94

Methods of Attacking and Defending Cryptosystems

Joost Houwen

AU7495—Chapter94—25/1/2007—21:14—PARTHIBAN—14754—XML MODEL CRC12a – pp. 1255–1269.

Encryption technologies have been used for thousands of years and, thus, being able read the secrets they are protecting has always been of great interest. As the value of our secrets have increased, so have the technological innovations used to protect them. One of the key goals of those who want to keep secrets is to keep ahead of techniques used by their attackers. For today's IT systems, there is increased interest in safeguarding company and personal information, and therefore the use of cryptography is growing. Many software vendors have responded to these demands and are providing encryption functions, software, and hardware. Unfortunately, many of these products may not be providing the protection that the vendors are claiming or customers are expecting. Also, as with most crypto usage throughout history, people tend to defeat much of the protection afforded by the technology through misuse or inappropriate use. Therefore, the use of cryptography must be appropriate to the required goals and this strategy must be constantly reassessed. To use cryptography correctly, the weaknesses of systems must be understood.

This chapter reviews various historical, theoretical, and modern methods of attacking cryptographic systems. Although some technical discussion is provided, this chapter is intended for a general information technology and security audience.

94.1 Cryptography Overview

A brief overview of definitions and basic concepts is in order at this point. Generally, *cryptography* refers to the study of the techniques and methods used to hide data, and *encryption* is the process of disguising a message so that its meaning is not obvious. Similarly, decryption is the reverse process of encryption. The original data is called *cleartext* or *plaintext*, and the encrypted data is called *ciphertext*. Sometimes, the words *encode/encipher* and *decode/decipher* are used in the place of *encrypt* and *decrypt*. A cryptographic algorithm is commonly called a *cipher*. *Cryptanalysis* is the science of breaking cryptography, thereby gaining knowledge about the plaintext. The amount of work required to break an encrypted message or mechanism is call the *work factor*. *Cryptology* refers to the combined disciplines of cryptography and cryptanalysis.

Cryptography is one of the tools used in information security to assist in ensuring the primary goals of confidentiality, integrity, authentication, and non-repudiation.

Some of the things a cryptanalyst needs to be successful are:

- Enough ciphertext
- Full or partial plaintext
- Known algorithm
- Strong mathematical background
- Creativity
- Time, time, and more time for analysis
- Large amounts of computing power

Motivations for a cryptanalyst to attack a cryptosystem include:

- Financial gain, including credit card and banking information
- Political or espionage
- Interception or modification of e-mail
- Covering up another attack
- Revenge
- Embarrassment of vendor (potentially to get them to fix problems)
- Peer or open-source review
- Fun/education (cryptographers learn from others' and their own mistakes)

It is important to review the basic types of commonly used ciphers and some historical examples of cryptosystems. The reader is strongly encouraged to review cryptography books, but especially Bruce Schneier's essential *Applied Cryptography*[1] and *Cryptography and Network Security*[2] by William Stallings.

94.2 Cipher Types

94.2.1 Substitution Ciphers

A simple yet highly effective technique for hiding text is the use of substitution cipher, where each character is switched with another. There are several of these types of ciphers with which the reader should be familiar.

94.2.1.1 Monoalphabetic Ciphers

One way to create a substitution cipher is to switch around the alphabet used in the plaintext message. This could involve shifting the alphabet used by a few positions or something more complex. Perhaps the most famous example of such a cipher is the Caesar cipher, used by Julius Caesar to send secret messages. This cipher involves shifting each letter in the alphabet by three positions, so that "A" becomes "D," and "B" is replaced by "E," etc. Although this may seems simple today, it is believed to have been very successful in ancient Rome. This is probably due, in large part, to the fact the even the ability to read was uncommon, and therefore writing was probably a code in itself.

A more modern example of the use of this type of cipher is the UNIX *crypt* utility, which uses the ROT13 algorithm. ROT13 shifts the alphabet 13 places, so that "A" is replaced by "N," "B" by "M," etc. Obviously, this cipher provides little protection and is mostly used for obscuration rather than encryption, although with a utility named *crypt*, some users may assume there is actually some real protection in place. Note that this utility should not be confused with the UNIX *crypt()* software routine that is used in the encryption of passwords in the password file. This routine uses the repeated application of the DES algorithm to make decrypting these passwords extremely difficult.[3]

94.2.1.2 Polyalphabetic Ciphers

By using more than one substitution cipher (alphabet), one can obtain improved protection from a frequency analysis attack. These types of ciphers were successfully used in the American Civil War[4] and have been used in commercial word-processing software. Another example of this type of cipher is the Vigenère cipher, which uses 26 Caesar ciphers that are shifted. This cipher is interesting as well because it uses a keyword to encode and decode the text.

94.2.2 One-Time Pad

In 1917, Joseph Mauborgne and Gilbert Vernam invented the unbreakable cipher called a one-time pad. The concept is quite effective, yet really simple. Using a random set of characters as long as the message, it is possible to generate ciphertext that is also random and therefore unbreakable even by brute-force attacks. In practice, having—and protecting—shared suitably random data is difficult to manage but this technique has been successfully used for a variety of applications. It should be understood by the reader that a true, and thus unbreakable, one-time pad encryption scheme is essentially a theoretical concept as it is dependent on true random data, which is very difficult to obtain.

[1]Schneier, Bruce. 1995. *Applied Cryptography*, p.19, John Wiley, New York.

[2]Stallings, William. 2002. *Cryptography and Network Security: Principles and Practice*, p. 19, Prentice-Hall, Englewood Cliffs.

[3]Spafford. 2003. *Practical UNIX and Internet Security*, p. 19, O'Reilly & Associates, Sebastapol, CA.

[4]Schneier, Bruce. 1995. *Applied Cryptography*, p. 11, John Wiley, New York.

94.2.3 Transposition Cipher

This technique generates ciphertext by performing some form of permutation on plaintext characters. One example of this technique is to arrange the plaintext into a matrix and perform permutations on the columns. The effectiveness of this technique is greatly enhanced by applying it multiple times.

94.2.4 Stream Cipher

When large amounts of data need to enciphered, a cipher must be used multiple times. To efficiently encode this data, a stream is required. A stream cipher uses a secret key and then accepts a stream of plaintext producing the required ciphertext.

94.2.4.1 Rotor Machines

Large numbers of computations using ciphers can be time-consuming and prone to errors. Therefore, in the 1920s, mechanical devices called rotors were developed. The rotors were mechanical wheels that performed the required substitutions automatically. One example of a rotor machine is the Enigma used by the Germans during World War II. The initial designs used three rotors and an operator plugboard. After the early models were broken by Polish cryptanalysts, the Germans improved the system only to have it broken by the British.

94.2.4.2 RC4

Another popular stream cipher is the Rivest Cipher #4 (RC4) developed by Ron Rivest for RSA.

94.2.5 Block Cipher

A block cipher takes a block of plaintext, a key, and produces a block of ciphertext. Current block ciphers produce ciphertext blocks that are the same size as the corresponding plaintext block.

94.2.5.1 DES

The Data Encryption Standard (DES) was developed by IBM for the National Institute of Standards and Technology (NIST) as Federal Information Processing Standard (FIPS) 46. Data is encrypted using a 56-bit key and 8 parity bits with 64-bit blocks.

94.2.5.2 3DES

To improve the strength of DES-encrypted data, the algorithm can be applied in the triple-DES form. In this algorithm, the DES algorithm is applied three times, either using two keys (112-bit) encrypt–decrypt–encrypt, or using three keys (168-bit) encrypt–encrypt–encrypt modes. Both forms of 3DES are considered much stronger than single DES. There have been no reports of breaking 3DES.

94.2.5.3 IDEA

The International Data Encryption Algorithm (IDEA) is another block cipher developed in Europe. This algorithm uses 128-bit keys to encrypt 64-bit data blocks. IDEA is used in Pretty Good Privacy (PGP) for data encryption.

94.3 Types of Keys

Most algorithms use some form of secret key to perform encryption functions. There are some differences in these keys that should be discussed.

1. *Private/Symmetric*. A private, or symmetric, key is a secret key that is shared between the sender and receiver of the messages. This key is usually the only key that can decipher the message.

2. *Public/Asymmetric*. A public, or asymmetric, key is one that is made publicly available and can be used to encrypt data that only the holder of the uniquely and mathematically related private key can decrypt.
3. *Data/Session*. A symmetric key, which may or may not be random or reused, is used for encrypting data. This key is often negotiated using standard protocols or sent in a protected manner using secret public or private keys.
4. *Key Encrypting*. Keys that are used to protect data encrypting keys. These keys are usually used only for key updates and not data encryption.
5. *Split Keys*. To protect against intentional or unintentional key disclosure, it is possible to create and distribute parts of larger keys which only together can be used for encryption or decryption.

94.4 Symmetric Key Cryptography

Symmetric key cryptography refers to the use of a shared secret key that is used to encrypt and decrypt the plaintext. Hence, this method is sometimes referred to as secret key cryptography. In practice, this method is obviously dependent on the "secret" remaining so. In most cases, there needs to be a way that new and updated secret keys can be transferred. Some examples of symmetric key cryptography include DES, IDEA, and RC4.

94.5 Asymmetric Key Cryptography

Asymmetric key cryptography refers to the use of public and private key pairs, and hence this method is commonly referred to as public key encryption. The public and private keys are mathematically related so that only the private key can be used to decrypt data encrypted with the public key. The public key can also be used to validate cryptographic signatures generated using the corresponding private key.

94.5.1 Examples of Public Key Cryptography

94.5.1.1 RSA

This algorithm was named after its inventors, Ron Rivest, Adi Shamir, and Leonard Adleman, and based on the difficulty in factoring large prime numbers. RSA is currently the most popular public key encryption algorithm and has been extensively cryptanalyzed. The algorithm can be used for both data encryption and digital signatures.

94.5.1.2 Elliptic Curve Cryptography (ECC)

ECC utilizes the unique mathematical properties of elliptic curves to generate a unique key pair. To break the ECC cryptography, one must attack the "elliptic curve discrete logarithm problem." Some of the potential benefits of ECC are that it uses significantly shorter key lengths and that is well-suited for low bandwidth/CPU systems.

94.6 Hash Algorithms

Hash or digest functions generate a fixed-length hash value from arbitrary-length data. This is usually a one-way process, so that it impossible to reconstruct the original data from the hash. More importantly, it is, in general, extremely difficult to obtain the same hash from two different data sources. Therefore, these types of functions are extremely useful for integrity checking and the creation of electronic signatures or fingerprints.

94.6.1 MD5

The Message Digest (MD) format is probably the most common hash function in use today. This function was developed by Ron Rivest at RSA, and is commonly used as a data integrity checking tool, such as in Tripwire and other products. MD5 generates a 128-bit hash.

94.6.2 SHA

The Secure Hash Algorithm (SHA) was developed by the NSA. The algorithm is used by PGP, and other products, to generate digital signatures. SHA produces a 160-bit hash.

94.7 Steganography

Steganography is the practice used to conceal the existence of messages. That is different from encryption, which seeks to make the messages unintelligible to others.[5]

A detailed discussion of this topic is outside the scope of this chapter, but the reader should be aware that there are many techniques and software packages available that can be used to hide information in a variety of digital data.

94.8 Key Distribution

One of the fundamental problems with encryption technology is the distribution of keys. In the case of symmetric cryptography, a shared secret key must be securely transmitted to users. Even in the case of public key cryptography, getting private keys to users and keeping public keys up-to-date and protected remain difficult problems. There are a variety of key distribution and exchange methods that can be used. These range from manual paper delivery to fully automated key exchanges. The reader is advised to consult the references for further information.

94.9 Key Management

Another important issue for information security professionals to consider is the need for proper key management. This is an area of cryptography that is often overlooked and there are many historical precedents in North America and other parts of the world. If an attacker can easily, or inexpensively, obtain cryptographic keys through people or unprotected systems, there is no need to break the cryptography the hard way.

94.10 Public versus Proprietary Algorithms and Systems

It is generally an accepted fact among cryptography experts that closed or proprietary cryptographic systems do not provide good security. The reason for this is that creating good cryptography is very difficult and even seasoned experts make mistakes. It is therefore believed that algorithms that have undergone intense public and expert scrutiny are far superior to proprietary ones.

94.11 Classic Attacks

Attacks on cryptographic systems can be classified under the following threats:

- Interception
- Modification

[5]Stallings, William. 2002. *Cryptography and Network Security: Principles and Practices*, p. 26, Prentice-Hall, Englewood Cliffs.

- Fabrication
- Interruption

Also, there are both passive and active attacks. Passive attacks involve the listening-in, eavesdropping, or monitoring of information, which may lead to interception of unintended information or traffic analysis where information is inferred. This type of attack is usually difficult if not impossible to detect. However, active attacks involve actual modification of the information flow. This may include:[6]

- Masquerade
- Replay
- Modification of messages
- Denial of service

There are many historical precedents of great value to any security professional considering the use of cryptography. The reader is strongly encouraged to consult many of the excellent books listed in the bibliography, but especially the classic, *The Codebreakers: The Story of Secret Writing*, by David Kahn.[7]

94.12 Standard Cryptanalysis

Cryptanalysis strives to break the encryption used to protect information, and to this end there are many techniques available to the modern cryptographer.

94.12.1 Reverse Engineering

Arguably, one of the simplest forms of attack on cryptographic systems is reverse engineering, whereby an encryption device (method, machine, or software) is obtained through other means and then deconstructed to learn how best to extract plaintext. In theory, if a well-designed crypto hardware system is obtained and even its algorithms are learned, it may still be impossible to obtain enough information to freely decrypt any other ciphertext.[8] During World War II, efforts to break the German Enigma encryption device were greatly aided when one of the units was obtained. Also, today when many software encryption packages that claim to be foolproof are analyzed by cryptographers and security professionals, they are frequently found to have serious bugs that undermine the system.

94.12.2 Guessing

Some encryption methods may be trivial for a trained cryptanalyst to decipher. Examples of this include simple substitutions or obfuscation techniques that are masquerading as encryption. A common example of this is the use of the logical XOR function, which when applied to some data will output seemingly random data, but in fact the plaintext is easily obtained. Another example of this is the Caesar cipher, where each letter of the alphabet is shifted by three places so that A becomes D, B becomes E, etc. These are types of cryptograms that commonly present in newspapers and puzzle books.

The *Principle of Easiest Work* states that one cannot expect the interceptor to choose the hard way to do something.[9]

[6]Stallings, William. 2002. *Cryptography and Network Security: Principles and Practice*, pp. 7–9, Prentice-Hall, Englewood Cliffs.

[7]Kahn, David. 1983. *The Codebreakers: The Story of Secret Writing*, p. 19, Scribner, New York.

[8]Smith, Richard, E. 1997. *Internet Cryptography*, p. 95, Addison-Wesley, Reading, MA.

[9]Pfleeger, E. Charles. 1996. *Security in Computing*, p. 19, Prentice-Hall, Englewood Cliffs.

94.12.3 Frequency Analysis

Many languages, especially English, contain words that repeatedly use the same patterns of letters. There have been numerous English letter frequency studies done that give an attacker a good starting point for attacking much ciphertext. For example, by knowing that the letters E, T, and R appear the most frequently in English text, an attacker can fairly quickly decrypt the ciphertext of most monoalphabetic and polyalphabetic substitution ciphers. Of course, critical to this type of attack is the ready supply of sufficient amounts of ciphertext from which to work. These types of frequency and patterns also appear in many other languages, but English appears particularly vulnerable. Monoalphabetic ciphers, such as the Caesar cipher, directly transpose the frequency distribution of the underlying message.

94.12.4 Brute Force

The process of repeatedly trying different keys to obtain the plaintext are referred to as brute-force techniques. Early ciphers were made stronger and stronger in order to prevent human "computers" from decoding secrets; but with the introduction of mechanical and electronic computing devices, many ciphers became no longer usable. Today, as computing power grows daily, it has become a race to improve the resistance, or work factor, to these types of attacks. This of course introduces a problem for applications that may need to protect data that may be of value for many years.

94.12.5 Ciphertext-Only Attack

The cryptanalyst is presented only with the unintelligible ciphertext, from which she tries to extract the plaintext. For example, by examining only the output of a simple substitution cipher, one is able to deduce patterns and ultimately the entire original plaintext message. This type of attack is aided when the attacker has multiple pieces of ciphertext generated from the same key.

94.12.6 Known Plaintext Attack

The cryptanalyst knows all or part of the contents of the ciphertext's original plaintext. For example, the format of an electronic funds transfer might be known except for the amount and account numbers. Therefore, the work factor to extract the desired information from the ciphertext is significantly reduced.

94.12.7 Chosen Plaintext Attack

In this type of attack, the cryptanalyst can generate ciphertext from arbitrary plaintext. This scenario occurs if the encryption algorithm is known. A good cryptographic algorithm will be resistant even to this type of attack.

94.12.8 Birthday Attack

One-way hash functions are used to generate unique output, although it is possible that another message could generate an identical hash. This instance is called a collision. Therefore, an attacker can dramatically reduce the work factor to duplicate the hash by simply searching for these "birthday" pairs.

94.12.9 Factoring Attacks

One of the possible attacks against RSA cryptography is to attempt to use the public key and factor the private key. The security of RSA depends on this being a difficult problem, and therefore takes significant computation. Obviously, the greater the key length used, the more difficult the factoring becomes.

94.12.10 Replay Attack

An attacker may be able to intercept an encrypted "secret" message, such as a financial transaction, but may not be able to readily decrypt the message. If the systems are not providing adequate protection or validation, the attacker can now simply send the message again, and it will be processed again.

94.12.11 Man-in-the-Middle Attack

By interjecting oneself into the path of secure communications or key exchange, it possible to initiate a number of attacks. An example that is often given is the case of an online transaction. A customer connects to what is thought to be an online bookstore; but in fact, the attacker has hijacked the connection to monitor and interact with the data stream. The customer connects normally because the attacker simply forwards the data onto the bookstore, thereby intercepting all the desired data. Also, changes to the data stream can be made to suit the attacker's needs.

In the context of key exchange, this situation is potentially even more serious. If an attacker is able to intercept the key exchange, he may be able to use the key at will (if it is unprotected) or substitute his own key.

94.12.12 Dictionary Attacks

A special type of known-plaintext and brute-force attack can be used to guess the passwords on UNIX systems. UNIX systems generally use the *crypt()* function to generate theoretically irreversible encrypted password hashes. The problem is that some users choose weak passwords that are based on real words. It is possible to use dictionaries containing thousands of words and to use this well-known function until there is a match with the encoded password. This technique has proved immensely successful in attacking and compromising UNIX systems. Unfortunately, Windows NT systems are not immune from this type of attack. This is accomplished by obtaining a copy of the NT SAM file, which contains the encrypted passwords, and as in the case of UNIX, comparing combinations of dictionary words until a match is found. Again, this is a popular technique for attacking this kind of system.

94.12.13 Attacking Random Number Generators

Many encryption algorithms utilize random data to ensure that an attacker cannot easily recognize patterns to aid in cryptanalysis. Some examples of this include the generation of initialization vectors or SSL sessions. However, if these random number generators are not truly random, they are subject to attack. Furthermore, if the random number generation process or function is known, it may be possible to find weaknesses in its implementation. Many encryption implementations utilize pseudorandom number generators (PRNGs), which as the name the name suggests, attempt to generate numbers that are practically impossible to predict. The basis of these PRNGs is the initial random seed values, which obviously must be selected properly. In 1995, early versions of the Netscape Navigator software were found to have problems with the SSL communication security.[10] The graduate students who reverse engineered the browser software determined that there was a problem with the seeding process used by the random number generator. This problem was corrected in later versions of the browser.

94.12.14 Inference

A simple and potential low-tech attack on encrypted communication can be via simple inference. Although the data being sent back and forth is unreadable to the interceptor, it is possible that the mere

[10]Smith, Richard, E. 1997. *Internet Cryptography*, p. 91, Addison-Wesley, Reading, MA.

fact of this communication may mean there is some significant activity. A common example of this is the communication between military troops, where the sudden increase in traffic, although completely unreadable, may signal the start of an invasion or major campaign. Therefore, these types of communications are often padded so as not to show any increases or decreases in traffic. This example can easily be extended to the business world by considering a pending merger between two companies. The mere fact of increased traffic back and forth may signal the event to an attacker. Also, consider the case of encrypted electronic mail. Although the message data is well encrypted, the sender and recipient are usually plainly visible in the mail headers and message. In fact, the subject line of the message (e.g., "merger proposal") may say it all.

94.13 Modern Attacks

Although classical attacks still apply and are highly effective against modern ciphers, there have been a number of recent cases of new and old cryptosystems failing.

94.13.1 Bypass

Perhaps one of the simplest attacks that has emerged, and arguably is not new, is to simply go around any crypto controls. This may be as simple as coercion of someone with access to the unencrypted data or by exploiting a flaw in the way the cipher is used. There are currently a number of PC encryption products on the market and the majority of these have been found to have bugs. The real difference in these products has been the ways in which the vendor has fixed the problem (or not). A number of these products have been found to improperly save passwords for convenience or have backdoor recovery mechanisms installed. These bugs were mostly exposed by curious users exploring how the programs work. Vendor responses have ranged from immediately issuing fixes to denying there is a problem.

Another common example is the case of a user who is using some type of encryption software that may be protecting valuable information or communication. An attacker could trick the user into running a Trojan horse program, which secretly installs a backdoor program, such as BackOrifice on PCs. On a UNIX system, this attack may occur via an altered installation script run by the administrator. The administrator can now capture any information used on this system, including the crypto keys and passphrases. There have been several demonstrations of these types of attacks where the target was home finance software or PGP keyrings. The author believes that this form of attack will greatly increase as many more users begin regularly using e-mail encryption and Internet banking.

94.13.2 Operating System Flaws

The operating system running the crypto function can itself be the cause of problems. Most operating systems use some form of virtual memory to improve performance. This "memory" is usually stored on the system's hard disk in files that may be accessible. Encryption software may cache keys and plaintext while running, and this data may remain in the system's virtual memory. An attacker could remotely or physically obtain access to these files and therefore may have access to crypto keys and possibly even plaintext.

94.13.3 Memory Residue

Even if the crypto functions are not cached in virtual memory or on disk, many products still keep sensitive keys in the system memory. An attacker may be able to dump the system memory or force the system to crash, leaving data from memory exposed. Hard disks and other media may also have residual data that may reside on the system long after use.

94.13.4 Temporary Files

Many encryption software packages generate temporary files during processing and may accidentally leave plaintext on the system. Also, application packages such as word processors leave many temporary files on the system, which may mean that even if the sensitive file is encrypted and there are no plaintext versions of the file, the application may have created plaintext temporary files. Even if temporary files have been removed, they usually can be easily recovered from the system disks.

94.13.5 Differential Power Analysis

In 1997, Anderson and Kuhn proposed inexpensive attacks against through which knowledgeable insiders and funded organizations could compromise the security of supposed tamper-resistant devices such as smart cards.[11] While technically not a crypto attack, these types of devices are routinely used to store and process cryptographic keys and provide other forms of assurance. Further work in this field has been done by Paul Kocher and Cryptographic Research, Inc. Basically, the problem is that statistical data may "leak" through the electrical activity of the device, which could compromise secret keys or PINs protected by it. The cost of mounting such an attack appears to be relatively low but it does require a high technical skill level. This excellent research teaches security professionals that new forms of high-security storage devices are highly effective but have to be used appropriately and that they do not provide *absolute* protection.

94.13.6 Parallel Computing

Modern personal computers, workstations, and servers are very powerful and are formidable cracking devices. For example, in *Internet Cryptography*,[12] Smith writes that a single workstation will break a 40-bit export crypto key, as those used by Web browsers, in about ten months. However, when 50 workstations are applied to this problem processing in parallel, the work factor is reduced to about six days. This type of attack was demonstrated in 1995 when students using a number of idle workstations managed to obtain the plaintext of an encrypted Web transaction.

Another example of this type of processing is *Crack* software, which can be used to brute-force guess UNIX passwords. The software can be enabled on multiple systems that will work cooperatively to guess the passwords.

Parallel computing has also become very popular in the scientific community due the fact that one can build a supercomputer using off-the-shelf hardware and software. For example, Sandia National Labs has constructed a massively parallel system called Cplant, which was ranked the 44th fastest among the world's 500 fastest supercomputers (http://www.wired.com/news/technology/0,1282,32706,00.html). Parallel computing techniques mean that even a moderately funded attacker, with sufficient time, can launch very effective and low-tech brute-force attacks against medium to high value ciphertext.

94.13.7 Distributed Computing

For a number of years, RSA Security has proposed a series of increasingly difficult computation problems. Most of the problems require the extraction of RSA encrypted messages and there is usually a small monetary award. Various developers of elliptic curve cryptography (ECC) have also organized such contests. The primary reason for holding these competitions is to test current minimum key lengths and obtain a sense of the "real-world" work factor.

Perhaps the most aggressive efforts have come from the Distributed.Net group, which has taken up many such challenges. The Distributed team consists of thousands of PCs, midrange, and high-end

[11] Anderson, Ross, Kuhn, and Markus. 1997. Low Cost Attacks on Tamper Resistant Devices, *Security Protocols, 5th Int. Workshop*.

[12] Smith, Richard, E. *Internet Cryptography*, p. 19.

systems that collaboratively work on these computation problems. Other Internet groups have also formed and have spawned distributed computing rivalries. These coordinated efforts show that even inexpensive computing equipment can be used in a distributed or collaborative manner to decipher ciphertext.

94.13.8 DES Cracker

In 1977, Whitfield Diffie and Martin Hellman proposed the construction of a DES-cracking machine that could crack 56-bit DES keys in 20 hours. Although the cost of such a device is high, it seemed well within the budgets of determined attackers. Then in 1994, Michael Weiner proposed a design for a device built from existing technology which could crack 56-bit DES keys in under four hours for a cost of $1 million. The cost of this theoretical device would of course be much less today if one considers the advances in the computer industry.

At the RSA Conferences held in 1997 and 1998, there were contests held to crack DES-encrypted messages. Both contests were won by distributed computing efforts. In 1998, the DES message was cracked in 39 days. Adding to these efforts was increased pressure from a variety of groups in the United States to lift restrictive crypto export regulations. The Electronic Freedom Foundation (EFF) sponsored a project to build a DES cracker. The intention of the project was to determine how cheap or how expensive it would be to build a DES cracker.

In the summer of 1998, the EFF DES cracker was completed, costing $210,000 and taking only 18 months to design, test, and build. The performance of the cracker was estimated at about five days per key. In July 1998, EFF announced to the world that it had easily won the RSA Security "DES Challenge II," taking less than three days to recover the secret message. In January 1999, EFF announced that in a collaboration with Distributed.Net, it had won the RSA Security "DES Challenge III," taking 22 hours to recover the plaintext. EFF announced that this "put the final nail into the Data Encryption Standard's coffin." EFF published detailed chip design, software, and implementation details and provided this information freely on the Internet.

94.13.9 RSA-155 (512-bit) Factorization

In August 1999, researchers completed the factorization of the 155-digit (512-bit) RSA Challenge Number. The total time taken to complete the solution was around five to seven months without dedicating hardware. By comparison, RSA-140 was solved in nine weeks. The implications of this achievement in relatively short time may put RSA keys at risk from a determined adversary. In general, it means that 768- or 1024-bit RSA keys should be used as a minimum.

94.13.10 TWINKLE RSA Cracker

In summer 1999, Adi Shamir, co-inventor of the RSA algorithm, presented a design for The Weizmann Institute Key Locating Engine (TWINKLE), which processes the "sieving" required for factoring large numbers. The device would cost about $5000 and provide processing equivalent to 100–1000 PCs. If built, this device could be used similarly to the EFF DES Cracker device. This device is targeted at 512-bit RSA keys, so it reinforces the benefits of using of 768- or 1024-bit, or greater keys.

94.13.11 Key Recovery and Escrow

Organizations implementing cryptographic systems usually require some way to recover data encrypted with keys that have been lost. A common example of this type of system is a public key infrastructure, where each private (and public) key is stored on the Certificate Authority, which is protected by a root key(s). Obviously, access to such a system has to be tightly controlled and monitored to prevent a compromise of all the organization's keys. Usually, only the private data encrypting, but not signing, keys are "escrowed."

In many nations, governments are concerned about the use of cryptography for illegal purposes. Traditional surveillance becomes difficult when the targets are using encryption to protect communications. To this end, some nations have attempted to pursue strict crypto regulation, including requirements for key escrow for law enforcement.

In general, key recovery and escrow implementations could cause problems because they are there to allow access to all encrypted data. Although a more thorough discussion of this topic is beyond the scope of this chapter, the reader is encouraged to consult the report entitled "The Risks of Key Recovery, Key Escrow, and Trusted Third Party Encryption," which was published in 1997 by an *ad hoc* group of cryptographers and computer scientists. Also, Whitfield Diffie and Susan Landau's *Privacy on the Line* is essential reading on the topic.

94.14 Protecting Cryptosystems

Creating effective cryptographic systems requires balancing business protection needs with technical constraints. It is critical that these technologies be included as part of an effective and holistic protection solution. It is not enough to simply implement encryption and assume all risks have been addressed. For example, just because an e-mail system is using message encryption, it does not necessarily mean that e-mail is secure, or even any better than plaintext. When considering a protection system, not only must one look at and test the underlying processes, but one must also look for ways around the solutions and address these risks appropriately. It is vital to understand that crypto solutions can be dangerous because they can easily lead to a false sense of information security.

94.14.1 Design, Analysis, and Testing

Fundamental to the successful implementation of a cryptosystem are thorough design, analysis, and testing methodologies. The implementation cryptography is probably one of the most difficult and most poorly understood IT fields. Information technology and security professionals must fully understand that cryptographic solutions that are simply dropped into place are doomed to failure.

It is generally recommended that proprietary cryptographic systems are problematic and usually end up being not quite what they appear to be. The best algorithms are those that have undergone rigorous public scrutiny by crypto experts. Just because a cryptographer cannot break his or her own algorithm, this does not mean that this is a safe algorithm. As Bruce Schneier points out in "Security Pitfalls in Cryptography," the output from a poor cryptographic system is very difficult to differentiate from a good one.

Smith[13] suggests that preferred crypto algorithms should have the following properties:

- No reliance on algorithm secrecy
- Explicitly designed for encryption
- Available for analysis
- Subject to analysis
- No practical weaknesses

When designing systems that use cryptography, it is also important to build in proper redundancies and compensating controls, because it is entirely possible that the algorithms or implementation may fail at some point in the future or at the hands of a determined attacker.

94.14.2 Selecting Appropriate Key Lengths

Although proper design, algorithm selection, and implementation are critical factors for a cryptosystem, the selection of key lengths is also very important. Security professionals and their IT peers often

[13]Smith, Richard, E. *Internet Cryptography*, p. 52.

associate the number of "bits" a product uses with the measure of its level of protection. As Bruce Schneier so precisely puts it in his paper "Security Pitfalls in Cryptography": "…reality isn't that simple. Longer keys don't always mean more security."[14] As stated earlier, the cryptographic functions are but part of the security strategy. Once all the components and vulnerabilities of a encryption strategy have been reviewed and addressed, one can start to consider key lengths.

In theory, the greater the key length, the more difficult the encryption is to break. However, in practice, there are performance and practical concerns that limit the key lengths to be used. In general, the following factors will determine what key sizes are used:

- Value of the asset it is protecting (compare to cost to break it)
- Length of time it needs protecting (minutes, hours, years, centuries)
- Determination of attacker (individual, corporate, government)
- Performance criteria (seconds versus minutes to encrypt/decrypt)

Therefore, high value data that needs to protected for a long time, such as trade secrets, requires long key lengths. Whereas, a stock transaction may only be of value for a few seconds, and therefore is well protected with shorter key lengths. Obviously, it is usually better to err toward longer key sizes than shorter. It is fairly common to see recommendations of symmetric key lengths, such as for 3DES or IDEA, of 112–128-bits, while 1024–2048-bit lengths are common for asymmetric keys, such as for RSA encryption.

94.14.3 Random Number Generators

As discussed earlier, random number generators are critical to effective cryptosystems. Hardware-based RNG are generally believed to be the best, but more costly form of implementation. These devices are generally based on random physical events, and therefore should generate data that is nearly impossible to predict.

Software RNGs obviously require additional operating system protection, but also protection from covert channel analysis. For example, systems that use system clocks may allow an attacker access to this information via other means, such as remote system statistics or network time protocols. Bruce Schneier has identified software random number generators as being a common vulnerability among crypto implementations [SOURCE], and to that end has made an excellent free PRNG available, with source code, to anyone. This PRNG has undergone rigorous independent review.

94.14.4 Source Code Review

Even if standard and publicly scrutinized algorithms and methods are used in an application, this does not guarantee that the application will work as expected. Even open-source algorithms are difficult to implement correctly because there are many nuances (e.g., cipher modes in DES and proper random number generation) that the programmer may not understand. Also, as discussed in previous sections, many commercial encryption packages have sloppy coding errors such as leaving plaintext temporary files unprotected. Cryptographic application source code should be independently reviewed to ensure that it actually does what is expected.

94.14.5 Vendor Assurances

Vendor assurances are easy to find. Many products claim that their data or communications are encrypted or are secure; however, unless they provide any specific details, it usually turns out that this protection is not really there or is really just "obfuscation" at work. There are some industry evaluations

[14]Schneier, Bruce. *Security Pitfalls in Cryptography*, http://www.counterpane.com/pitfalls.html.

and standards that may assist in selecting a product. Some examples are the Federal Information Processing Standards (FIPS), the Common Criteria evaluations, ICSA, and some information security publications.

94.14.6 New Algorithms

94.14.6.1 Advanced Encryption Algorithm (AES)

A new robust encryption algorithm was needed to replace the aging Data Encryption Standard (FIPS 46-3), which had been developed in the 1970s. In September 1997, NIST issued a Federal Register notice soliciting an unclassified, publicly disclosed encryption algorithm that would be available royalty-free, worldwide. Following the submission of 15 candidate algorithms and three publicly held conferences to discuss and analyze the candidates, the field was narrowed to five candidates:

- MARS (IBM)
- RC6TM (RSA Laboratories)
- RIJNDAEL (Joan Daemen, Vincent Rijmen)
- Serpent (Ross Anderson, Eli Biham, Lars Knudsen)
- Twofish (Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson).

NIST continued to study all available information and analyses about the candidate algorithms, and selected one of the algorithms, the Rijndael algorithm, to propose for the AES. The Secretary of Commerce approved FIPS 197, Advanced Encryption Standard (AES), which, effective May 26, 2002, makes it compulsory and binding on federal agencies for the protection of sensitive, unclassified information. The development and public review process has proven very interesting, showing the power of public review of cryptographic algorithms.

94.15 Conclusion

The appropriate use of cryptography is critical to modern information security, but it has been shown that even the best defenses can fail. It is critical to understand that cryptography, while providing excellent protection, can also lead to serious problems if the whole system is not considered. Ultimately, practitioners must understand not only the details of the crypto products they are using, but what they are in fact protecting, why these controls are necessary, and who they are protecting these assets against.