Attacking Authentication Protocols

John Clark

31 March 1996

1 Introduction

The past two decades have seen an enormous increase in the development and use of networked and distributed systems, providing increased functionality to the user and more efficient use of resources. To obtain the benefits of such systems parties will cooperate by exchanging messages over the network. The parties may be users, hosts or processes; they are generally referred to as *principals* in authentication literature.

Principals use the messages received, together with certain modelling assumptions about the behaviour of other principals to make decisions on how to act. These decisions depend crucially on what validity can be assumed of messages that they receive. Loosely speaking, when we receive a message we want to be sure that it has been created recently and in good faith for a particular purpose by the principal who claims to have sent it. We must be able to detect when a message has been created by a malicious principal or when a message was issued some time ago (or for a different purpose) and is currently being replayed on the network.

An authentication protocol is a sequence of message exchanges between principals that either distributes secrets to some of those principals or allows the use of some secret to be recognised [4]. At the end of the protocol the principals involved may deduce certain properties about the system; for example, that only certain principals have access to particular secret information (typically cryptographic keys) or that a particular principal is operational. The principals may then use this knowledge to verify claims about subsequent communication, for example, that a received message encrypted with a newly distributed key must have been created after distribution of that key and so is *timely*.

A considerable number of authentication protocols have been specified and implemented. The area is, however, remarkably subtle and many pro-

tocols have been shown to be flawed a long time after they were published. In this paper we outline some of the ways in which protocols may fail to meet their goals. Since authentication relies heavily on encryption and decryption to achieve its goals we first provide the appropriate knowledge about cryptography.

2 Cryptographic Prerequisites

2.1 General Principles

Cryptographic mechanisms are fundamental to authentication protocols. Suppose that we have some message text P which we wish to transmit over the network. P is generally referred to as **plaintext** or a **datagram**. A cryptographic algorithm converts P to a form that is unintelligible to anyone monitoring the network. This conversion process is called **encryption**. The unintelligible form is known as **ciphertext** or a **cryptogram**. The precise form of the ciphertext C corresponding to a plaintext P depends on an additional parameter K known as the **key**.

The intended receiver of a ciphertext C may wish to recover the original plaintext P. To do this, a second key K^{-1} is used to reverse the process. This reverse process is known as **decryption**. This is depicted in figure 1 below.

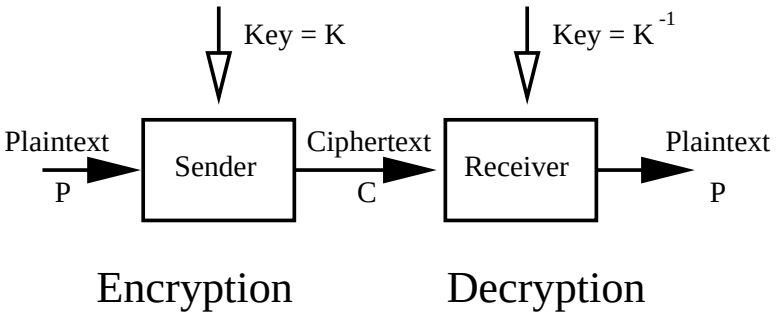

Figure 1: Encryption and Decryption

By restricting appropriately who has access to the various keys involved we can limit the ability to form ciphertexts and the ability to derive the corresponding plaintexts.

2.2 Symmetric and Public Key Cryptography

In **symmetric key cryptography** the encryption key K and the decryption key K^{-1} are easily obtainable from each other by public techniques. Usually they are identical and we shall generally assume that this is the case. The key K is used by a pair of principals to encrypt and decrypt messages to and from each other. Of course, anyone who holds the key can create and read the contents of ciphertext messages. To ensure security of communication this key is kept secret between the communicating principals. Following established convention we shall use the notation Kab to denote a key intended for communication between principals A and B using a symmetric key cryptosystem.

In **public key encryption** each principal A is associated with some key pair (Ka, Ka^{-1}). The **public key** Ka is made publicly available but the principal A does not reveal the **private key** Ka^{-1}. Any principal can encrypt a message P using Ka and only principal A can then decrypt it using Ka^{-1}. Thus, the secrecy of a messages to A can be ensured.

Some public key algorithms allow the private key to be used to encrypt plaintext with the public key being used to decrypt the corresponding ciphertext. If a ciphertext C decrypts (using Ka) to a meaningful plaintext message P then it is assumed that the ciphertext must have been created by A using the key K_a^{-1}. This can be used to guarantee the *authenticity* of the message. The most widely-known public key algorithm that allows such use was developed by Rivest, Shamir and Adleman [19] and is universally referred to as RSA.

2.3 Notational Conventions

In this paper we shall use the notation $E(K : P)$ to denote the result of encrypting message plaintext P with key K.

A protocol run consists of a sequence of messages between principals and will be described using the standard notation. Principals are generally denoted by A, B and S (for a server). The sequence of messages

$$\begin{array}{lll} (1)\ A \rightarrow B & : & M_1 \\ (2)\ B \rightarrow S & : & M_2 \\ (3)\ S \rightarrow B & : & M_3 \end{array}$$

denotes a protocol in which A sends M_1 to B, B then sends M_2 to S who then sends M_3 to B.

Attacks on protocols often involve some mischievous principal pretending to be another. We denote a mischievous principal by Z. The notation $Z(A)$ denotes the principal Z acting in the role of A. Z has unfettered access to the network medium and may place at will messages onto the net claiming to be sent from A and intercepting messages destined for A (and possibly removing them).

A number generated by a principal A is denoted by Na. Such numbers are intended to be used *only once* for the purposes of the current run of the protocol and are generally termed **nonces**. A message may have several components; some will be plaintext and some will be encrypted. Message components will be separated by commas. Thus

$$(1)\ A \rightarrow B \quad : \quad A, E(Kab : Na)$$

denotes that in the first message of the protocol A sends to B the message whose components are a principal identifier A together with an encrypted nonce $E(Kab : Na)$.

2.4 Reasoning about Protocols

To determine what a protocol actually achieves we need to be clear about the informal reasoning practices adopted by authentication protocol designers. The following are due to Burrows, Abadi and Needham [4]. Assume that you are principal A.

- If you've sent B a number that you have never used for this purpose before and if you subsequently receive from B something that depends on knowing that number, then you ought to believe that $B's$ message originated recently — in fact, after yours.
- If you believe that only you and B know the key K then you ought to believe that anything you receive encrypted with K as key comes originally from B — or from you!
- If you believe that Kb is $B's$ public key, then you should believe that any message you can decrypt with Kb comes originally from B.
- If you believe that only you and B know X then you ought to believe that any encrypted message that you receive containing X comes originally from B.

These principles are used repeatedly in the protocols we analyse in this paper.

3 Freshness Attacks

A freshness attack occurs when a message (or message component) from a previous run of a protocol is recorded by an intruder and replayed as a message (component) in the current run of the protocol. The classic example of such an attack occurs in the Needham Schroeder conventional (symmetric) key protocol.

The protocol was described in the seminal work by Needham and Schroeder in 1978 [15] and is described below:

$$
\begin{array}{lcl}
(1)\ A \rightarrow S & : & A, B, Na \\
(2)\ S \rightarrow A & : & E(Kas : Na, B, Kab, E(Kbs : Kab, A)) \\
(3)\ A \rightarrow B & : & E(Kbs : Kab, A) \\
(4)\ B \rightarrow A & : & E(Kab : Nb) \\
(5)\ A \rightarrow B & : & E(Kab : Nb - 1)
\end{array}
$$

In this protocol A requests from the server S a key to communicate with B. He includes a random number generated specially for this run of the protocol. This nonce Na will be used by A to ensure that message (2) is timely. S creates a key Kab and creates message (2). Only A can decrypt this message successfully since he possesses the key Kas. In doing so he will obtain the key Kab and check that the message contains the nonce Na. A passes on to B the encrypted message component $E(Kbs : Kab, A)$ as message (3).

Principal B decrypts this message to discover the key Kab and that it is to be used for communication with A. He then generates a nonce Nb, encrypts it (using the newly obtained key), and sends the result to A as message (4).

Principal A, who possesses the appropriate key Kab , decrypts it, forms $Nb - 1$, encrypts it and send the result back to B as message (5). B decrypts this and checks that the message is correct.

At the end of a correct run of the protocol, both principals should be in possession of the secret key Kab newly generated by the server S.

That is what the protocol is *intended to achieve*. In 1981, Denning and Sacco demonstrated that the protocol was flawed [7]. Consider message (3). Although B decrypts this message and (if it is indeed well-formed) assumes legitimately that it was created by the server S, there is nothing in the message to indicate that it was actually created by *S as part of the current protocol run*. Thus, suppose a previously distributed key $K'ab$ has been compromised (for example, by cryptanalysis) and is known to an intruder Z. Z might have monitored the network when the corresponding protocol

run was executed and recorded message (3) consisting of $E(Kbs : K'ab, A)$. He can now fool B into accepting the key as new by the following protocol.

$$\begin{array}{lll} (3)\ Z(A) \rightarrow B & : & E(Kbs : K'ab, A) \\ (4)\ B \rightarrow Z(A) & : & E(K'ab : Nb) \\ (5)\ Z(A) \rightarrow B & : & E(K'ab : Nb - 1) \end{array}$$

B believes he is following the correct protocol. Z is able to form the correct response in (5) because he knows the compromised key $K'ab$. He can now engage in communication with B using the compromised key and masquerade as A. Denning and Sacco suggested that the problem could be fixed by the inclusion of timestamps in relevant messages. The original authors suggested an alternative fix to this problem by means of an extra handshake at the beginning of the protocol [16].

4 Type Flaws

A message consists of a sequence of components each with some value (for example, the name of a principal, the value of a nonce, or the value of a key). The message is represented at the concrete level as a sequence of bits. A **type flaw** arises when the recipient of a message accepts that message as valid but imposes a different interpretation on the bit sequence than the principal who created it.

For example, consider the Andrew Secure RPC Protocol [**?**]:

$$\begin{array}{lll} (1)\ A \rightarrow B & : & A, E(Kab : Na) \\ (2)\ B \rightarrow A & : & E(Kab : Na + 1, Nb) \\ (3)\ A \rightarrow B & : & E(Kab : Nb + 1) \\ (4)\ B \rightarrow A & : & E(Kab : K'ab, N'b) \end{array}$$

Here, principal A indicates to B that he wishes to communicate with him and sends an encrypted nonce $E(Kab : Na)$ as a challenge in message (1). B replies to the challenge and issues one of his own by sending the message $E(Kab : Na + 1, Nb)$ in message (2). A replies to $B's$ challenge by forming and sending $E(Kab : Nb + 1)$ to B. B now creates a session key $K'ab$ and distributes it (encrypted) together with a sequence number identifier $N'b$ for future communication.

However, if the nonces and keys are both represented as bit sequences of the same length, say 64 bits, then an intruder could record message (2),

intercept message (3) and replay message (2) as message (4). Thus the attack looks like:

$$
\begin{array}{lll}
(1) & A \rightarrow B & : \; A, E(Kab : Na) \\
(2) & B \rightarrow A & : \; E(Kab : Na + 1, Nb) \\
(3) & A \rightarrow Z(B) & : \; E(Kab : Nb + 1) \\
(4) & Z(B) \rightarrow A & : \; E(Kab : Na + 1, Nb)
\end{array}
$$

Thus principal A may be fooled into accepting the nonce value $Na + 1$ as the new session key. The interpretations imposed on the plaintext bit string of the message are shown in figure 3.

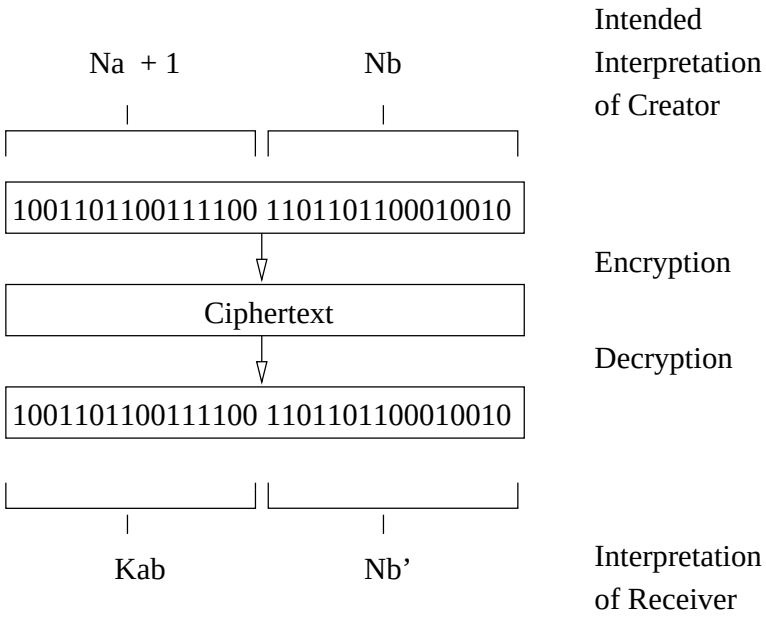

Figure 2: Bit Stream Interpretations and Type Flaw

If the nonces are random then the use of the nonce value as a key may not lead to a security compromise but it should be noted that nonces cannot be assumed to be good keys. Furthermore, nonces do not necessarily have to be random, just unique to the protocol run. Thus a predictable nonce might be used. In such cases A will have been fooled into accepting a key whose value is known to the intruder.

The above protocol is flawed in other ways too. For example, it is equally possible to record message (4) of a previous run and replay it in the current run, i.e. there is a freshness attack, as pointed out by Burrows, Abadi and Needham in their classic analysis work [4].

The Otway-Rees protocol [18] provides another example of a protocol subject to a type attack.

$$\begin{array}{lll} (1)\ A \rightarrow B & : & M, A, B, E(Kas : Na, M, A, B) \\ (2)\ B \rightarrow S & : & M, A, B, E(Kas : Na, M, A, B), E(Kbs : Nb, M, A, B) \\ (3)\ S \rightarrow B & : & M, E(Kas : Na, Kab), E(Kbs : Nb, Kab) \\ (4)\ B \rightarrow A & : & M, E(Kas : Na, Kab) \end{array}$$

The above protocol causes a key Kab created by the trusted server S to be distributed to principals A and B. M is a protocol run identifier.

After initiating the protocol A expects to receive a message back in (4) that contains the nonce Na he used in message (1) together with a new session key Kab created by S. If M is (say) 32 bits long, A and B each 16 bits long and Kab is 64 bits then an intruder Z can simply replay the encrypted component of message (1) as the encrypted component of message (4). Thus

$$\begin{array}{lll} (1)\ A \rightarrow Z(B) & : & M, A, B, E(Kas : Na, M, A, B) \\ (4)\ Z(B) \rightarrow A & : & M, E(Kas : Na, M, A, B) \end{array}$$

Here A decrypts $E(Kas : Na, M, A, B)$ checks for the presence of the nonce Na and accepts (M,A,B) as the new key. M, A and B are all publicly known (since they were broadcast in the clear). It is clear that an intruder can play the role of S in messages (3) and (4) simply replaying the encrypted components of message (2) back to B. The attack is:

$$\begin{array}{lll} (1)\ A \rightarrow B & : & M, A, B, E(Kas : Na, M, A, B) \\ (2)\ B \rightarrow Z(S) & : & M, A, B, E(Kas : Na, M, A, B), E(Kbs : Nb, M, A, B) \\ (3)\ Z(S) \rightarrow B & : & M, E(Kas : Na, M, A, B), E(Kbs : Nb, M, A, B) \\ (4)\ B \rightarrow A & : & M, E(Kas : Na, M, A, B) \end{array}$$

He can now listen in to conversation between A and B using the now publicly available key (M, A, B).

Further examples of type flaws are given by Syverson [24] and Hwang *et al* [11].

5 Parallel Session Attacks

A parallel session attack occurs when two or more protocol runs are executed concurrently and messages from one are used to form messages in another.

As a simple example consider the following one-way authentication protocol:

$$\begin{array}{lll} (1)\ A \rightarrow B & : & E(Kab : Na) \\ (2)\ B \rightarrow A & : & E(Kab : Na + 1) \end{array}$$

Successful execution should convince A that B is operational since only B could have formed the appropriate response to the challenge issued in message (1). In addition, the nonce Na may be used as a shared secret for the purposes of further communication between the two principals. In fact, an intruder can play the role of B both as responder and initiator. The attack works by starting another protocol run in response to the initial challenge.

$$\begin{array}{lll} (1.1)\ A \rightarrow Z(B) & : & E(Kab : Na) \\ (2.1)\ Z(B) \rightarrow A & : & E(Kab : Na) \\ (2.2)\ A \rightarrow Z(B) & : & E(Kab : Na + 1) \\ (1.2)\ Z(B) \rightarrow A & : & E(Kab : Na + 1) \end{array}$$

Here A initiates the first protocol with message (1.1). Z now pretends to be B and starts the second protocol run with message (2.1) which is simply a replay of mesage (1.1). A now replies to this challenge with message (2.2). But this is the precise value A expects to receive back in the first protocol run. Z therefore replays this as message (1.2). At the very least A believes that B is operational. In fact, B may no longer exist. The attack is illustrated in figure 4. Solid arrows indicate messages of the first protocol run, broken arrows indicate messages of the second protocol run.

In the above attack Z used principal A to do some work on his behalf. He needed to form an appropriate response to the encrypted challenge but could not do so himself and so he "posed the question" to A who provided the answer. A is is said to act as an **oracle** (because he always provides the correct answer) and attacks of this form are often called oracle attacks.

An interesting example of an oracle attack occurs in the Wide-Mouthed Frog Protocol (not intended for use in real systems) described by Burrows, Abadi and Needham [4].

$$\begin{array}{lll} (1)\ A \rightarrow S & : & A, E(Kas : Ta, B, Kab) \\ (2)\ S \rightarrow B & : & E(Kbs : Ts, A, Kab) \end{array}$$

Here, each principal (A and B in the above) shares a key with the server S. If A wishes to communicate with a principal B then he generates a key Kab and a timestamp Ta and forms message (1) which is sent to S.

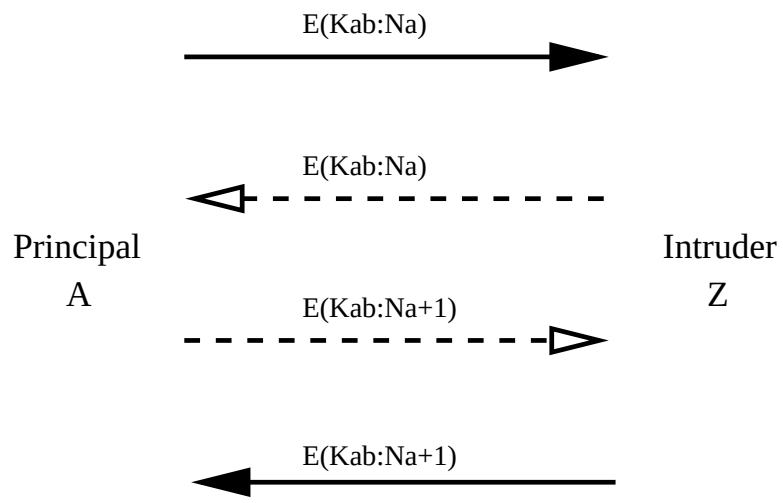

Figure 3: Simple Parallel Session Attack

On receiving message (1) S checks whether the timestamp Ta is "timely" and, if so, forwards the key to B with its own timestamp Ts. B checks whether the message (2) has a timestamp that is later than any other message it has received from S (and so will detect a replay of this message).

The first way it can be attacked is by simply replaying the first message within an appropriate time window - this will succeed since S will produce a new second message with an updated timestamp. If $S's$ notion of timeliness is the same as $B's$ (i.e. it accepts messages only if the timestamp is later than that of any other message it has received from the sender) then this attack will not work.

The second method of attack allows one protocol run to be recorded and then the attacker continuously uses S as an oracle until he wants to bring about reauthentication between A and B.

$$
\begin{array}{lrcl}
(1) & A \rightarrow S & : & A, E(Kas : Ta, B, Kab) \\
(2) & S \rightarrow B & : & E(Kbs : Ts, A, Kab) \\
(1') & Z(B) \rightarrow S & : & B, E(Kbs : Ts, A, Kab) \\
(2') & S \rightarrow Z(A) & : & E(Kas : T's, B, Kab) \\
(1'') & Z(A) \rightarrow S & : & A, E(Kas : T's, B, Kab) \\
(2'') & S \rightarrow Z(B) & : & E(Kbs : T''s, A, Kab)
\end{array}
$$

Z now continues in the above fashion until he wishes to get A and B to accept the key again. He does this by allowing A and B to receive messages intended for them by S.

There is some ambiguity in the available descriptions as to how timestamps are checked. It would seem sensible for a recipient A or B to impose some type of time window on the timestamps of messages received from S (as well as checking the message it has received from S is timestamped later than any other it has received from S). The efficacy of the attack is not compromised. Z simply plays ping-pong with S until it wants to rearrange authentication between A and B. Continuous use of S as a timestamp oracle ensures that all messages are sufficiently up to date.

Parallel session attacks abound in the literature [21, 25, 23, 11].

6 Implementation Dependent Attacks

Carlsen [5] indicates that some protocol definitions allow both secure and insecure implementations. Typing attacks could be prevented if the concrete representations of component values contained redundancy to identify a sequence of bits as representing a specific value of a specific type (and the principals made appropriate checks). Few protocol descriptions require such enforcement of types explicitly. Thus, the implementation approach adopted may severely affect the actual security of a protocol that actually conforms to the description. thus, we have the possibility of implementation-dependent attacks.

Similarly we saw in section 4 how the implementation of nonces (random or predictable) could severely affect the security of a protocol, though in that case it merely determined the degree of damage caused by an already flawed protocol.

Perhaps the most interesting (and least well understood) area where implementation dependent attacks may arise is the interaction between a specific protocol and the actual encryption method used. In the protocols we have described so far little has been said about the properties required of an encryption algorithm. We shall now show that the naïve use of certain algorithms (that are generally considered *strong*) in the context of specific protocols may produce insecure results.

6.1 Stream Ciphers

A stream cipher encrypts a plaintext bit stream on a bit by bit basis. The encrypted value of a particular bit may depend on the key K, random initialisation data R and the plaintext bits encrypted so far.

Consider the last two messages of the Needham Schroeder protocol

described in section 3.

$$(4)\ B \rightarrow A \ : \ E(Kab : Nb)$$
$$(5)\ A \rightarrow B \ : \ E(Kab : Nb - 1)$$

Suppose that the cipherstream for message (4) is $b_1 b_2 \ldots b_{n-1} b_n$. Now if Nb is odd then the final plaintext bit (assumed to be the least significant bit) will be 1 and $Nb - 1$ will differ only in that final bit. On a bit by bit encryption basis, the cipherstream for message (5) can be formed simply by flipping the value of the final bit b_n. On average the nonce will be odd half of the time and so this form of attack has a half chance of succeeding. This form of attack was originally described by Boyd [3].

6.2 Cipher Block Chaining

Another form of attack concerns the use of Cipher Block Chaining. Cipher Block Chaining (CBC) is a relatively good method of encrypting several blocks of data with an algorithm for encrypting a single block. It is one mode in which the widely used Data Encryption Standard (DES) can be employed. Block i of plain text is exclusively-ored with block $i - 1$ of ciphertext and is then encrypted with the keyed function e_K to form block i of ciphertext. A random block is used to initialise the process. For example, with initialisation block I we get:

$$E(K : P_1 P_2 \ldots P_n) = I C_1 C_2 \ldots C_n$$

where

$$C_0 = I$$
$$\forall i, i > 0 \bullet C_i = e(K : (C_{i-1} \oplus P_i))$$

Successive ciphertext blocks are decrypted using the keyed function d_K according to the rule

$$P_i = C_{i-1} \oplus d_K(C_i)$$

Thus, for any successive pair of ciphertext blocks we can recover the plaintext block corresponding to the second (provided we have the key).

Note that the message $C_1 C_2 C_3$ looks like a ciphertext that begins with initialisation block C_1, and decrypts to $P_2 P_3$. Similarly $C_1 C_2$ decrypts to P_2 (it uses C_1 as an initialisation block) and $C_2 C_3$ decrypts to P_3.

A good text on cryptography, such as that by Schneier [20], should be consulted for further details.

Thus we can see that without appropriate additional protection valid messages may be created if their contents are subsequences of generated messages. To distinguish this form of attack from those that follow we shall call this form of flaw a *subsequence flaw*.

Consider again message (2) of the Needham Schroeder protocol of section 3.

$$(2)\ S \rightarrow A \quad : \quad E(Kas : Na, B, Kab, E(Kbs : Kab, A))$$

Suppose that this has ciphertext $C_0C_1C_2C_3\ldots$ and that all components have length one block. Then $E(Kas : Na, B) = C_0C_1C_2$. But such a message is of the form A might expect to receive in message (3) when B has initiated the protocol. Thus, he can be fooled into accepting the publicly known Na as a key. Thus simple use of CBC mode of encryption with this protocol will not suffice.

Stubblebine and Gligor [22] have demonstrated attacks via *cut and paste* methods where the ciphertexts of messages are split and conjoined appropriately to form the ciphertexts of other messages (which should only be formable by those in possession of the appropriate key). This is illustrated in figure 5.

We see that the spliced ciphertext message decrypts to appropriate plaintext except for the block immediately after the join. Denoted by X in the figure, it is likely that it is random gibberish but in some cases that may be precisely what is expected (e.g. if the block is expected to contain a random number). Mao and Boyd have also highlighted the dangers of CBC use [13] pointing out that in many cases it will be possible to determine precisely what value X takes if the intruder has knowledge of the plaintext block corresponding to the ciphertext immediately after the ciphertext join. In the example shown in figure 5, we have

$$X = C_3 \oplus d_K(C_2')$$

$$X = C_3 \oplus (C_1' \oplus P_2')$$

and so if P_2' is known then so is X since the ciphertext blocks are publicly broadcast.

It is dangerous to believe that attacks of the above form lose their power if the plaintext block is not publicly known or guessable; such blocks will generally be known to the parties communicating in a protocol who may misuse their knowledge (see below).

Of particular note are *initialisation attacks* — attacks that involve modulation of the initialisation vector C_0. Consider a ciphertext that starts with

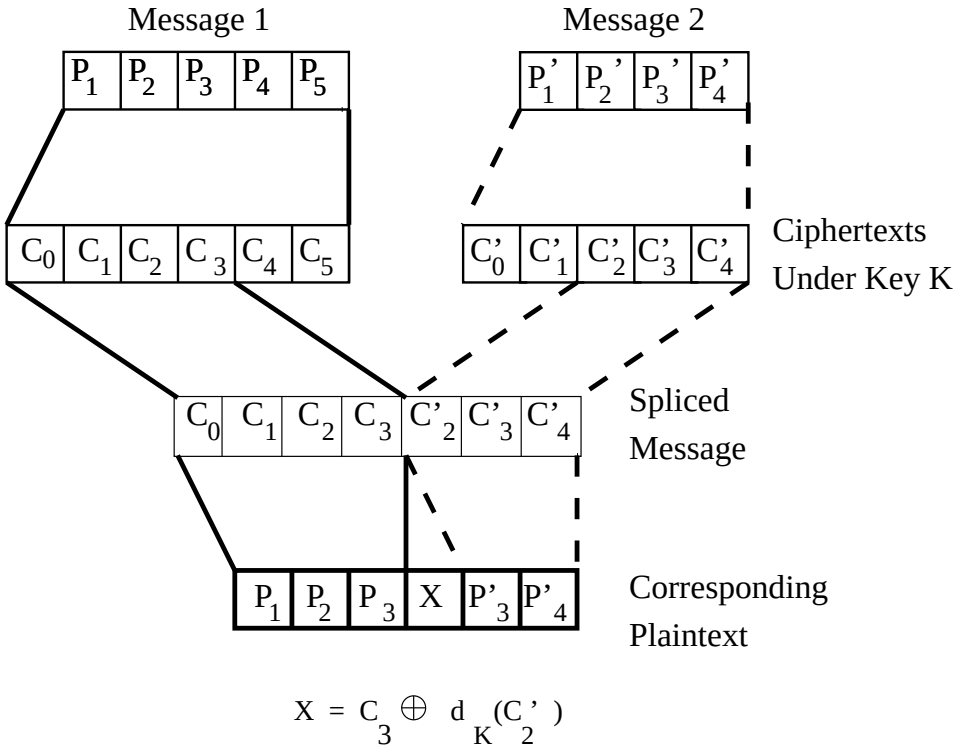

Figure 4: Splicing Attack on Cipher Block Chaining

C_0C_1 and suppose that we know that the initial plaintext block was P_1. Then

$$P_1 = C_0 \oplus d_K(C_1)$$

Now for any desired block value W we have i

$$W = W \oplus P_1 \oplus P_1$$

since anything exclusively-ORed with itself is 0. And so we have (substituting for the second P_1)

$$W = W \oplus P_1 \oplus (C_0 \oplus d_K(C_1))$$

and so

$$W = C_0' \oplus d_K(C_1)$$

where $C_0' = W \oplus P_1 \oplus C_0$ and so $C_0'C_1$ is the ciphertext corresponding to plaintext W. In this fashion we can replace the initial known plaintext block P_1 with our own choice W. This is potentially very disturbing since the rest of the message is unaffected.

As an example of the danger of this attack, consider again message (2) of the Needham Schroeder protocol. We can record message (2) of a previous run of this protocol between A and B. In particular we can replay the old message (2) after modifying the initial block from the old (and known) value of the nonce Na with the new one issued in the current run of the protocol. Thus, we can impersonate the trusted server S. Now consider the contents of message (3) of that protocol:

$$(3)\ A \rightarrow B \quad : \quad E(Kbs : Kab, A)$$

Since A knows the key in message (3), he can create a new message (3) whenever he likes for any key value he likes. One might argue that if A wants to misbehave he can do so much more simply than this (by simply broadcasting information he is expected to keep secret) but this misses the point: B works on the assumption that the contents of message (3) were created by the trusted server S. This is clearly not the case.

We have illustrated these attacks using the Needham Schroeder protocol simply because it is the best known and simple to understand. The authors have looked at many protocols. The above forms of attack present problems with nearly all of them.

We have illustrated various forms of cryptoalgorithm dependent flaws. The above is by no means exhaustive. Indeed, other mdes of encryption have given rise to problems in implemented protocols. In particular, Propagating Cipher Block Chaining (PCBC) mode was shown to be deficient and lead to the Kerberos V.5 protocol adopting CBC mode (V.4 used PCBC). Criticisms of Kerberos were given by Bellovin and Merritt [1].

7 Binding Attacks

In public key cryptography the integrity of the public keys is paramount. Suppose your public key is Ky and an intruder's public key is Ki. The intruder is able to decrypt any messages encrypted with Ki. Principals wishing to convey information to you secretly will encrypt using what they believe is your public key. Thus, if the intruder can convince others that your public key is Ki then they will encrypt secret information using Ki and this will be readable by the intruder.

Thus, the principals in charge of distributing public keys must ensure that the above cannot occur; there must be a verifiable binding between a public key and the corresponding agent. In some authentication protocols, this has not been achieved. Consider the following protocol:

$$\begin{array}{lll} (1)\ C \rightarrow AS & : & C, S, Nc \\ (2)\ AS \rightarrow C & : & AS, E(Kas^{-1} : AS, C, Nc, Ks) \end{array}$$

Here, a prospective client C wants to communicate with S and needs the public key of S. The *authentication server* AS is the repository for principals' public keys. C sends a message (1) to request the public key of S. He includes a nonce Nc to ensure the freshness of the expected reply.

AS replies with message (2). The principal identifier AS is sent in the clear to tell C which public key to use to decrypt the following ciphertext. The components of the encrypted part signify that the message was created by AS, that this message has been created in response to a request from a client C with nonce Nc and that the public key requested is Ks. However, the reader may note that there is nothing in the encrypted part of message (2) that ensures the recipient that the key is really the public key of S. This leads to the following parallel session attack:

$$\begin{array}{lll} (1.1)\ C \rightarrow Z(AS) & : & C, S, Nc \\ (2.1)\ Z(C) \rightarrow AS & : & C, Z, Nc \\ (2.2)\ AS \rightarrow Z(C) & : & AS, E(Kas^{-1} : AS, C, Nc, Kz) \\ (1.2)\ Z(AS) \rightarrow C & : & AS, E(Kas^{-1} : AS, C, Nc, Kz) \end{array}$$

Here the intruder Z intercepts the initial message from C to AS and simply replaces the identifier of the intended server S with his own identifier Z. and sends the result to AS as message 2.1. AS, believing that C has requested Z's public key, replies with message (2.2). Z simply allows this to be received by C as message (1.2). C performs appropriate decryptions and checks and believs that he has received the public key of S. This attack (and a similar one) was identified by Hwang and Chen [10]. They suggest

that this problem can be solved by explicitly including the identifier of the requested server S in message (2). The protocol then becomes:

(1) $C \rightarrow AS \quad : \quad C, S, Nc$
(2) $AS \rightarrow C \quad : \quad E(Kas^{-1} : AS, C, Nc, S, Ks)$

8 Other Forms of Attack

The above forms of attack may be regarded as representative of the dangers involved in designing authentication protocols. In general, they do not require a great deal of mathematical sophistication to comprehend. More sophisticated attacks that take advantage of particular algebraic properties of the cryptoalgorithm when used in the context of authentication protocols are given in the excellent paper by Judy Moore [14].

In addition, more traditional forms of attack such as cryptanalysis can be launched on several protocols. Mao and Boyd [12] have recently investigated ways of protecting against such attacks. Aspects of redundancy have also been addressed by Gong [8]. Protocols using passwords have been addressed by several authors [2] [9]. Carlsen [5] has a category called *elementary flaws* which is used to group protocols which are breakable with little effort because they provide little or no protection. The (unintentionally flawed) CCITT X.509 protocol and the (intentionally flawed — it was intended as a example to highlight deficiencies in the use of BAN logic) Nessett protocol [17] are included in this category. It is a matter of opinion as to when a flaw is considered elementary and the choice is somewhat arbitrary. We have discovered a flaw similar to the CCITT X.509 one in a recently published protocol [6].

9 Conclusions

Protocol definition might seem a simple task; protocols often comprise only a few messages. This is, however, clearly deceptive and the examples we have shown above indicate that the definition of secure protocols is a remarkably subtle affair.

The current explosion in distributed system development and network usage means that there is a pressing need for a framework and tool support for the rigorous development and analysis of new security protocols. Although significant advances have been made in recent years, there is clearly some way to go!

Acknowledgements

This work was completed under contract to the Defence Research Agency as part of the Security Protocols Strategic Research Plan run by Peter Ryan.

The authors would also like to acknowledge the contribution of Gavin Lowe of the Programming Research Group at Oxford for several illuminating email conversations.

John Clark is the CSE Lecturer in Safety Critical Systems at the University of York a post sponsored in part by CSE Ltd. The views in this paper are the authors' own and do not necessarily reflect the views of CSE Ltd.

References

[1] S. M. Bellovin and M. Merritt. Limitations of the Kerberos Authentication System. *Computer Communication Review*, 20(5):119–132, October 1990.

[2] S. M. Bellovin and M. Merritt. Encrypted Key Exchange: Password based protocols secure against dictionary attacks. In *Proceedings 1992 IEEE Symposium on Research in Security and Privacy*, pages 72–84. IEEE Computer Society, May 1992.

[3] Colin Boyd. Hidden Assumptions in Cryptographic Protocols. *Proceedings of the IEE*, 137 Pt E(6):433–436, November 1990.

[4] Michael Burrows, Martin Abadi, and Roger Needham. A Logic of Authentication. Technical Report 39, Digital Systems Research Center, February 1989.

[5] Ulf Carlsen. Cryptographic Protocol Flaws. In *Proceedings 7th IEEE Computer Security Foundations Workshop*, pages 192–200. IEEE Computer Society, 1994.

[6] John Clark and Jeremy Jacob. On The Security of Recent Protocols. *Information Processing Letters*, 56(3):151–155, November 1995.

[7] Dorothy Denning and G. Sacco. Timestamps in Key Distribution Protocols. *Communications of the ACM*, 24(8), August 1981.

[8] Li Gong. A Note on Redundancy in Encrypted Messages. *Computer Communication Review*, 20(5):18–22, October 1990.

[9] Li Gong, A. Lomas, R. Needham, and J. Saltzer. Protecting Poorly Chosen Secrets from Guessing Attacks. *IEEE Journal on Selected Areas in Communications*, 11(5), jun 1993.

[10] Tzonelih Hwang and Yung-Hsiang Chen. On the security of SPLICE/AS: The authentication system in WIDE Internet. *Information Processing Letters*, 53:97–101, 1995.

[11] Tzonelih Hwang, Narn-Yoh Lee, Chuang-Ming Li, Ming-Yung Ko, and Yung-Hsiang Chen. Two Attacks on Neuman-Stubblebine Authentication Protocols. *Information Processing Letters*, 53:103–107, 1995.

[12] Wenbo Mao and Colin Boyd. Towards the Formal Analysis of Security Protocols. In *Proceedings of the Computer Security Foundations Workshop VI*, pages 147–158. IEEE Computer Society Press, 1993.

[13] Wenbo Mao and Colin Boyd. Development of Authentication Protocols: Some Misconceptions and a New Approach. In *Proceedings 7th Computer Security Foundations Workshop*, pages 178–186. IEEE Computer Society Press, 1994.

[14] Judy H. Moore. Protocol Failures in Cryptosystems. *Proceedings of the IEEE*, 76(5), May 1988.

[15] Roger Needham and Michael Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21(12), December 1978.

[16] Roger M. Needham and M. D. Schroeder. Authentication Revisited. *Operating Systems Review*, 21(7):7–7, January 1987.

[17] Daniel M. Nessett. A Critique of the Burrows, Abadi and Needham Logic. *ACM Operating Systems Review*, 24(2):35–38, April 1990.

[18] D. Otway and O. Rees. Efficient and Timely Mutual Authentication. *Operating Systems Review*, 21(1):8–10, January 1987.

[19] R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[20] Bruce Schneier. *Applied Cryptography*. Wiley, 1994.

[21] E. Snekkenes. Roles in Cryptographic Protocols. In *Proceedings of the 1992 IEE Symposium on Security and Privacy*. IEEE Computer Society Press, 1992.

[22] Stuart G. Stubblebine and Virgil D. Gligor. On Message Integrity in Cryptographic Protocols. In *Proceedings of the 1992 IEEE Symposium on Research in Security and Privacy*, pages 85–104. IEEE, 1992.

[23] Paul Syverson. A Taxonomy Of Replay Attacks. In *Proceedings of the 7th IEEE Computer Security Foundations Workshop*, pages 131–136. IEEE Computer Society Press, 1994.

[24] Paul Syverson. On Key Distribution for Repeated Authentication. In *Operating Systems Review*, pages 24–30, 1994.

[25] T. Y. C. Woo and S. S. Lam. A Lesson on Authentication Protocol Design. *Operating Systems Review*, pages 24–37, 1994.