Protocol Failures 541

CHAPTER 11 541
1 INTRODUCTION 343
2 THE NOTARY PROTOCOL 544
3 THE COMMON MODULUS PROTOCOL....... 546
4 THE SMALL EXPONENT PROTOCOL.......... 549
5 THE LOW ENTROPY PROTOCOL................ 550
6 A SINGLE KEY PROTOCOL FAILURE.......... 552
7 SUMMARY AND ANALYSIS 554
REFERENCES 557

CHAPTER 11

Protocol Failures in Cryptosystems"

J. H. Moore
Sandia National Laboratories
Albuquerque, New Mexico 87185

1. Introduction
2. The Notary Protocol
3. The Common Modulus Protocol
4. The Small Exponent Protocol Failure
5. The Low Entropy Protocol Failure
6. A Single Key Protocol Failure
7. Summary and Analysis

*This chapter first appeared in the *Proceedings of the IEEE, vol. 76,* no. 5, May 1988

Abstruct -When a cryptoalgorithm is used to solve data security or authentication problems, it is implemented within the context of a protocol that specifies the appropriate procedures for data handling. The purpose of the protocol is to ensure that when the cryptosystem is applied, the level of security or authentication required by the system is actually attained. In this chapter, we survey a collection of protocols in which this goal has not been met, not because of a failure of the cryptoalgorithm used, but rather because of shortcomings in the design of the protocol. Guidelines for the development of sound protocols will also be extracted from the analysis of these failures.

1 INTRODUCTION

At one time, cryptography was an area of interest only in military and diplomatic circles, with the possible exception of a few eccentric souls with a curiosity for the bizarre. However, in this age of electronic mail, electronic transfer of funds, and huge databases of sensitive medical and personal histories stored in computers with dial-up capabilities, the use of cryptography is widespread enough to touch everyone in our society. Keeping step with these technological developments which require secrecy and authentication, cryptographers have developed new algorithms using complex mathematical systems. These algorithms often require quite sophisticated computing capabilities for their implementation and are designed to withstand attack by equally sophisticated opponents with nearly unlimited resources available to them. However, the mere existence of strong cryptoalgorithms is not enough to solve the problems for which they were developed. A cryptoalgorithm must be used within a set of rules or procedures, known as a ***protocol,*** which insures that the algorithm will actually provide the

Previous Chapter Previous Page Home Next page Next Chapter

security and/or authentication required by the system. Consequently, the development of a system to protect data secrecy and/or integrity actually involves two areas of analysis: the design of strong cryptoalgorithms, and the design of sound protocols.

From the point of view of this chapter, the design of a protocol includes the specification of the characteristics of the cryptoalgorithms that may be used in the protocol without degradation of the security of the system. This specification might be so detailed as to actually define the algorithm that should be used, in which case the protocol should include guidelines for the choice of parameters necessary for that algorithm to protect the data to which it is applied. However, the specification could simply be a list of properties that the cryptoalgorithm must satisfy in order for the protocol to provide the security or authentication for which it was designed.

In this chapter, we will consider examples of protocol failures. By this we mean instances in which the protocol fails to provide the advertised level of security or authentication. Since the examples considered do not use cryptoalgorithms that are inherently weak, the failure involves the protocol design. In all cases, the rules specified in the protocol are sufficient to establish the desired secure data communications if there were no cheating. Unfortunately, those who wish to attack the system are not restricted to only the operations specified by the protocol. As long as they abide by the rules of the protocol while using the system, they can perform any side calculations or manipulations with the data using the information available to them. Although these examples of protocol failures are of interest in and by themselves, more than intellectual curiosity motivates this paper. Careful analysis of these examples should provide insight into principles of protocol design. The extraction of such principles could lead to guidelines for development of future protocols that are resistant to the type of attacks demonstrated in this chapter. Therefore, as we proceed through these examples, we will try to extract some general guidelines that will be summarized in the last section of this chapter.

2 THE NOTARY PROTOCOL

We begin our survey of protocol failures with an example that seems to have first made the cryptocommunity aware that some attacks on cryptosystems were not really revealing weaknesses of the algorithms but rather weaknesses of the protocols calling for their use. This protocol was designed to allow a message to be signed by an entity A, in a way that allowed others to verify at a later date that that message was in fact signed by A. Because in such a concept, A seems to behave as a notary public, we call this the notary protocol.

To set up a notary protocol, A must choose Rivest-Shamir-Adleman (RSA) parameters: primes p and q, and encryption and decryption exponents e and d satisfying $e \cdot d = 1 \bmod \phi(n)$, where $n = pq$. The value of n as well as the public exponent e are published, while d and the factorization of n are kept secret by A.

To sign a document M, the notary uses the private exponent to compute a signature $S = M^d \bmod n$. The signature is then appended to the document much as a notary seal is placed on a paper document. Anyone can use the public information to verify that $S^e = M \bmod n$. Since only A has access to the value d used to create S, the protocol claims to have provided proof that only A could have calculated the signature S.

However, as we pointed out by Davida [3] and Denning [4], it is possible to use the protocol to obtain a forged signature on a document. There are several methods for doing this, but we will include only a simple version here to illustrate the nature of the failure.

For this attack, a forger can use the notary protocol to obtain a signature on a document M, which can be manipulated to obtain a forgery for a signature on another document P. To do this, the forger arbitrarily chooses a value X and computes $Y = X^e$ mod n. He can then use this value to modify the document P on which he wants a signature by calculating $M = YP$. By having the notary sign M, the forger obtains $S = M^d$ mod n. Using this signature, a forged signature S' for P can be obtained by calculating $S' = SX^{-1}$. Notice that the signature for P would be P^d mod n, and that

$$M^d = (YP)^d = Y^d P^d \bmod n$$

By the definition of Y, we know that $Y^d = X$ mod n, so that the above equation becomes $S = XP^d$ mod n, or equivalently, $P^d = SX^{-1} = S'$ mod n. Therefore, the claimed value of this protocol, that is the ability to produce a signature that could only have been obtained if A in fact signed the document, is not attained.

All of the attacks on this protocol rely on the fact that RSA uses a mathematical function, namely, exponentiation, which preserves the multiplicative structure of the input. In fact, the crux of these attacks is that, for any choice of X, M, d, and n,

$$(XM)^d = X^d M^d \bmod n$$

This means that any attempt to modify the protocol to defeat these attacks will have to destroy the ability of a forger to make use of this multiplicative structure. A more general version of Davida's attack has been developed by Desmedt and Odlyzko [9] which can be applied to protocols calling for RSA encryption. In their attack, by obtaining the decryption of a particular collection S of ciphertexts, the cryptanalyst can decrypt further messages by representing them as products of the members of S. In a sense, this makes the ultimate use of the multiplicative structure to defeat a protocol using RSA. It is curious that there are other protocols [2], designed to solve different problems, which in fact exploit this multiplicative property to the benefit of the legitimate users of the system. Apparently then, this structure does not represent an intrinsic weakness of the cryptosystem, although the application in a particular protocol should consider the effect of this structure on the system.

The essence of the flaw in this protocol design, from which a useful design principle can be deduced, is that the signature in an authentication scheme should be the encryption of a message chosen from a selected subset of all possible messages to which the encryption function applies. That is, in this example, the only documents that a notary will sign must possess certain predetermined structure. When the signature is verified, that structure must also be present in the document if the signature is to be accepted as authentic. For this particular example, the set of possible messages is Z_m, the ring of integers modulo m, for some modulus m. To eliminate the ability of the forger to make use of the multiplicative structure, the set of acceptable messages must not be an ideal of Z_m. By this we mean that multiplying any acceptable message by any other message should not necessarily produce an acceptable message. The purpose of such a requirement is to make the creation of forgeries extremely unlikely since the multiplicative structure of the entire system will not carry over to the set of acceptable messages. Protocols for other signature schemes should consider the type of structure which the set of acceptable messages must possess in order to protect against forgeries.

A mathematical model for the system involved in the protocol would make the description of the required properties much easier to state. Some theoretical protocol analysis has begun this type of modeling in [5] and [10], and the results should be valuable in the prevention of similar protocol failures. However, that work is beyond the scope of this chapter, so the reader is referred to the literature for more details.

3 THE COMMON MODULUS PROTOCOL FAILURE

Next we consider a protocol using RSA, which has been proposed on several occasions. In this system, a central keying authority (CKA), would generate two good primes, p and q, calculate the modulus $M = p \cdot q$, and generate encryption/decryption pairs $\{e_i, d_i\}$. Each subscriber in the system would be issued a secret key d_i, along with the public information which consists of the common modulus M and the complete list of public keys $\{e_i\}$. Anyone possessing this public information can send a message X to the nth subscriber by using the RSA encryption algorithm with the public key e_n as the encryption exponent. That is, the ciphertext becomes

$$Y = X^{e_n} \bmod M$$

The protocol is designed to protect the secrecy of the message X sent to subscriber n, since only the nth subscriber knows the secret key, d_n, which allows the decryption of Y. A signature channel is also available in this system, since the nth subscriber can sign a message X by encrypting it using the secret exponent d_n. Both X and the signature, $S = X^{d_n} \bmod M$, is then made public. Anyone can then verify that X was signed by the nth user by using the public exponent e_n to check that $S^{e_n} = X$. As long as only the nth subscriber can uniquely produce the correct signature, that is, a message that when encrypted using the public exponent e_n yields X, the verifier can be confident that the message was authentically signed. By using a common modulus M for all subscribers, the key management is simplified, since only the decryption exponents $\{d_i\}$ must be protected.

However, the use of a common modulus poses several problems for this protocol. As was pointed out by Simmons [21], if a message is ever sent to two subscribers whose public encryption exponents e_i and e_j happen to be relatively prime, then the message can be recovered without breaking the cryptosystem. To demonstrate this, consider the effect of encrypting the message X using e_i and e_j

$$Y_i = X^{e_i} \bmod M$$

$$Y_j = X^{e_j} \bmod M$$

Since e_i and e_j are relatively prime, integers r and s can be found using the Euclidean algorithm, so that

$$re_i + se_j = 1$$

Obviously, either r or s must be negative and for this discussion we will assume that $r < 0$ and write $r = -1 \cdot |r|$. We can also assume that Y_i and Y_j are relatively prime to M, since if this were not true, the Euclidean algorithm could be used to factor the modulus, thereby breaking the cryptosystem. Since Y_i is relatively prime to M, we can

Previous Chapter Previous Page Home Next page Next Chapter

once again use the Euclidean algorithm to calculate the multiplicative inverse of Y_i mod M. The following calculation shows how the message is then recovered

$$[Y_i^{-1}]^{|r|} \cdot [Y_j]^s = [X^{e_i}]^{-1|r|} \cdot [X^{e_j}]^s = X^{re_i+se_j} = X \bmod M$$

Consequently, the protocol fails to protect the secrecy of the message X sent to two subscribers whose public keys are relatively prime. Notice that this does not break the cryptosystem in the traditional sense, since the ability to read the message X does not transfer to an ability to read arbitrary messages encrypted with the same system.

The use of a common modulus also makes this protocol vulnerable to two other attacks in which a subscriber can break the cryptosystem. Once the cryptosystem has been broken, of course, the privacy channel fails since such a subscriber can therefore decrypt messages intended for other users and the signature channel fails since he can also forge the signature of a user without detection. The first attack of this type involves a probabilistic method for factoring the modulus, while the second uses a deterministic algorithm for calculating the encryption/decryption exponent without factoring the modulus. Both of these attacks are described in detail by DeLaurentis [6], and we will only outline the concepts in this chapter.

The basic idea used to factor the modulus, is the identification of a square root of 1 mod M. By this we mean a number b, satisfying

1. $b^2 = 1 \bmod M$,
2. $b \neq \pm 1 \bmod M$,
3. $1 < b < M - 1$.

If such a number can be found, then the modulus M can be factored in the following way. Since $b^2 = 1 \bmod M$

$$\begin{aligned} b^2 - 1 &= 0 \bmod M, \text{ or} \\ (b+1)(b-1) &= 0 \bmod M, \text{ or} \\ (b+1)(b-1) &= sM = spq, \text{ for some integer } s. \end{aligned}$$

However, $1 < b < M$, so that $0 < b - 1 < b + 1 < M = pq$ must hold. These inequalities make clear that p and q cannot both divide either $b - 1$ or $b + 1$. Hence, the greatest common divisor of $b + 1$ and M must be p or q. Applying the Euclidean algorithm will thus yield a factorization of M. Consequently the attack on this system now centers on a method for finding a nontrivial square root of 1 mod M.

Let e_1 and d_1 be the encryption and decryption exponents for a user of the system. By the definition of these exponents, $e_1 d_1 = 1 \bmod \phi(M)$. Thus, $e_1 d_1 - 1$ must be some integer multiple of $\phi(M)$, and the Euclidean algorithm will allow us to find nonnegative integers c and k so that $e_1 d_1 - 1 = c \cdot \phi(M) = 2^k \varphi$ where φ is odd. Consider the following procedure for finding a nontrivial square root of 1 mod M:

1. Choose an integer a such that $(a, M) = 1$ and $1 < a < M - 1$.
2. Find the smallest positive integer, j, which satisfies $a^{2^j \varphi} = 1 \bmod M$. (Since $2^k \varphi$ is a multiple of $\phi(M)$, we know that such an integer does exist.)
3. Let $b = a^{2^{j-1}\varphi}$. If $b \neq -1 \bmod M$, then it is a nontrivial square root of 1.

4. If $b = -1 \bmod M$, return to step 1.

DeLaurentis has shown that this procedure will fail at most half of the time, so that the expected number of trials before a nontrivial square root is found is no greater than 2. Therefore, an insider can break the cryptosystem within this protocol with unacceptably high probability, by using information that each subscriber in the system must have. The attack that we have just described is quite significant since it points out that the knowledge of one encryption/decryption exponent pair for a given modulus M is sufficient to allow the factorization of M.

The second **type** of failure breaks the cryptosystem by demonstrating that a subscriber can use his own public and private keys to generate the private key of another user. That is, given a public encryption exponent e_1, the holder of an encryption/decryption pair e_2, d_2 can find an integer n such that $e_1 n = 1 \bmod \phi(M)$, without actually knowing $\phi(M)$.

To find such an n, it is enough to find an integer which is relatively prime to e_1 and is a multiple of $\phi(M)$. This is verified by noting that if n and e_1 are relatively prime, then there are integers r and s satisfying $rn + se_1 = 1$. If n is also a multiple of $\phi(M)$, then se, = 1 mod $\phi(M)$.

Consider the following procedure for finding such an integer, in which the only values needed are e_1, e_2, and d_2:

1. Using the Euclidean algorithm, find the greatest common divisor f of e_1 and $e_2 d_2 - 1$.
2. Let $n = (e_2 d_2 - 1)/f$.

It is obvious that n is relatively prime to e,. By definition, we know that e_1 is relatively prime to $\phi(M)$. Since f is a divisor of e_1, it must also be relatively prime to $\phi(M)$. However, $nf = e_2 d_2 - 1$ is a multiple of $\phi(M)$, which means n must also be a multiple of $\phi(M)$. The above procedure then yields a decryption exponent for e,. Since the computational difficulty of this procedure has been shown to be at worst $O[(\log M)^2]$, this does pose a viable threat to the system. Once again, the information available to a legitimate user of the system is actually sufficient to break the cryptosystem. Of course, such a user is not operating strictly within the protocol designer's concept of a user, but the information required is obtainable by the user without stepping out of the bounds of the protocol.

The net conclusion of the above three attacks must be that the common modulus protocol fails miserably to protect the secrecy of messages or to provide a means for authenticating the signatures of users of the system. Therefore, in designing new protocols with the RSA algorithm, the use of a common modulus should be avoided. However, these attacks provide several more detailed guidelines for the development of further secure protocols. From the first attack, we note that the protocol designer must consider what an opponent can do with a collection of ciphertext whose plaintexts are related (in the example they were equal) or whose keys are related (in the example they were relatively prime). With respect to the RSA algorithm, two guidelines for applications of the algorithm were established.

1. Knowledge of one encryption/decryption pair of exponents for a given modulus gives rise to a probabilistic algorithm for factoring the modulus whose expected number of trials before success is no greater than 2.

2. Knowledge of one encryption/decryption pair of exponents for a given modulus M gives rise to a deterministic algorithm for calculating other encryption/decryption pairs without having to first determine $\phi(M)$.

4 THE SMALL EXPONENT PROTOCOL FAILURE

Another commonly suggested protocol using RSA involves the use of a small exponent for the public key in order to make the calculations for encryption fast and inexpensive to perform. The general setting for this type of protocol involves a large communication network in which the messages sent between two users should not be readable to other users in the system or by outsiders. The protocol specifies that the ith user should choose two large primes, p_i and q_i, and publish their product n_i as the modulus for an RSA algorithm used in communicating with him. An encryption/decryption pair, $\{e_i, d_i\}$ is chosen and one of these, say d_i, is published. We are interested in the case when the encryption exponent is chosen to be a small integer. For some applications, this is a very appealing choice since the implementation with a small exponent can be simpler and quicker to operate. However, such a specification causes the protocol to fail if the exponent is d and the same message is sent to at least d users. This observation has been made by several people, at least Blum, Lieberherr, and Williams, and seems to have become part of the cryptoanalytic folklore. To illustrate the problem, consider the case when $d = 3$. Suppose that user 1, whose public exponent is 3, decides to send a message M to users 2, 3, and 4. The ciphertexts then are

$$C_2 = M^3 \bmod n_2$$
$$C_3 = M^3 \bmod n_3$$
$$C_4 = M^3 \bmod n_4$$

If n_2, n_3, and n_4 are relatively prime, the Chinese remainder theorem will enable the calculation of $M^3 \bmod (n_2n_3n_4)$ from the knowledge of C_2, C_3, C_4. But $M^3 < n_2n_3n_4$, so M can be recovered. If n_2, n_3, and n_4 are not relatively prime, then the attacks from the common modulus protocol apply. Thus, even an eavesdropper has enough information to recover the message.

A common technique for salvaging this protocol is to never send exactly the same message by using something like a time stamp concatenated to the message before the encryption takes place. In our example above, the new ciphertexts in this protocol would be

$$C_2 = (2^{|t_2|}M + t_2)^3 \bmod n_2$$
$$C_3 = (2^{|t_3|}M + t_3)^3 \bmod n_3$$
$$C_4 = (2^{|t_4|}M + t_4)^3 \bmod n_4$$

where t_2, t_3, and t_4 are times associated with each message. This foils the previously described attack.

However, Hastad [11] has shown that this may not vary the plaintext enough to overcome the weakness inherent in the low exponent protocol. In his paper, Hastad showed that a system of modular polynomial equations

$$P_i(x) = 0 \bmod n_i, \ 1 \le i \le k$$

of degrees no greater than d can be solved in polynomial time if the number of equations is larger than $d(d+1)/2$. Thus in the case of an exponent of 3, if the message, adjusted by a time stamp, is sent to at least seven members of the network, the message may no longer be secret to an eavesdropper. Of course, for this attack to be viable, the time stamps must be known. However, the time stamps involve a small number of bits as compared with the total size of the message, so that these could be estimated before applying Hastad's algorithm. Obviously, the security of the system cannot reasonably depend on the secrecy of these time stamps alone.

Hastad's results require some lattice theory which is beyond the scope of this chapter. Therefore, we will not delve deeper into the actual techniques used to attack the secrecy of messages in such a network. The interested reader can investigate the details by referring to the original paper. However, it is important to note that the proof of the result requires the use of an algorithm for lattice basis reduction [17] which has only recently been developed and has had a rather extensive effect on cryptosystem analysis. The reader is referred to Brickell and Odlyzko's chapter, "Cryptanalysis: A Survey of Recent Results," which also appears in this volume, for further discussion of the impact of this algorithm on cryptanalysis.

The protocol failure considered in this section emphasizes the point made in the last section that the protocol designer must consider the information that can be gained from a collection of ciphertexts whose plaintexts are related (here equal or differing by time stamps) or whose keys are related (here the same exponent with relatively prime moduli).

5 THE LOW ENTROPY PROTOCOL FAILURE

In both the low exponent and the common modulus protocol failures, some mathematical properties of the cryptoalgorithm enabled the attacker to cause the protocol to fail. It might be surmised at this point, that all protocol failures are brought about by using some mathematical idiosyncrasies of the cryptoalgorithm. However, this is not always the case and in this section, we will examine a failure that is not really dependent on the particular algorithm used.

The purpose of a secrecy channel is to hide the meaning of a message from all but the authorized receivers. A strong cryptoalgorithm is employed to make decryption of the message infeasible for an outsider. However, if only a small number of messages can possibly be sent, the meaning of the message might be discernible without decryption. This is particularly a problem when a public key system is used, as was pointed out by Holdridge and Simmons [22].

When a public key system is used in a protocol providing for a secrecy channel, the encryption key is publicly known so that anyone can send a message through the channel that can be understood only by the intended receiver, who possesses the private decryption key. But, if only a small number of messages are meaningful, an opponent could precompute the encryption of those messages. Then when an encrypted message is sent through the channel, only a search through these precomputed values must be made to realize the meaning of the message. The simplest example of such a failure would occur if there were only two possible messages to be sent, such as "yes" and "no." Anyone possessing the public encryption key could encrypt these two messages and then when a "secret" message was sent through the system, easily distinguish between the two possible plaintext messages. Obviously, this is an extreme example, but the potential problem is well illustrated by it.

The above discussion makes clear that the message space to which a secrecy channel using public key encryption is applied, must be large enough to preclude an attack in which the set of all possible messages is pre-encrypted by an adversary who can then intercept ciphertext and recover the plaintext by exhaustive search. However, the encryption of all messages in the space is not required for this attack to work. If the ciphertext for a significant part of the message space is precomputed, the meaning of a given encrypted message may be discernible by simply matching, whenever possible, ciphertext in the message with the precomputed table, without decryption of the rest of the message. The definition of “significant” in the previous statement depends on the nature of the particular message space used in the system.

The title for this section refers to low entropy as the cause of this protocol failure. Entropy is a measure of uncertainty in the message space in the sense that it measures the amount of information that must be determined about a given message in order to make sense of it. If the message space has low entropy, only a small amount of information about a given message will reveal the **total** information contained in the message. Using this now-defined concept, we can state that the protocol failure examined in this section is the use of a public key system with a message space of low entropy.

In the paper referenced above, Holdridge and Simmons pointed out and demonstrated this protocol failure in an application proposed by Bell Telephone Laboratories for use in secure telephony [12,13]. In this system, each subscriber would enter his encryption key in a public key directory, keeping his decryption key secret. The messages to be encrypted in this system consist of digitized voice transmissions in a mobile radio telephone net. To speak with a given subscriber in the net, the sampled and digitized speech signal of the sender would be encrypted with the subscriber’s encryption key, available from the public directory. On receipt, the subscriber could use his privately held decryption key to recover the plaintext message.

However, voice signals have a rather narrow bandwidth thus enabling an attack on the system since most of the information being transmitted occurs in a rather small collection of data. This combined with the ability of the human mind to use the redundancy, caused by the intersymbol and interword dependencies, of our language to ascertain the meaning of a sample of corrupted speech signals, makes the protocol for this system fail to protect the secrecy of messages.

To be more specific, in their demonstration, Holdridge and Simmons used digitized speech data that was formatted into 32-bit blocks in preparation for encryption. Thus the actual message space for the encryption algorithm consisted of all 32-bit words. However, they were able to determine that the entropy of such speech-derived messages was more like 16 to 18 bits, rather than 32. Using a precomputed table of about 100,000 ciphers, they reconstructed parts of the original data from various enciphered messages. Then using interpolation to fill in missing data, they created approximations to the original plaintext data. Finally these approximations were made into audio tapes and played to several listeners who were able to recover at least 90% of the original messages.

The particular public key algorithm used has no real bearing on the protocol failure. The cryptosystems used were not broken by such an attack, while the secrecy channel failed. Once again, it is merely a failure of the protocol-namely, the use of a public key encryption system to protect messages drawn from a message space with low entropy-to provide the advertised privacy.

6 A SINGLE KEY PROTOCOL FAILURE

Thus far, in our survey of protocol failures, all of the examples have used public key algorithms. To avoid the erroneous conclusion that single-key encryption systems are impervious to protocol failures, we will consider in this section, a protocol using DES that fails to provide the authentication for which it was designed.

Whenever encrypted data are transmitted over nonsecure lines, the vulnerability of the data to manipulation by an intruder must be considered. Even though such an intruder may not be able to decrypt the ciphertext, he may be able to use the characteristics of the protocol to manipulate the data to his benefit.

A solution to this problem is to append to the ciphertext a code that is a function of the actual plaintext message and that can be checked by the receiver against the result of applying that function to the received message. Agreement of these two values would validate that the message received had not been altered in transmission. If this function is chosen appropriately, an opponent should find it difficult to modify the ciphertext and/or the code without failing this validity check.

Two quite distinctive techniques for computing such a code have been considered. The first calculates a Message Authentication Code (MAC) by using a modified version of an encryption function. This function requires a secret key, different from the one used for preparation of the ciphertext, but chooses only some of the bits from the ciphertext as the code in an effort to keep the total message size as small as possible. The second method computes a Manipulation Detection Code (MDC), using a function that requires no secret information. The security advantages of the MAC over the MDC are apparent, but for some applications, a further encryption step or a complication of the key management system may place an unbearable burden on the communication channel.

A protocol for protecting information encrypted with DES with an MDC was proposed for inclusion in a federal standard for data communications. The proposal was quite appealing, probably because of its simplicity, and was mentioned in several publications [20] and books [19] before it was discovered that it failed to detect several types of manipulations that appear to be viable areas of concern.

The setting in which this protocol was designed to operate is as follows. The data to be transmitted are divided into n blocks each consisting of k bits, where k varies between 1 and 64. Denote these plaintext blocks by $X_1, X_2, \ldots, X_n$. After the blocks have been encrypted using the appropriate mode of DES to form the ciphertext blocks, $Y_1, Y_2, \ldots, Y_n$, the MDC is formed, Y_{n+1}, which is the exclusive-or sum of the n plaintext blocks. This block can be used by the authorized receiver to verify that the data received were not tampered with during transmission. The receiver simply decrypts the data and calculates the exclusive-or sum of these blocks, comparing the result to the MDC. If they agree, the receiver will conclude that the ciphertext had not been manipulated.

To illustrate the failure of this protocol, consider an application using Cipher Block Chaining as the mode of DES. In this mode, a 64-bit initializaton vector Y_0 and a 56-bit key K are exchanged secretly between those wishing to communicate. The ciphertext then consists of n blocks, $Y_1, Y_2, \ldots, Y_n$, each consisting of 64 bits, calculated by

$$Y_i = E(K, X_i \oplus Y_{i-1})$$

where $E(K, X_i)$ represents encryption with DES under the key K of the message X,. The MDC is computed as

$$Y_{n+1} = \bigoplus_{i=1}^{n} X_i$$

When the message is received, the first n blocks, $Z_1, Z_2, \ldots, Z_n$, are decrypted using the reverse operations. That is, the plaintext received is calculated as

$$W_i = Z_{i-1} \oplus D(K, Z_i)$$

where $D(K, Z_i)$ represents decryption with DES under the key K of the message Z,. The exclusive-or sum of these n blocks is then formed and compared with Y_{n+1}. If these are identical, the receiver will conclude that the message had not been manipulated.

Of **course,** this check merely confirms that $\oplus X_i = \oplus W_i$ so that an intruder can modify the message without detection as long as these sums remain equal. An obvious technique for spoofing this system is to rearrange the order of the blocks. Since the sum of the decrypted text will not change, this manipulation will not be detected. The seri**ousness** of this problem depends on the nature of the plaintext. If the message was a transfer of $50,000 from your account to mine, I would certainly like to show that the account numbers were not rearranged, resulting in the transfer of $50,000 from my account to yours. If the account identification numbers happened to fall on 64-bit boundaries, the above protocol would not detect such a change. Secondly, blocks can be inserted without detection as long as they occur in pairs, since the exclusive-or sum of a block with itself yields a zero **block** and therefore will not affect the MDC. This could be advantageous to the intruder, particularly if he knew an encrypted block of data that could change the value of a deposit to his account from $1000 to $1,000,000. The severity of these failures would have to be analyzed for any particular application, but regardless, the protocol does not provide the advertised protection against undetected manipulation.

In Output Feedback Mode, the DES is used to generate a cryptographic bit stream which is then exclusive-or summed with the plaintext. To be specific, the plaintext is divided into n blocks of 64 bits each. Letting R_0 denote the initialization vector, n blocks of pseudorandom bits are found by

$$R_i = E(K, R_{i-1})$$

The ciphertext blocks are then obtained by

$$Y_i = X_i \oplus R_i, \qquad 1 \le i \le n$$

The MDC, Y_{n+1}, is once again calculated as the exclusive-or sum of the plaintext blocks. The receiver of the ciphertext calculates the values of R_i for as many blocks of data as were received. These are then exclusive-or summed with the ciphertext blocks. As long as the **sum** of these decrypted blocks is the same as Y_{n+1}, the message is accepted without question. Once again, the intruder can manipulate the data without detection, provided the sum of the decrypted blocks remains the same as Y_{n+1}.

It is easy to see that rearrangement of the order of the ciphertext blocks will not be detected by using the MDC, although such a rearrangement may produce, on decryption, blocks of random numbers, which may or may not be interpreted as valid data. By the same token, any blocks can be substituted as long as the exclusive-or sum of all the ciphertext blocks remains fixed. To see this, let $Z_1, Z_2, \ldots, Z_n$ be the received ciphertext blocks and assume that the exclusive or sum $Z_1 \oplus Z_2 \oplus \ldots \oplus Z_n$ is the same as $Y_1 \oplus Y_2 \oplus \ldots \oplus Y_n$. In this case when the ciphertext is decrypted, the recovered plaintext will be $W_1, W_2, \ldots, W_n$, which satisfies

$$\begin{aligned} W_1 \oplus W_2 \oplus \ldots \oplus W_n \\ &= R_1 \oplus R_2 \oplus \ldots \oplus R_n \oplus Z_1 \oplus Z_2 \oplus \ldots \oplus Z_n \\ &= R_1 \oplus R_2 \oplus \ldots R_n \oplus Y_1 \oplus Y_2 \oplus \ldots \oplus Y_n \\ &= X_1 \oplus X_2 \oplus \ldots \oplus X_n = Y_{n+1} \end{aligned}$$

The intruder thus has considerable flexibility in choosing messages to substitute in the ciphertext, although the decrypted values may be jibberish. Once again, the type and format of the plaintext will determine whether or not the received message would be accepted as genuine, but the protocol, as proposed, does not provide adequate protection against data manipulation. Obviously, further constraints will be required by this protocol to insure that such manipulations will not allow the acceptance of tampered ciphertext by the receiver.

Similar techniques also show that the MDC proposed does not detect insertions by pairs with Cipher Feedback Mode. The interested reader is referred to a paper by Jueneman, Meyer, and Matyas [15] for details of all attacks on this protocol for authentication of messages using DES.

7 SUMMARY AND ANALYSIS

We have surveyed a number of protocol failures which have hopefully covered a broad enough spectrum to convince the reader that the problems are widespread and that much can be learned from considering these past failures. In fact, many reports of the "breaking" of cryptosystems, may be better described as the "revealing" of protocol failures. The distinction seems to be that when a weakness is reported in a cryptosystem, the effect of which is to merely limit the scope of application or more clearly define the range of parameters that should be used for the algorithm, then the flaw discovered probably represents a protocol failure. However, if the effect is to leave the cryptosystem useless in any setting or to so severely restrict the possible range of parameters that the definition of a strong cryptofunction is infeasible, then the cryptosystem is actually "broken." The point which we appear to be belaboring is not merely a question of semantics, since the reaction by the cryptocommunity to these two results will be drastically different. A protocol failure can lead to the definition of new guidelines for the use of a particular algorithm or class of algorithms, whereas a broken cryptosystem simply removes a given algorithm from consideration by protocol designers.

The examples of protocol failures considered in this chapter seem to naturally fall into three distinct classifications based on the type of flaw which was revealed. The first classification is characterized by the identification of a weakness in the cryptoalgorithm, of the type discussed in the last paragraph, as applied in the protocol. The low exponent and common modulus protocols using RSA are examples of this class of failures. We were able to. identify several restrictions on the use of RSA by analyzing these failures. For emphasis and for completeness, we will restate those here.

1. Knowledge of one encryption/decryption pair of exponents for a given modulus gives rise to a probabilistic algorithm for factoring the modulus whose expected number of trials before success is no greater than 2.

2. Knowledge of one encryption/decryption pair of exponents of a given modulus M gives rise to a deterministic algorithm for calculating other encryption/decryption pairs without having to first determine $\phi(M)$.

3. A common modulus should not be used in a protocol using RSA in a communication network. (This follows from the first two points above as well as from the other attack discussed in this section on the common modulus protocol failure.)

4. Given a collection of k modular equations of the form $(X + t_i)^d$ mod n_i, where $k > d(d + 1)/2$ and the $\{t_i\}$ are known, then X can be found in time polynomial in both k and log n,. The implication for RSA of this is that the exponents chosen in a protocol should not be small.

The second class of protocol failure is caused by the oversight of some principle applicable to a broad class of cryptoalgorithms, as seen in the low entropy or the notary protocol failure examples. In the case of the low entropy protocol failure, a general rule-that public key encryption should not be applied to protect messages that are drawn from a message space with low entropy-was not heeded by the protocol. With the notary example, a broad principle is exposed, namely, that applying encryption to plaintext to form signatures requires that the plaintext possess some verifiable structure to make forgeries difficult to obtain. When analyzing a particular implementation of a signature scheme, the nature of this structure may be obvious and even easy to describe. Unfortunately, for a generic signature scheme, the properties that the set of acceptable messages used to form signatures must possess, are difficult to state without reference to the specific algorithm being used. This makes a precise statement of the principle for signature schemes, which should be extractable from the notary protocol failure, difficult to express. However, the current trends in theoretical protocol analysis have been moving in the direction needed to formally state the principle required for a secure signature protocol.

The last classification, and probably the one from which the least amount of information can be elicited, is the protocol failure in which the designer simply overstated the amount of security that the protocol can provide. Such is the case in the single-key example in which a code to enable a receiver to detect ciphertext manipulation was calculated with no secret input by the sender. The designer of the protocol should have clearly stated the level of manipulation detection that such a code could provide, and as pointed out in our discussion of that scheme, the level of protection against undetected tampering was questionably low. Jueneman [14] has continued to work on modifications to this code, with some level of success, but not without considerably more complications to the calculations for the code. These complications are probably unavoidable simply from the application of the colloquial adage “you can’t buy something for nothing.” Protocol designers, however, should try to be as precise as possible in their claims about the level of security or authentication which can be expected from a given protocol.

Other papers surveying protocols have included some analysis of the problems which arise in verifying security. A two-step approach for protocol designers was suggested in [7] and [8] which bears repeating here.

1. Identify explicit cryptographic assumptions.

2. Determine that any successful attack on the protocol requires the violation of at least one of those assumptions.

This is a reasonable and straightforward set of guidelines, which really should be regarded as essential analysis by protocol designers. Of course, in practice, the proof required in step (2) may be quite difficult, if not impossible. The following modified version of these two steps might provide even more information to potential users of the protocol, while still providing the benefits of the analysis as originally stated.

1. Identify **all** assumptions made in the protocol.
2. For each assumption in step (1), determine the effect on the security of the protocol if that assumption were violated.

It is not clear that the first steps in the two versions are really different. However, the discussion that followed the identification of the original steps in the paper from which they were taken, leaves open the possibility that only the mathematical assumptions would be considered in the first version. The intention in step (1) of the modified version is to clarify even the setting that the protocol designer may have assumed would exist for the application of the protocol. The second step in the modified version could be thought of as an expansed version of the original step (2). This expansion provides real benefits, in that it helps to clarify the purpose of each assumption, helps to identify those elements of the protocol that are most critical, and makes modification of the protocol simpler to accomplish, if in practice some assumption cannot be met. The basic level of analysis for a protocol designer should include these steps. By themselves they will not prevent protocol failures, but should help in the early detection of flaws.

The restrictions for applications of RSA enumerated earlier in this section, may be simply specific applications of a set of more general principles for protocol design. In describing these general principles, the notation $E_i(M_j)$ will be used to denote the encryption with a fixed algorithm E, using the ith key, of a message M_j. The possible relationships among a collection of these ciphertexts should always be considered, especially when the encryption functions may be related because the keys satisfy some known relationship, or when the plaintexts satisfy some known relationship.That is, protocol designers should consider the following collections of ciphertext, which may be available to an opponent through the protocol, analyzing their effects on security.

1. A collection $\{E_i(M)\}$, where various keys are used to encrypt the same message, particularly if the keys used in this collection are related.
2. A collection $\{E(M_j)\}$, where the same key is used to encrypt messages M_j which satisfy some known relationship.
3. A collection $\{E_i(M_j)\}$, where various keys are used to encrypt known variations M_j of the same message. This is particularly interesting if the keys used also satisfy some known relationship.

The first principle was the basis of an attack on the common modulus protocol, while the third principle was involved in the low exponent protocol failure. These should actually be used in the analysis of all cryptoalgorithms, with any results serving as guidelines for the application of the algorithm by protocol designers.

Research in the area of cryptographic protocols has also been increasing with some interesting results. The basic approach used in this research has been to make a few simplifying assumptions, then mathematically model the setting and operations of the protocol. The mathematical model gives the researcher the tools to prove statements about the security of protocols which fit the model. One line of research has produced a computer program which searches for security vulnerabilities in protocols for key distribution [16]. Although still under development, it has demonstrated some level of success by its ability to rediscover some known failures in key distribution protocols. Some important theoretical work has been done in [1], [5], and [10] in which the results are more abstract. In each of these papers, restricted settings have been considered in order to obtain results, but progress is being made to give some foundation to the area of security analysis of protocols. Meanwhile, protocol designers may have to rely upon analysis of past errors as a major source of guidelines for strong cryptographic protocol development.

REFERENCES

[I] R. Berger, S. Kannan, and R. Peralta, "A framework for the study of cryptographic protocols," in *Lecture Notes in Computer Science 218; Advances in Cryptology: Proc. Crypto* '85, H. C. Williams, Ed., Santa Barbara, CA, Aug. 18-22, 1985, pp. 87-103. Berlin: Springer-Verlag, 1986.

[2] D. Chaum, ' 'Untraceable electronic mail, return addresses and digital pseudonyms," *Commun. ACM,* vol. 24, no. 2, pp. 84-88, Feb. 1981.

[3] G. I. Davida, "Chosen signature cryptanalysis of the RSA(MIT) public key cryptosystem," *Tech. Rept. TR-82-2.* Milwaukee: University of Wisconsin Department of Electrical Engineering and Computer Science, Oct. 1982.

[4] D. E. Denning, "Digital signatures with RSA and other public key cryptosysterns," Commun. ACM, vol. 27, pp. 388-392, April 1984.

[5] D. Dolev and A. C. Yao, "On the security of public key protocols," *IEEE Trans. Inform. Theory,* vol. IT-29, no. 2, pp. 198-208, March 1983.

[6] J. M. DeLaurentis, "A further weakness in the common modulus protocol for the RSA cryptoalgorithm," *Cryptologia,* vol. 8, no. 3, pp. 253-259, July 1984.

[7] R. A. DeMillo, G. L. Davida, D. I? Dobkin, M. A. Harrison, and R. J. Lipton, *Applied Cryptology, Cryptographic Protocols, and Computer Security Models, Proc. Symposia Appl. Math.,* Providence: American Mathematical Society, vol. 29, 1983.

[8] R. A. DeMillo and M. J. Merritt, "Protocols for data security," Computer, vol. 16, no. 2, pp. 39-50, Feb. 1983.

[9] Y. Desmedt and A. M. Odlyzko, "A chosen text attack on the RSA cryptosystem and some discrete logarithm schemes," in *Lecture Notes in Computer Science 218; Advances in Cryptology: Proc. Crypto '85,* H. C. Williams, Ed., Santa Barbara, CA, Aug. 18-22, 1985, pp. 516-522. Berlin: Springer-Verlag, 1986.

[IO] S. Even, 0. Goldreich, and A. Shamir, "On the security of ping-pong protocols using the RSA," in *Lecture Notes in Computer Science 218; Advances in Cryptology: Proc. Crypto* '85, H. C. Williams, Ed., Santa Barbara, CA, Aug. 18-22, 1985, pp. 58-72. Berlin: Springer-Verlag, 1986.

[11] J. Hastad, "On using RSA with low exponent in a public key network," in *Lecture Notes in Computer Science 218; Advances in Cryptology: Proc. Crypto '8.5,* H. C. Williams, Ed., Santa Barbara, CA, Aug. 18-22, 1985, pp. 403-408. Berlin: Springer-Verlag, 1986.

[12] P. S. Henry, "Fast implementation of the knapsack cipher," *Bell Syst. Tech. J.,* vol. 60, pp. 767-773, 1981.

[13] P. S. Henry and R. D. Nash, "High speed hardware implementation of the knapsack cipher," paper presented at Crypto '8 1, IEEE Workshop on Communications Security, Santa Barbara, CA, Aug. 24-26, 1981.

[14] R. R. Jueneman, "A high speed manipulation detection code," in *Lecture Notes in Computer Science 263; Advances in Cryptology: Proc. Crypto '86,* A. M. Odlyzko, Ed., Santa Barbara, CA, Aug. 11-15, 1986, pp. 327-346. Berlin: Springer-Verlag, 1987.

[15] R. R. Jueneman, S. M. Matyas, and C. H. Meyer, "Message authentication with manipulation detection codes," in *Proc. 1983 IEEE Symp. Security and Privacy,* R. Blakley and D. Denning, Eds., Oakland, CA, April 1983, pp. 33-54. Los Angeles: IEEE Computer Society Press, 1983.

[16] J. K. Millem, S. C. Clark, and S. B. Freedman, "The interrogator: Protocol security analysis," *IEEE Trans. Software Eng.,* vol. SE-13, no. 2, pp. 274-278, Feb. 1987

[17] A. K. Lenstra, H. W. Lenstra, and L. Lovasz, "Factoring polynomials with integer coefficients," *Mathematische Annalen, vol.* 261, pp. 513-534, 1982.

[18] M. J. Merritt, "Cryptographic protocols," Ph.D. Thesis GIT-ICS-83/06, Georgia Institute of Technology, 1983.

[19] C. H. Meyer and S. M. Matyas, *Cryptography: A New Dimension in Computer Data Security.* New York: Wiley, 1982, pp. 457-458.

[20] Proposed Federal Standard 1026, "Telecommunications: Interoperability and security replacements for the use of the data encryption standard in the physical and data link layers of data communications," National Communications System, Washington, DC, draft of June 1, 1981.

[21] G. J. Simmons, "A 'weak' privacy protocol using the RSA cryptoalgorithm," *Cryptologia,* vol. 7, pp. 180-182, 1983.

[22] G. J. Simmons and D. B. Holdridge, "Forward search as a cryptanalytic tool against a public key privacy channel," in *Proc. 1982 Symp. on Security and Privacy,* R. Schell, Ed., pp. 117-128. Los Angeles: IEEE Computer Society Press, 1982.