First because as we pick a longer key some issues related to performance, efficiency, cost (distribution) may arise in the horizon; and second because we can not predict the future computational power, and its ability to brake the key. However, it is possible to work balancing computational power available today, predictions on time/effort required to brake a given length key and impact on performance in the real world.

Finally, the solution to the "key" problem does not reside in the key itself, but in the method for a safe distribution. Combining it with other keys (using public encryption method for example) to encrypt it and then transmit it to the desired user is the best solution for this problem. It not only ensures that the key gets to the right people, but also helps to prevents it from being intercepted by attackers. [9]

Disadvantages	Advantages
It relies on the communication channel to distribute the key	Relatively small and fast algorithms
Not effective for authentication	Low memory use
Not effective for non-repudiation [10]	There are hardware implementations that are faster than software implementations
The strength of this method relies in the key	Algorithms are permutation based with multiple rounds (more strength)
If only one key gets compromised, all the other ones need to be replaced	

Well known implementations of Symmetric Key algorithms are DES (Data Encryption Standard), 3-DES (triple DES), IDEA, RC5, Blowfish, and AES (Advanced Encryption Standard). AES was designated as the de facto encryption algorithm replacing DES, but we'll discuss this later on this paper.

Public Key

Martin Hellman and Whitfield Diffie developed this method in the mid 1970s. It is said that the British Secret Service invented this method several years ago, but they kept it as a secret [11]. This is an asymmetric encryption method, as it uses two different keys (one public and one private) to encrypt and decrypt the information.

NOTE: Although we will talk through this paper about encrypt and decrypt, there are several other implementations that exist around encryption as digital certificates, signatures and hashes.

These two keys are generated one for the public (public key) and another for personal use (private key). Depending on the implementation that you are running, one or the other could be used to decrypt or verify the information received. For example:

You encrypt with someone's public key - that person decrypts with his private key. You sign a message with your private key – the intended recipient/s check the signature with your public key.

The beauty of this method is that is computationally unfeasible to infer the private key from the public, ensuring that the information is safe to travel through an unsecured channel (despite the fact that the public encryption key is available to the public). Having said this, it is clear that the main problem that this method solves is the issue of distributing a secret key via an unsecured channel, as no private key is transmitted nor distributed. Additionally it has the advantage of being cheaper as there is no need for a secure channel that could be used for information or key distribution.

Disadvantages	Advantages
Slower than Symmetric Key Algorithms	It is computationally unfeasible to infer the private key from the public
Require larger keys than Symmetric Key Algorithms (resource intensive)	Solves is the issue of distributing a secret key via an unsecured channel
User needs to keep integrity of the private key (you lost the key, you lost all messages currently and previously encrypted using that key)	It is possible to create digital signatures
Hard to prove the authenticity of a key's origin [12]	

There are several algorithms that implement the public key encryption methodology, such as RSA, Diffie-Hellman, and El-Gamal.

The Key Length Issue

As the core component of encryption, the algorithms have a crucial role in the process. Basically, it is not possible to encrypt without them. Since the very first days of permutation, the biggest challenge was how to create a complex; fast, reliable algorithm that could keep our information away from estranges eyes. Nowadays, algorithms use a "key" to encrypt and decrypt the information, and there is a direct relation between the length of the key and the computing resources required to crack the code. But before we go into the different algorithms available, let's review some facts about key length and possible combinations (tied to computing resources required):

- A 56-bit key creates 72 quadrillion possible combinations
- A 128-bit key creates 4.7 sextillion possible combinations
- It took 1 month (back in 1999) to crack a 140-bit RSA Key [13]
- It took 7.4 months (1999 - 2000) to crack a 512-bit RSA Key [13]

Having said this, it sounds pretty clear that, although a 512-bit key may be considered adequate for RSA encryption [z], 1024-bit keys are the ones that can really push back attackers by adding complexity and computing cycles to potential brute force intents, however there is a paper that describes how to built a machine that could break a 1024-bit key in minutes [14]. But if we think in actual hardware, with the processing power available today a 56-bit key is considered crack able, a 128-bit key is not [15], (with ordinary equipment and reasonable effort).

Depending on the Encryption Method that we decide to use, we will approach the key strength issue from different sides: When we talk about Public Encryption we should take under consideration the length topic, as it is always preferred to have a stronger public/private key to cipher the information. On the other hand, if we are talking about Symmetric Encryption, the focus should be in the method of distribution (see the "Symmetric Key" section within "Public and Symmetric Encryption Methods") above.

RSA and Diffie-Hellman

RSA

This public-key cryptosystem is available since 1977, and offers both encryption and signatures. In real life we will see it working with DES or any other symmetric key algorithm. Additionally is used for authentication (leaving a digital fingerprint that can be validated later). The hash utilized by RSA creates the public and private keys based in mathematic operations using prime numbers. The assumption is that factoring is difficult (and then, it is difficult to get the private key from the public key). In order to encrypt the message, the only key needed is the public key of the recipient of that message, who will decrypt the message using his own private key [16].

This is not a fast algorithm, and by comparison DES can be up to 100 times faster if we are talking about a software implementation (hardware implementations of DES are indeed faster). However this algorithm is widely used (for example in financial institutions) and has became a de facto standard in digital signatures.

Diffie-Hellman

This protocol is used mainly for key negotiation. It was created in 1976, and it "permits the exchange of a secret key over an insecure medium without any prior secrets" [17].

The protocol works by sending public values generated by each party, then they compute the received value with theirs, and both get to the same result. That result is the shared secret that both parties share now. The problem encountered in this process, is that the protocol is vulnerable to an attack (man-in-the-middle attack) in which if someone intercepts the first value exchanged between both parties, he can gain access to the

encrypted information that start to flow from one to the other, as he will be able to generate the shared secret using the values intercepted. In order to solve this problem, digital signature has been added to the exchange of information, allowing them to check for the authenticity of the exchange of values (the interceptor may still be able to intercept the message, but will not be able to reproduce the signature)

DES, 3-DES, AES and IDEA

DES and 3-DES

Originally developed to be embedded in hardware, DES is the one that has been used and observed for the longest time: It started to be used in the late 1970s as a result of the effort between IBM and the NSA. It uses a 56-bit key (which at the end brings the block size to 64-bit with the addition of 8 bits intended for parity functions). The way the cipher works is split in two, then applies the round function to only one of the two parts and then combines it with the other part (XOR), and this continues up to 16 rounds [18]. Nowadays, DES is no longer considered secure (see "The Key Length Issue"), as it is easily crack able with the commercial computing power available today.

In order to provide a response to the DES security problem, 3-DES is actually considered a more secure algorithm. What 3-DES does is to run the encryption three times. If we translate this into computing power, you could say that (just theory math) 3-DES take 2.5 times more CPU power than DES. We should keep in mind that DES uses a 56-bit key while 3-DES uses a 168-bit key.

AES

The Advance Encryption Standard replaces DES as the standard for governmental use. It is the new Federal Information Processing Standard (FIPS-197) algorithm. The process of selecting this algorithm started back in 1997 and finished in November 2001.The main objective for this new algorithm was the strength and ease of implementation. The algorithm is public and unclassified, and was designed as a symmetric block cipher using a minimum of 128-bit input blocks and supports 3 key sizes of 128, 192 and 256 bits.

The selected cipher was Rijndael, out of a group of fives that included MARS, RC6, Serpent and Twofish. AES works as a cipher that has a variable block and key length that iterates. This means that the plain text information gets transformed once and again multiple times before producing the output (if a 128-bit key is being used then there are nine rounds, if a 192-bit key is being used then there are eleven rounds, and if a 256-bit key is being used there are thirteen rounds [19]). Although Rijndael permits multiple combinations of key and block lengths, the AES implementation only contains some of them [20].

But, how secure is AES? What if an AES Cracker hits the streets? Well, assuming that you could break a DES key in one second, then it would take to that same machine approximately 149 thousand billion years to crack a 128-bit key.

As stated above, there were other four finalists for AES, and Rijndael was selected based not only on being considered the most secure, but also primarily on its low memory requirements, efficiency, performance, flexibility and ease of implementation [21].

IDEA

As a result of the difficulties that laid ahead in the export of encryption outside of the United States, there was a need (that grew more and more as time passed and the available computational power for commercial equipment was more than qualified to brake the 56-bit DES key) for a stronger cipher that could be used outside the US. Using RSA was too slow if we compare it with other algorithms that were as secure as it is, and then the idea of switching to another encryption schema after the initial authentication seemed to be the solution. But were to jump?

Created by Xuejia Lai and James Massey of the Swiss Federal Institute of Technology, this block cipher uses a 128-bit key length (twice as long as DES) and it goes up to 8 rounds in the cipher, to provide strong enough encryption for some time into the future. As it uses Symmetric Encryption, it provides security based upon the ignorance of the secret key, as the algorithm is public. Additionally, it has no export limitations and can be implemented in hardware.

In order to measure the level of effectiveness, we could analyze the resulting information out of the cipher and we would not be able to find references to the original plain text message, making it as robust as needed in terms of effectiveness. One topic that is pointed out more than once is the speed and the ability of IDEA to work faster than other algorithms (even DES). So, lets take IDEA and compare its performance with a well-known CPU chip: Any Pentium MMX compatible or Pentium II. We will then find that is faster not only than DES, but also faster than RC5 or Blowfish encryption [22].

As a result of being stable, still unbreakable (although there were several the attempts to brake it) and widely accepted, IDEA was entered in several standards as follows: ISO 9979/002: ISO Register of Cryptographic Algorithms, UN/EDIFACT: EDIFACT Security Implementation Guidelines, ITU-T Recommendation H.233: Confidentiality System for Audiovisual Services, IETF RFC 3058: Use of the IDEA Encryption Algorithm in CMS, TBSS: Swiss Telebanking Security Standard, OpenSSL Cryptographic Librar, WAP Wireless Transport Layer Security.

Pretty Good Privacy

PGP

All of the different cryptosystems that we have analyzed use whether Public Key or Symmetric Key Encryption. Then, we have PGP. PGP (from Pretty Good Privacy) was a created by Philip Zimmermann in 1991; it combines RSA and IDEA, resulting in an excellent implementation that made it the most massive and popular of all methods. Nowadays, it is easy to find plug-ins for popular email programs such as Outlook Express or Eudora [23]. And the reason for this is quite simple: With PGP, the information (email in this case) will be protected, is easy to use, and is fast.

The way PGP works is interesting, because it combines Public Key and Symmetric Key Encryption in order to get the best of both worlds as follows: It takes the plain text information, and cipher it using IDEA (in the beginning a symmetric key cipher called Bass-O-Matic was used, but this was dropped in PGP v2 as this encryption cipher was weak). Is important to notice that this first encryption is accomplished by generating a one-time session key [24]. So at this point, we have the text encrypted. As the next step, using the receiving person's public key it encrypts the session key, and now the encryption is complete: The receiving end gets both encrypted pieces of information, so it decrypts the session key first (using its private key) and now that is has the session key it decrypts the message.

To cite an example of PGP and the traction that this cryptosystems gain in the Internet mail community, we can talk about OpenPGP. Its implementation relays in two RFCs (RFC 1991, PGP Message Exchange Formats and RFC 2015, MIME Security with Pretty Good Privacy) and its pretty solid, but it has some things that may prevent it to be adopted as a standard. The first think would be that S/MIME (that already had 3 revisions) is widely accepted with also defined extensions such as secure mailing lists and security labels among others [25]. The second, OpenPGP RFCs were not accepted yet (RPC 2015 is a proposed standard) and what may be preventing this from happening is that is based in RSA key exchange and IDEA encryption, being both patented. But despite the fact that OpenPGP may not became a IETF standard is being considered seriously and has a very consistent implementation that not only works and is efficient, but also that is accepted by the Internet mail community (and developers) today.

Secure Electronic Transaction

As we try to analyze an implementation in the real world that resumes all the information discussed along this paper, it is important to notice that it will be found most commercial software packages taking advantage of each Key Method (Public and Symmetric) strength and overcoming its weaknesses. As an example we could cite the Secure

Electronic Transaction (SET) protocol [26]. MasterCard and Visa announced SET back in February 1996.

It is clear that the main objective of this implementation is to avoid any leak of information in the flow of data between the sender and the recipient. Having in mind the nature of the http protocol (that by design is not intended to persist on each connection) we can immediately infer the risk associated with commercial transactions in this hostile environment [27].

But which are the gaps and what they mean in terms of security? When we talk about commercial transactions we are talking about money, and it is obvious that no one wants to see his money in the line of fire: It needs to be safe, and get to the hands of the right people. No one wants his credit card number and payment information to travel "light" through the Internet. This leads to the first goal accomplished by SET: *Confidentiality of the information transmitted*. The way this is accomplished is by doing the following:

1. Encrypt the message using the Symmetric Key Method
2. Encrypt the key generated in step 1 using the receiver's Public key
3. The encrypted message gets to the receiver recipient
4. The encrypted key gets to the receiver recipient
5. The receiver decrypts the key using his private key
6. The receiver decrypts the message using the decrypted key

When we send all the information about payment method and so on, the next step in the transaction is to receive a confirmation on that. And it is important the buyer can check the authenticity of that confirmation: *Integrity of data* is the second goal. Using a hash combined with its private key, the sender produces a 160-bit message digest and this signature is appended to the message, so then the receiver could check it with the sender's public key. If the contents of the message were altered in the transit, the end party will know, as the signature will not be able to be confirmed.

But at this point, we checked that the information could travel safe, hidden within a cipher; and that the messages interchanged were unaltered. But about checking if the person that is selling/buying is the person that he/she says he/she is? And we are in the third goal accomplished by SET: *Party Authentication*. Using a combination of x.509v3 certificates and RSA signatures, it is possible to confirm the identity of each end by the other one.

In resume, combining both (Public and Symmetric) methods, the information gets to the right place in shape, securing not only the distribution of the message, but also the distribution of the key. The preservation of the Confidentiality of information, the Integrity of data, the Cardholder account authentication and Merchant authentication are the goals accomplished, making this implementation a great example of the good use and application of encryption.

Conclusion

When we talk about encryption, we are talking about protecting our information from being exposed to a non-intended audience. We are talking about securing the most valuable asset that an organization has. And by doing that, we are not only making sure that the information gets delivered to the intended recipient, but also that in the process of being sent it does not get cracked nor stole.

After discussing the two implementations for key distribution, it's clear that both have its pros and cons. As we pursue the best implementation by assuring the confidentiality, integrity and availability of the information, it is very important to keep in mind that a combination of both represents the best alternative.

By combining the right methods, choosing the appropriate algorithms along with the right key length (remember that not always longer means better) and a hash algorithm that provides a signature that can alert on tampering, we will be able to implement encryption at the corporate level in such a way that cost, performance and efficiency are balanced.

References

[1] "Beginners' Guide to Cryptography". April 1 2003. URL: http://www.murky.org/cryptography/classical/caesar.shtml (June 12 2003)

[2] DEPARTMENT OF COMMERCE, Bureau of Export Administration. "BXA Encryption Export Regulations, Jan. 12, 2000". January 10 2000. URL: http://www.eff.org/Privacy/ITAR_export/2000_export_policy/20000112_cryptoexport_regs.html (June 12 2003)

[3] Frazier, R. E. "Data Encryption Techniques". August 21 2002. URL: http://www.mrp3.com/encrypt.html (June 12 2003)

[4] "80386 Programmer's Reference Manual -- Opcode XLAT". URL: http://www.itis.mn.it/linux/quarta/x86/xlat.htm (June 12 2003)

[5] CyberWorld. "How To Encrypt Data". URL: http://powow.com/india/encrypt.htm (June 12 2003)

[6] Kay, H. & Neufeld, L. "Methods of Encrypting Data". Encryption. July 25 1999 URL: http://oscar.cprost.sfu.ca/~isp253/99-2/gagne/Encryption.htm (June 12 2003)

[7] Litterio, Francis. "Cryptography". URL: http://world.std.com/~franL/crypto/ (June 12 2003)

[8] "Introduction to Encryption: What is cryptography?". URL: http://waubonsie.com/security/intro.html (June 12 2003)

[9] Barkley, John. "Secret Key Distribution". X.400 Message Handling Services. October 7 1994. URL http://csrc.nist.gov/publications/nistpubs/800-7/node209.html#SECTION08321100000000000000 (June 12 2003)

[10] Johnston, R.B. "Trading Systems and Electronic Commerce" 1998. URL: http://www.dis.unimelb.edu.au/staff/jcarroll/252/encryption.pdf (June 12 2003)

[11] Ellis, J H. "The Possibility of Secure Non-Secret Digital Encryption". CESG Report. January 1970

[12] Sobell, Mark. "Linux Security: Public Key and Symmetric Encryption". April 17 2003. URL: http://www.net-security.org/article.php?id=489 (June 12 2003)

[13] Edwards, Mark Joseph. "Choosing an Encryption Key Length". Sept 1 1999. URL: http://www.ntsecurity.net/articles/index.cfm?articleID=7164&pg=1&show=648 (June 12 2003)

[14] Schneier, Bruce. "Crypto Gram Newsletter". March 15 2002. URL: http://www.counterpane.com/crypto-gram-0203.html#6 (June 12 2003)

[15] Froomkin, Dan. “Deciphering Encryption”. May 8 1998. URL: http://www.washingtonpost.com/wp-srv/politics/special/encryption/encryption.htm (June 12 2003)

[16] RSA Laboratories. “What is the RSA cryptosystem?”. RSA Laboratories' Frequently Asked Questions About Today's Cryptography, Version 4.1. 2000. URL: http://www.rsasecurity.com/rsalabs/faq/3-1-1.html (June 12 2003)

[17] “What is Diffie-Hellman” URL: http://www.hack.gr/users/dij/crypto/overview/diffie.html (June 12 2003)

[18] Ocean Logic Pty Ltd. “OL_DES DES Cryptoprocessor” URL http://www.ocean-logic.com/pub/OL_DES.pdf (June 12 2003)

[19] Savard, John J. G. “The Advanced Encryption Standard (Rijndael)”. A Cryptographic Compendium.URL: http://home.ecn.ab.ca/~jsavard/crypto/co040401.htm (June 12 2003)

[20] SignalGuard International Ltd. “The Advanced Encryption Standard - AES/Rijndael”.URL: http://www.signalguard.com/encryption/aes-rijndael.htm (June 12 2003)

[21] Ford, Susan. “Advanced Encryption Standard (AES) - Questions and Answers”. October 2 2000. URL: http://www.nist.gov/public_affairs/releases/aesq&a.htm (June 12 2003)

[22] Lipmaa, Helger. “Fast IDEA for Pentium MMX compatibles”. February 28 1998. URL: http://home.cyber.ee/helger/implementations/fastidea/ (June 12 2003)

[23] Greene, Anthony E. “A Newcomer's Introduction to Pretty Good Privacy (PGP)”. URL: http://www.mindspring.com/~aegreene/pgp/ (June 12 2003)

[24] Senderek, Ralf. “The Protection of Your Secret Key”. January 2001. URL: http://senderek.de/security/secret-key.protection.html (June 12 2003)

[25] “Enhanced Security Services for S/MIME” Request for Comments: 2634. June 1999. URL http://www.imc.org/rfc2634 (June 12 2003)

[26] Tall, Eric and Ginsburg, Mark. “Microsoft Internet Security Initiatives”. LATE NIGHT ACTIVEX. October 1996. URL: http://sunsite.iisc.ernet.in/virlib/activex/latenight/ch8.htm (June 12 2003)

[27] IBM. “How SET Works”. URL: http://www-3.ibm.com/software/genservers/commerce/payment/support/works/workss.html (June 12 2003)