Available online at www.sciencedirect.com
SCIENCE DIRECT®

Electronic Notes in Theoretical Computer Science

ELSEVIER Electronic Notes in Theoretical Computer Science 128 (2005) 87–104
www.elsevier.com/locate/entcs

# Analysing the Vulnerability of Protocols to Produce Known-pair and Chosen-text Attacks

Steve Kremer [1]

*School of Computer Science*
*University of Birmingham*
*Birmingham, UK*

Mark D. Ryan [2]

*School of Computer Science*
*University of Birmingham*
*Birmingham, UK*

**Abstract**

In this paper we report on an analysis for finding known-pair and chosen-text attacks in protocols. As these attacks are at the level of blocks, we extend the attacker by special capabilities related to block chaining techniques. The analysis is automated using Blanchet's protocol verifier and illustrated on two well-known protocols, the Needham-Schroeder-Lowe public-key protocol as well as the Needham-Schroeder symmetric-key protocol. On the first protocol, we show how the special intruder capabilities related to chaining may compromise the secrecy of nonces and that chosen-ciphertext attacks are possible. We propose two modified versions of the protocol which strengthen its security. We then illustrate known-pair and chosen-plaintext attacks on the second protocol.

*Keywords:* Formal analysis of security protocols, known-pair and chosen-text attacks, block chaining.

## 1 Introduction

Computer security has gained increasing importance with the overwhelming growth of the Internet and electronic commerce. However, history has shown

[1] Email: S.Kremer@cs.bham.ac.uk
[2] Email: M.D.Ryan@cs.bham.ac.uk

1571-0661/$ – see front matter © 2005 Published by Elsevier B.V.
doi:10.1016/j.entcs.2004.11.043

that security protocols are extremely error-prone and the need for applying formal methods to security systems has been widely accepted. One of the most well-known examples is the flaw Lowe [9] found in the Needham-Schroeder public-key protocol [15] presented 17 years earlier. The protocol was believed and even proved correct before (although one has to admit that Lowe changed some, probably unrealistic, hypotheses).

In 1983, Dolev and Yao [6] set up the foundations for applying formal methods to security protocols by defining what has been known since then as the Dolev-Yao model. This model is based on the following two hypotheses:

- the intruder has complete knowledge of and access to the communication network: he can remove any sent message and insert any messages he can construct;
- the cryptography is perfect, e.g. unless you know the key, it is impossible to decrypt an encrypted message.

This model has been well studied during the last twenty years. In one direction, many theoretical results, related to the decidability and complexity of the verification, have been established, e.g. [24,7]. In another direction, a large number of methods and (partially or completely automated) tools have been used and developed, for instance [1,8,9,12,16,19].

While the first hypothesis of the model is widely accepted, the perfect encryption assumption is considered as too strong. For instance, many encryption schemes present *algebraic properties* which an attacker might take advantage of. Examples include the multiplicative property of the RSA encryption scheme, i.e. $(m_1^e \mod n) \times (m_2^e \mod n) = (m_1 \times m_2)^e \mod n$, the fact that Diffie-Hellman key agreement is based on the commutativity of exponentiation, i.e. $g^{ab} = g^{ba}$, or that an encryption based on “exclusive-or” cancels itself out, i.e. $a \oplus b \oplus b = a$. Many recent papers aim at extending the Dolev-Yao model with these additional properties [2,3,14,17].

Another kind of attack which is not captured in a standard Dolev-Yao model are *guessing attacks* [4,5,10]. In a guessing attack an intruder tries to break a weak password, by trying to *guess* it. A guessing attack is possible whenever the protocol offers the possibility to verify that a previously made guess was correct. A similar kind of cryptanalysis, although not restricted to weak data, can be performed when a protocol offers the possibility to obtain *known-pairs* or *chosen-texts*. In a known-pair attack, an attacker gets knowledge of both a plaintext and the corresponding ciphertext. This information may then be useful to a cryptanalyst when trying to obtain some information about the key. More powerful attacks are chosen-plaintext and chosen-ciphertext attacks, where the attacker can choose the plaintext for which he gets to know the encryption or vice-versa. Known plaintext attacks were for instance used in WWII, to break Enigma keys. Cryptanalysts used the fact that the weather report generally contained the word *Wetter* [3]. Another example of a cryptosystem which is vulnerable to known plaintext attacks is when exclusive-or is used as the encryption operator. Knowledge of one known pair is enough to recover the secret key (although one should never use the same key twice in this cryptosystem). Most modern ciphers resist against this kind of attack. However, Shoup showed in [18] that the proof of resistance against chosen-ciphertext attacks of OAEP was flawed. Known and chosen-ciphertexts can also be used to mount (offline) dictionary attacks. Given a known pair encrypted under a weak encryption key, e.g. password, an attacker can try to encrypt the plaintext with keys from a list or dictionary and compare the results. When given a chosen-text text, the attacker prepares (once and for all) a list of one text encrypted (or decrypted) with a list of possible keys. Then, the attacker can choose to obtain the encryption (or decryption) of this text and try to find the result in his list to recover the key. This would be a special case of guessing attacks.

We believe that it is important to make explicit the hypotheses made on a cryptosystem used in a protocol, rather than treating it as a black box. A protocol makes chosen-plaintext (ciphertext) attacks possible if the attacker can use the protocol as an encryption (decryption) oracle. Even, if this might not always lead to an attack the presence of such an oracle should be analysed: an attacker could for instance mount an attack by mixing up two different protocols, using the same keys, each of which might be secure independently.

In this paper we aim to develop an automated method for determining whether a protocol offers the possibility to mount known-pair or chosen-text attacks on the underlying encryption scheme. We do this in the framework of block ciphers. In such ciphers encryption is performed at the level of blocks, and we need to take into account what technique is used to *chain* the different blocks. These techniques offer some additional attacker capabilities, allowing to “reuse” pieces of encrypted messages to obtain other valid ciphertexts.

## Related work

The work closest to ours is Stubblebine and Meadows’ analysis of known-pair and chosen-text attacks [21,22]. They use the NRL protocol analyser to automate their analysis and show how to automatically find a known attack in an early version of the IP Encapsulating Security Protocol. One major difference with our work is that Stubblebine and Meadows model cipher block

[3] the German word for weather

chaining at a low level, modelling the exclusive-or operation. However, due to the limitations of the NRL Protocol analyser, associative-commutative properties are not included. In this paper, we give a more abstract view of chaining modes, where special capabilities of the intruder related to chaining are represented as deduction rules. Moreover, we use a free term algebra in opposition to the rewrite system used by the protocol analyser. This results in some of the technical differences, which we will discuss when explaining our model. To the best of our knowledge, Stubblebine and Meadows' work is the only automated analysis of known-pair and chosen-text attacks. There exist several other works, such as [11,20], which emphasise problems that can arise when chaining modes are used without additional integrity mechanisms. However, these works do not use automated tools to find such attacks.

### Outline

In section 2 we give the preliminaries, including notation, definitions of known-pair and chosen-pair attacks. We also explain block chaining techniques and show how an attacker can use special properties of these techniques. In section 3, we briefly present Blanchet's protocol verifier, which we used to automate our analysis. We define known-pair and chosen-text attacks as queries given to the tool and encode special attacker capabilities, related to block chaining, in the protocol verifier. In section 4, we illustrate our techniques on two well-known protocols, the Needham-Schroeder-Lowe public-key protocol and the Needham-Schroeder symmetric-key protocol. We demonstrate the ability of the tool to find known-pair and chosen-text attacks, as well as a secrecy attack, related to the extended attacker capabilities. To facilitate others in replicating our work and possibly developing it further, we have made our analyses electronically available[4]. Finally, in section 5, we conclude.

## 2 Known-pair, chosen-text attacks and block chaining

In this section we give the basic notations and introduce known-pair and chosen-text attacks. We discuss them in the framework of block ciphers, which can be either symmetric or public key cryptosystems. When using a block cipher, the plaintext is divided into different blocks which are encrypted one by one. These encrypted blocks are composed, or *chained* using special techniques to form the ciphertext. As known pairs or chosen texts have to be found at the level of blocks, we discuss some of them, and show how an attacker could take advantage of some properties of these chaining techniques.

[4] http://www.ulb.ac.be/di/scsi/skremer/Pairs/

### 2.1 Notation

In the remainder of the paper, we use the following notation. $A$, $B$, $I$ represent the entities Alice, Bob and the intruder. The expression $\mathsf{pk}(k)$ denotes the public key corresponding to the secret key $k$; $\mathsf{sE}_k(m)$ and $\mathsf{sD}_k(c)$ are the symmetric encryption and decryption (resp.) of $m$ using the key $k$. The public-key decryption and encryption of $m$ using the key $k$ and its public counterpart $\mathsf{pk}(k)$ are written $\mathsf{pD}_k(c)$ and $\mathsf{pE}_{\mathsf{pk}(k)}(m)$. We write $\mathsf{h}(m)$ for the cryptographic hash of message $m$.

### 2.2 Known-pair and chosen-text attacks

Generally, the security of a cipher is evaluated with regard to the information a cryptanalyst is provided (we assume that according to Kerkhoff's assumption, the attacker knows all the details about the encryption function, except the secret key). Then, there exist different classes of attacks:

- ciphertext-only: the attacker has no information about the plaintext at all;
- known-plaintext: the attacker knows both the plain text and the ciphertext;
- chosen-plaintext: the attacker can choose the plaintext for which he wants to know the cipher;
- chosen-ciphertext: the attacker can choose the cipher for which he wants to know the plaintext.

Note, that when considering public-key encryption, an encryption scheme should always at least hold against known- and chosen-plaintext attacks, as these can trivially be realized, as the encryption key is public knowledge. A variant of chosen-text attacks are *adaptively* chosen-text attacks. The difference is that when a text is chosen, it may depend on the previous choices. Whenever a protocol can be executed several times, it offers the possibility of adaptively chosen-text attacks whenever chosen-text attacks are possible.

In the context of protocol analysis, we want to examine what are the exact assumptions a protocol designer has to make about the use of the encryption functions. Rather than merely considering encryption functions as black boxes, we want to quantify against which of the above attacks the used cipher should resist. Therefore, we want to verify whether a given protocol offers the possibility to mount one of the above attacks. Note that we do not pretend that the presence of a known-pair or chosen-text attack breaks the protocol. If the protocol offers such a possibility, the used encryption function should explicitly be stated to resist against these kind of attacks.

### 2.3 Block chaining techniques

Electronic Code Book (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), and Output Feedback (OFB).

When using block ciphers (whether for symmetric or public-key cryptography), the plaintext is first divided into blocks of fixed size, corresponding to the blocksize of the encryption function. These blocks are then ciphered one by one. Classical block chaining mechanisms are ECB (*electronic code book*), CBC (*cipher block chaining*), OFB (*output feedback*) and CFB (*cipher feedback*). However, as noted in [13] p. 285, OFB and CFB cannot be applied to public-key encryption. Therefore we focus on ECB and CBC. We illustrate block chaining on symmetric key encryption, but the techniques directly apply to public-key encryption as well. The simplest chaining is ECB. For a plaintext $P_1 \dots P_n$ of size $n$ × blocksize, the corresponding ciphertext $C_1 \dots C_n$ under key $k$ is computed by setting

$$C_i = \mathsf{sE}_k(P_i) \quad (1 \leq i \leq n).$$

CBC represents a more elaborate way of chaining. In CBC the cipher of each block depends on the cipher of the previous one and is computed as follows:

$$C_i = \mathsf{sE}_k(P_i \oplus C_{i-1}) \quad (1 \leq i \leq n)$$

where $C_0 = IV$, an initialisation vector. In order to decrypt such a message we compute

$$P_i = \mathsf{sD}_k(C_i) \oplus C_{i-1}$$

where $\oplus$ denotes bitwise exclusive-or, which cancels itself out. According to [13], the secrecy of the $IV$ is not required for security, although secrecy might be used to ensure $IV$'s integrity. In the following, we suppose that the $IV$ is not secret and no additional mechanism is used to ensure its integrity, or the integrity of the ciphered message. We defend the point of view that if an additional mechanism is needed to ensure the integrity of the message, such as a cryptographic hash for instance, then it should be stated explicitly in the protocol description.

We now explain how an attacker could tamper with ciphered messages that use either ECB or CBC. It is easy to see that if ECB is used, an attacker can extract any ciphered block and compose a new sequence. Given a ciphertext $C_1 \dots C_n$, he can for instance extract the cipher $C_i \dots C_j$ $(1 \leq i \leq j \leq n)$ which corresponds to a correct cipher of $P_i \dots P_j$. In a similar way, the attacker could delete some of the blocks and obtain $C_1 \dots C_i C_j \dots C_n$ $(1 \leq i < j \leq n)$, corresponding to a valid cipher of $P_1 \dots P_i P_j \dots P_n$. The attacker could even recompose parts of different messages that were ciphered with the same key. Moreover, when both $C_i$ and $P_i$ are known, we immediately have a pair $P_i, \mathsf{sE}_m(P_i)$, which is not the case when using CBC due to the exclusive-or operation.

When CBC is used, altering messages and obtaining pairs might seem more difficult. However, as we will show now, there are several properties that can be exploited. A property that has already been noticed and automated in [23], is that any prefix of a valid cipher is also a valid cipher: given $C_1 \dots C_n$, $C_1 \dots C_i$ $(1 \leq i \leq n)$ is a valid cipher corresponding to the encryption of the $i$ first blocks. It might be less obvious that one can also reuse any postfix of a cipher as a valid cipher. All we need to do is to set the $IV$ to $C_i$ and we obtain a valid cipher $C_{i+1} \dots C_n$ corresponding to the encryption of the $n - i$ last plaintext blocks. There is an additional property which can be exploited when a public-key encryption scheme is used. Given any cipher $C_1 \dots C_i$ of a plaintext $P_1 \dots P_i$, we can obtain a valid cipher $C_1 \dots C_n$ corresponding to the plaintext $P_1 \dots P_n$ where the last $n - i$ plaintext blocks are freely chosen. Doing so yields into a valid ciphertext for which we do not know the beginning of the plaintext, i.e. the $i$ first blocks. Note that this is only possible because the encryption key is known. It would not work when using symmetric encryption, unless we are in the uninteresting case where the encryption key, and hence the decryption key too, would be known. However, there exists a similar property, which also works in the symmetric case. Given $C_1, \dots C_n$ and $C'_1 \dots C'_{n'}$, the attacker can concatenate these two ciphers. The decryption results into $P_1 \dots P_n, X, P'_2 \dots P'_{n'}$, where $X = \mathsf{sD}_k(C'_1) \oplus C_n$ is a random-looking block as it is not properly decrypted. In [11] this property has been exploited to find new flaws, as $X$ could be accepted as being a fresh nonce or key.

In addition to the fact that using CBC gives new capabilities to the attacker, we will show how known-pair or chosen-text attacks can be found when CBC is used. As $C_i \neq \mathsf{sE}_k(P_i)$, obtaining a pair is not direct. However, if we know $P_i$, $C_i$ and $C_{i-1}$, we can compute the decryption of $C_i$ as

$$\mathsf{sD}_k(C_i) = P_i \oplus C_{i-1}$$

and obtain a known pair. This same technique can be used in a chosen-ciphertext attack to recover the decryption of the block. In a chosen-plaintext attack, we want to get the encryption of a given block $P_i$. To do this we have to insert $P'_i = P_i \oplus C_{i-1}$. Indeed, when applying the encryption algorithm with CBC to the block $P'_i$, we obtain

$$\mathsf{sE}_k(P'_i \oplus C_{i-1}) = \mathsf{sE}_k(P_i).$$

The “tricks” described above are not useful in practice unless we make some assumptions related to the size of blocks and message fields. In the following, we assume that the size of all fields is a multiple of the blocksize. This means that we can always "cut" message at the right place and that there is never a need for padding to achieve the right blocksize. Although this assumption might be unrealistic, we want to verify that the security of the protocol holds in a worst case behaviour and does not depend on unstated assumptions of the implementation. Also note that when public-key cryptography is used, a 1024-bit RSA block is unlikely to be of the same size as a nonce. However, public-key cryptography based on elliptic curves uses much smaller key and block sizes, making the size match much more realistic.

Under these assumptions, we conclude that when knowing a ciphertext and the corresponding plaintext, it is always possible to get the plaintext for each block and the corresponding encryption. Moreover, given a chosen cipher block, we know how to integrate it correctly in a message, such that, when obtaining the decryption of the message, we can extract the decryption of the chosen block. In section 3, we show how we extend the attacker's capabilities in an abstract way, where we do not need to actually model the functioning of CBC.

## 3 The protocol verifier tool and our model

In this section, we briefly introduce the tool which we use to automate our analysis. We use Blanchet's protocol verifier tool [1], which is based on Prolog rules and a specialised solving algorithm, which terminates for many protocols. Although, we believe that many other tools could easily be extended to perform a similar analysis our choice was guided by the following arguments: the protocol verifier does not bound the number of sessions, is very efficient and above all allows us to encode the attacker capabilities without need of modifying the tool itself.

### *3.1 Protocol description*

Each protocol is described by a set of Prolog rules where protocol messages are composed of terms. For a detailed syntax, we refer the reader to [1].

Cryptographic primitives are represented by functions. As an example, encryption is modeled as a function taking two parameters, the plaintext and the key. The cryptosystems are modelled in terms of a free term algebra, which means that we do not explicitly model decryption. Only explicitly encrypted terms can be decrypted.

Secret keys are names and nonces are names parametrised by the messages previously received in the given protocol session, in order to model fresh name generation (even if this is slightly weaker than really creating a new name).

Public keys are represented by a function, which maps a private key to the corresponding public key, i.e. whenever one knows the secret keys, the public key can be deduced.

There is also a special predicate att(m), which is interpreted as *"the attacker knows message m"*.

The protocol messages themselves are encoded as Prolog rules, one for each message. The intuition is that the intruder completely controls the network and each message is sent to him. Hence, each message foreseen by the protocol allows the attacker to increase his knowledge, provided that the attacker has knowledge of all the preceding messages this entity should have received in the protocol. This is required to ensure that a honest protocol entity cannot execute the protocol out of order. As an example, consider the classical Needham-Schroeder public key protocol fixed by Lowe [9], in its three message version, as described in protocol 1.

**Protocol 1** Needham-Schroeder-Lowe public key protocol

1. A → B: $\mathsf{pE}_{\mathsf{pk}(B)}(N_A, \mathsf{pk}(A))$
2. B → A: $\mathsf{pE}_{\mathsf{pk}(A)}(N_A, N_B, \mathsf{pk}(B))$
3. A → B: $\mathsf{pE}_{\mathsf{pk}(B)}(N_B)$

This protocol description gives us the following set of rules, corresponding to each of the three messages.

$$\frac{\mathsf{att}(\mathsf{pk}(x))}{\mathsf{att}(\mathsf{pE}_{\mathsf{pk}(sB[])}(N_A[\mathsf{pk}(x)], \mathsf{pk}(sA[])))} \quad \frac{\mathsf{att}(\mathsf{pE}_{\mathsf{pk}(sB[])}(x, y))}{\mathsf{att}(\mathsf{pE}_y(x, N_B[x, y], \mathsf{pk}(sB[])))}$$

$$\frac{\mathsf{att}(\mathsf{pk}(x)) \wedge \mathsf{att}(\mathsf{pE}_{\mathsf{pk}(sA[])}(N_A[\mathsf{pk}(x)], y, \mathsf{pk}(x)))}{\mathsf{att}(\mathsf{pE}_{\mathsf{pk}(x)}(y))}$$

In the first rule, corresponding to the first message, we require the attacker to know some public key of an entity $x$. This reflects the fact that Alice is willing to engage a session with any other entity, including possibly the attacker himself. Therefore, we allow the attacker to trigger Alice's protocol execution, choosing with whom she starts the protocol. When doing so, the attacker learns the first message of the protocol. We denote Alice's (respectively Bob's private key) by the names $sA[]$ (respectively $sB[]$). Note that the nonce $N_A$ is parametrised by $\mathsf{pk}(x)$ to ensure that a different nonce is used for each session. Also note that we model entity names by their public keys, which is a possible implementation of the protocol.

The second rule says, that whenever the intruder can provide a pair $(x, y)$ encrypted with Bob's public key, Bob will respond by sending the second message of the protocol, interpreting $x$ as the initiator's nonce and $y$ as the initiator's private key. The interpretation of the third rule is similar.

$$\frac{\mathsf{att}(m)}{\mathsf{att}(\mathsf{h}(m))} \qquad \frac{\mathsf{att}(m) \wedge \mathsf{att}(k)}{\mathsf{att}(\mathsf{sE}_k(m))} \qquad \frac{\mathsf{att}(\mathsf{sE}_k(m)) \wedge \mathsf{att}(k)}{\mathsf{att}(m)}$$

$$\frac{\mathsf{att}(x)}{\mathsf{att}(\mathsf{pk}(x))} \qquad \frac{\mathsf{att}(m) \wedge \mathsf{att}(\mathsf{pk}(k))}{\mathsf{att}(\mathsf{pE}_{\mathsf{pk}(k)}(m))} \qquad \frac{\mathsf{att}(\mathsf{pE}_{\mathsf{pk}(k)}(m)) \wedge \mathsf{att}(k)}{\mathsf{att}(m)}$$

**Fig. 1:** Standard attacker capabilities

### *3.2 Attacker capabilities*

We now define what are the capabilities of the attacker to construct and decompose messages in a Dolev-Yao style. The capabilities of the intruder are encoded as Prolog rules in a similar way as the protocol messages themselves.

In figure 1, we present the attacker rules corresponding to the standard attacker capabilities. The first rule corresponds to the attacker's capability of hashing a message. The second and third rules represent symmetric key encryption and decryption. The remaining rules correspond to public-key operations and should be clear. The attacker can also perform basic operations which we do not mention explicitly, such as concatenation, generating new terms, etc. Moreover, we can give initial knowledge to the attacker. To add the message $m$ to the attacker's initial knowledge, we add the fact $\mathsf{att}(m)$.

In figure 2, we describe the additional capabilities, which we grant to the attacker when we assume that encryption is used with either CBC mode or ECB mode[5]. When using CBC, we allow the attacker, given the encryption of a message, to extract the encryption of a prefix, respectively a postfix of this message. We explained in section 2 how this is realized. In this abstract modelling, we suppose that all messages are either atomic or a concatenation of atomic messages. It is not possible to cut an atomic message, even if in real life this might be feasible and could be used to mount subtle attacks. As discussed in section 2, when using public key encryption it is also possible to extend an encrypted message, if the encryption key is known. This is reflected by the third rule. In the fourth and fifth rule we model the fact that a random-looking block is produced by introducing a new name $rnd$, which depends on the two blocks that are incorrectly merged when decrypting. When ECB is used as a chaining technique, we add similar capabilities.

[5] Encoding these attacker models is slightly more complicated as shown in the figure, as we have to use the pairing operator inside functions rather than operations on tuples.

**Cipher block chaining:**

$$\frac{\mathsf{att}(\mathsf{sE}_k(m,n))}{\mathsf{att}(\mathsf{sE}_k(m)) \land \mathsf{att}(\mathsf{sE}_k(n))} \qquad \frac{\mathsf{att}(\mathsf{pE}_{\mathsf{pk}(x)}(m,n))}{\mathsf{att}(\mathsf{pE}_{\mathsf{pk}(x)}(m)) \land \mathsf{att}(\mathsf{pE}_{\mathsf{pk}(x)}(n))}$$

$$\frac{\mathsf{att}(\mathsf{pE}_{\mathsf{pk}(x)}(m)) \land \mathsf{att}(n) \land \mathsf{att}(\mathsf{pk}(x))}{\mathsf{att}(\mathsf{pE}_{\mathsf{pk}(x)}(m,n))}$$

$$\frac{\mathsf{att}(\mathsf{sE}_k(m,n)) \land \mathsf{att}(\mathsf{sE}_k(m',n'))}{\mathsf{att}(\mathsf{sE}_k(m,n,rnd[n,m'],n'))} \qquad \frac{\mathsf{att}(\mathsf{pE}_{\mathsf{pk}(x)}(m,n)) \land \mathsf{att}(\mathsf{pE}_{\mathsf{pk}(x)}(m',n'))}{\mathsf{att}(\mathsf{pE}_{\mathsf{pk}(x)}(m,n,rnd[n,m'],n'))}$$

**Electronic code book:**

$$\frac{\mathsf{att}(\mathsf{sE}_k(m,n))}{\mathsf{att}(\mathsf{sE}_k(m)) \land \mathsf{att}(\mathsf{sE}_k(n))} \qquad \frac{\mathsf{att}(\mathsf{pE}_{\mathsf{pk}(x)}(m,n))}{\mathsf{att}(\mathsf{pE}_{\mathsf{pk}(x)}(m)) \land \mathsf{att}(\mathsf{pE}_{\mathsf{pk}(x)}(n))}$$

$$\frac{\mathsf{att}(\mathsf{sE}_k(m)) \land \mathsf{att}(\mathsf{sE}_k(n))}{\mathsf{att}(\mathsf{sE}_k(m,n))} \qquad \frac{\mathsf{att}(\mathsf{pE}_{\mathsf{pk}(x)}(m)) \land \mathsf{att}(\mathsf{pE}_{\mathsf{pk}(x)}(n))}{\mathsf{att}(\mathsf{pE}_{\mathsf{pk}(x)}(m,n))}$$

**Fig. 2:** Extended attacker capabilities due to block chaining

### *3.3 Property specification*

Blanchet's protocol verifier implements a Prolog-like solving algorithm, which can be used to check whether a fact is derivable from the available rules or not. The most direct properties one can verify are secrecy properties. If we want to know whether a protocol guarantees a secret $s$, we verify that $\mathsf{att}(s)$ cannot be deduced. Moreover, whenever $s$ is derivable from the rules, the tool shows the derivation trace.

We are interested in verifying whether a protocol offers possibility for known-pairs or chosen-text attacks, in order to determine what hypotheses are required on the primitives. We will show now that these different attacks can be formalised as secrecy properties.

### Known-pair attacks

Remember that in order to mount a known-pair attack, we need to know a ciphertext and the corresponding plaintext. Therefore, we say that a protocol allows a known-pair attack with respect to key $k$, if $\mathsf{att}(\mathsf{sE}_k(x), x)$ can be deduced for some $x$.

### Chosen-plaintext attacks

For an attacker to be able to mount a chosen-plaintext attack, the attacker should be able to use the protocol as an *oracle* and choose whatever plaintext to be encrypted. We model this by giving the attacker a *challenge*, i.e. we add a special name called challenge to his knowledge, which is a plaintext that is never used by any of the protocol participants. If the attacker succeeds in getting this challenge encrypted, we conclude that he can do so for any plaintext of his choice. We say that a protocol allows a chosen-plaintext attack with respect to key $k$, if given $\mathsf{att}(challenge[])$, it is possible to deduce $\mathsf{att}(\mathsf{sE}_k(challenge[]))$.

### Chosen-ciphertext attacks

A chosen-ciphertext attack is formalised in a similar way to chosen-plaintext attacks. The idea is that the challenge we give to the attacker is a ciphertext, for which he does not know the plaintext. As we are in a free term algebra, we cannot express that a challenge is a ciphertext; instead, we give the attacker the encryption of the challenge and require him to recover the challenge itself. We say that a protocol allows a chosen-ciphertext attack with respect to key $k$, if given $\mathsf{att}(\mathsf{sE}_k(challenge[]))$, it is possible to deduce $\mathsf{att}(challenge[])$. The query for a public-key encryption function is similar.

We can now briefly compare our definitions to Stubblebine and Meadows' definitions [21,22]. First, we use the idea of a challenge to model chosen-text attacks. Meadows and Stubblebine used a special function *not sent* which enables the attacker to *choose* a value. The definition of this function is however rather tricky. Second, we do not explicitly need decryption to define a chosen-ciphertext attack. Giving the attacker knowledge of the encryption of the challenge and requiring him to produce the challenge itself captures the notion of chosen-cipher attack.

## 4 Analysis

In this section, we illustrate our analysis on two well known protocols: the Needham-Schroeder-Lowe public-key protocol and the Needham-Schroeder symmetric-key protocol.

### 4.1 The Needham-Schroeder-Lowe public-key protocol

The protocol is described in protocol 1. We suppose that the reader is familiar with its main ideas. Our analysis reveals two things. On one hand, we show that giving the attacker additional capabilities related to block chaining allows the attacker to compromise the secrecy of the nonces used in the protocol. On the other hand, the protocol allows the attacker to mount a chosen-ciphertext attack. We then show a slight variation of the protocol, which only requires to change the order of fields in the first message to prevent both attacks.

### An attack compromising secrecy of nonces

We first show, how the attacker learns Alice's nonce. The attack is based on the attacker's capabilities to extract a prefix from a ciphertext and to add a postfix to an existing ciphertext. The attack is presented in attack 1, where $I$ denotes the attacker, or intruder. The attacker starts observing the first message in a protocol session between Alice and Bob. Then, the attacker extracts $\mathsf{pE}_{\mathsf{pk}(B)}(N_A)$ from this message. For this, he takes the prefix of the ciphertext. As the attacker knows Bob's public key, he can construct the message $\mathsf{pE}_{\mathsf{pk}(B)}(N_A, I)$. Remember that this is possible, by ciphering his identity, and considering the last block of $\mathsf{pE}_{\mathsf{pk}(B)}(N_A)$ as the $IV$. Now the attacker can himself start a session with Bob, who decrypts the unknown nonce $N_A$ and sends it back encrypted with the attacker's public key.

**Attack 1** An attack compromising the secrecy of Alice's nonce

1. A → B: $\mathsf{pE}_{\mathsf{pk}(B)}(N_A, B)$
2. extract $\mathsf{pE}_{\mathsf{pk}(B)}(N_A)$ from $\mathsf{pE}_{\mathsf{pk}(B)}(N_A, B)$ and construct $\mathsf{pE}_{\mathsf{pk}(B)}(N_A, I)$
3. I → B: $\mathsf{pE}_{\mathsf{pk}(B)}(N_A, I)$
4. B → I: $\mathsf{pE}_{\mathsf{pk}(I)}(N_A, N_B, \mathsf{pk}(B))$

An even simpler attack can be used to learn Bob's nonce. Here, the attacker observes the third message of the protocol, which is $\mathsf{pE}_{\mathsf{pk}(B)}(N_B)$. From this message, the attacker constructs $\mathsf{pE}_{\mathsf{pk}(B)}(N_B, I)$, starts a fresh session with Bob and proceeds exactly as in the previous attack. Note, that in this attack, we do not even need to extract a prefix from a message. We describe the scenario in attack 2.

### Chosen-ciphertext attacks

We now show how a chosen-ciphertext attack can be launched against Bob's secret key. The attack is again based on the idea of postfixing an existent ciphertext. Suppose that we give the intruder the ciphertext $\mathsf{pE}_{\mathsf{pk}(B)}(challenge)$. In order to succeed the attacker only needs to be able to recover the plaintext *challenge*. The attacker can achieve this as follows. First, he constructs the message $\mathsf{pE}_{\mathsf{pk}(B)}(challenge, I)$ and then starts a protocol session with Bob. Bob will accept the encrypted challenge as the encryption of the attacker's

**Attack 2** An attack compromising the secrecy of Bob's nonce

1. A → B: $\mathsf{pE}_{\mathsf{pk}(B)}(N_A, B)$
2. B → A: $\mathsf{pE}_{\mathsf{pk}(A)}(N_A, N_B, \mathsf{pk}(B))$
3. A → B: $\mathsf{pE}_{\mathsf{pk}(B)}(N_B)$
4. construct $\mathsf{pE}_{\mathsf{pk}(B)}(N_B, I)$ from $\mathsf{pE}_{\mathsf{pk}(B)}(N_B)$
5. I → B: $\mathsf{pE}_{\mathsf{pk}(B)}(N_B, I)$
6. B → I: $\mathsf{pE}_{\mathsf{pk}(I)}(N_B, N'_B, \mathsf{pk}(B))$

nonce and will send back its decryption, encrypted with the intruder's private key. This scenario is presented in attack 3. Note that there is no chosen-ciphertext attack against Alice's private key (i.e. against the private key of the person acting as the initiator of the protocol). The intruder could of course start a session with Alice, where Alice is taking Bob's *role*.

**Attack 3** A chosen-ciphertext attack on Bob's private key

1. construct $\mathsf{pE}_{\mathsf{pk}(B)}(challenge, I)$ from $\mathsf{pE}_{\mathsf{pk}(B)}(challenge)$
2. I → B: $\mathsf{pE}_{\mathsf{pk}(B)}(challenge, I)$
3. B → I: $\mathsf{pE}_{\mathsf{pk}(I)}(challenge, N_B, \mathsf{pk}(B))$

### A more robust version of the protocol

There exists a simple way to avoid the above described potential attacks in the case CBC is used. If we have a closer look at them, we realize that they are all based on the fact that the first message can be forged by appending the attacker's identity to the encryption of an unknown nonce. A simple change consists in inverting the nonce and the identity in the first field, which gives us the message $\mathsf{pE}_{\mathsf{pk}(B)}(B, N_A)$, instead of $\mathsf{pE}_{\mathsf{pk}(B)}(N_A, B)$. As the unknown part of the message, the nonce, is the last field, we cannot use the same "trick" on CBC as before. It is not possible to add a prefix to an encrypted text, as the $IV$ would not match anymore. Note however, that when ECB would be used as the block chaining mode, our fix does not help. Nor does it help if we use right-to-left instead of the usual left-to-right CBC. The fact that changing the order of the fields avoids the above attack is rather a curiosity than a real fix. A better protocol practice would be to ensure integrity of the message by the means of a cryptographic hash of the message, which could be included in the cipher. We changed the protocol by adding this additional integrity mechanism, i.e. we systematically replace $\mathsf{pE}_{\mathsf{pk}(x)}(m)$ with $\mathsf{pE}_{\mathsf{pk}(x)}(m, \mathsf{h}(m))$. This modified version resists against all of the above attacks.

### *4.2 The Needham-Schroeder symmetric key protocol*

Let us first briefly describe the classical Needham-Schroeder symmetric key protocol [15], which we use as a second case study, illustrating the analysis of known-pair and chosen-plaintext attacks.

**Protocol 2** The Needham-Schroeder symmetric key protocol

1. A → S: A,B,$N_A$
2. S → A: $\mathsf{sE}_{K_{AS}}(N_A, B, K_{AB}, \mathsf{sE}_{K_{BS}}(K_{AB}, A))$
3. A → B: $\mathsf{sE}_{K_{BS}}(K_{AB}, A)$
4. B → A: $\mathsf{sE}_{K_{AB}}(N_B)$
5. A → B: $\mathsf{sE}_{K_{AB}}(N_B - 1)$

The protocol is described in protocol 2. It uses a trusted key server $S$ to establish a shared key $K_{AB}$ between Alice and Bob. It is assumed that the server shares long term keys with each entity. In the first message, Alice requests the server to generate a key to be shared between Bob and herself. The server answers by sending back the key, as well as the key encrypted for Bob. Moreover a nonce $N_A$ is used to ensure freshness. Then Alice sends the encrypted key to Bob, in message 3. Messages 4 and 5 are a challenge-response handshake to ensure that Alice knows the key and that the key is fresh. Note that $N_B - 1$ is modelled by declaring a function $decrease()$.

**Known-pair attacks**

Known-pair attacks against the long-term keys $K_{AS}$ and $K_{BS}$ are simple. To find such pairs, the attacker does not even require any special capability. A known pair with respect to $K_{AS}$ is found when Alice starts a session with the attacker. When the three first messages have been exchanged, the attacker has learned all the elements, which are encrypted in the second message, as the third message is directly sent to the attacker and he can extract the shared key $K_{AI}$. To learn a known pair with respect to Bob's long term key $K_{BS}$ the attacker only needs to send $I, B, N_I$ to the server. The server's answer will contain $\mathsf{sE}_I(K_{IB})$, and hence the attacker has a known pair. When analysing the protocol for known-pair attacks against the session key $K_{AB}$, the tool came up with a *false attack*. This means that we can neither conclude that an attack exists nor that the protocol is secure. However, when we disable the attacker capability of concatenating two ciphertexts, generating a random block, no known-pair attack is possible against the established session key $K_{AB}$. It is rather surprising that the protocol offers more possibilities to attack the long term keys than the session keys, whose security level should be considered as lower.

### Chosen-plaintext attacks

When we do not add additional attacker capabilities, chosen-plaintext attacks are not feasible. However, when we allow the intruder to deduce $\mathsf{sE}_k(m)$ and $\mathsf{sE}_k(n)$ from $\mathsf{sE}_k(m, n)$, the two long term keys can again be attacked.

We first describe the attack against $K_{AS}$, where the attacker has to learn the encryption of the challenge under this key. The attack is presented in attack 4, where the notation A(I) means that the attacker is masquerading as Alice. Note that this attack only involves the presence of the server and not of Alice nor Bob. Therefore this same attack can be directly reused to mount an attack against Bob's long shared key.

**Attack 4** A chosen-plaintext attack on Alice's long term key

1. A(I) → S: A, B, *challenge*
2. S → A(I): $\mathsf{sE}_{K_{AS}}(challenge, B, K_{AB}, \mathsf{sE}_{K_{BS}}(K_{AB}, A))$
3. I extracts $\mathsf{sE}_{K_{AS}}(challenge)$

## 5 Conclusion

In this paper, we report on an automated analysis of known-pair and chosen-text attacks. For this purpose we give definitions in terms of reachability for these attacks. In particular, we use the notion of challenge when defining chosen-text attacks. As known-pair and chosen-text attacks are performed at the level of blocks, we model two block chaining techniques, ECB and CBC. These block chaining techniques are modelled in an abstract way, without the need of actually modelling the exclusive or operation for CBC, using deduction rules. Our method is automated using Blanchet's protocol verifier and illustrated on two famous examples, the Needham-Schroeder-Lowe public-key protocol and the Needham-Schroeder symmetric-key protocol. On the first protocol, we show how the special intruder capabilities related to chaining may compromise the secrecy of nonces and that chosen-ciphertext attacks are possible. We propose two modified versions of the protocol which strengthen its security. We then illustrate known-pair and chosen-plaintext attacks on the Needham-Schroeder symmetric key protocol.

As future work we foresee to apply these techniques to real-life protocols, which precise what chaining techniques are used. An interesting case study would be Kerberos, as it is based on the here studied Needham-Schroeder symmetric key protocol. Moreover, we believe that the additional attacker capabilities allow us to discover new guessing attacks. Consider a protocol that uses (deterministic) public-key encryption to encrypt weak data $w$, i.e. $\mathsf{pE}_{\mathsf{pk}(x)}(w)$. A trivial guessing attack is to guess $w'$ and verify the guess by re-encrypting $w'$ and comparing it to the original cipher. While adding a fresh nonce inside the encryption, i.e. $\mathsf{pE}_{\mathsf{pk}(x)}(N, w)$, would make this guessing attack impossible in a standard model, our additional attacker capabilities would keep the attack feasible.

### Acknowledgements

The authors wish to thank the anonymous reviewers for helpful comments, as well as Bruno Blanchet for his precious advice while using his tool. We gratefully acknowledge funding from EPSRC grant number GR/R02214/01, “The Feature Constuct in Programming and Specification Languages”.

## References

[1] Blanchet, B., *An efficient cryptographic protocol verifier based on prolog rules*, in: S. Schneider, editor, *14th IEEE Computer Security Foundations Workshop* (2001), pp. 82–96.

[2] Chevalier, Y., R. Kuester, M. Rusinowitch and M. Turuani, *An NP decision procedure for protocol insecurity with xor*, in: P. G. Kolaitis, editor, *Symposium on Logic in Computer Science (LICS'03)* (2003), pp. 261–270.

[3] Comon-Lundh, H. and V. Shmatikov, *Intruder deductions, constraint solving and insecurity decision in presence of exclusive or*, in: P. G. Kolaitis, editor, *Symposium on Logic in Computer Science (LICS'03)* (2003), pp. 271–280.

[4] Corin, R., S. Malladi, J. Alves-Foss and S. Etalle, *Guess what? Here is a new tool that finds some new guessing attacks (extended abstract)*, in: R. Gorrieri and R. Lucchi, editors, *In Proceedings of the Workshop on Issues in the Theory of Security (WITS '03)*, Warsaw, Poland, 2003, pp. 62–71.

[5] Delaune, S. and F. Jacquemard, *A theory of dictionary attacks and its complexity*, in: R. Focardi, editor, *17th IEEE Computer Security Foundations Workshop* (2004), pp. 2–15.

[6] Dolev, D. and A. C. Yao, *On the security of public key protocols*, IEEE Transactions on Information Theory **29** (1983), pp. 198–208.

[7] Durgin, N. A., P. D. Lincoln, J. C. Mitchell and A. Scedrov, *Multiset rewriting and the complexity of bounded security protocols*, Journal of Computer Security (2002).

[8] Guttman, J. D., “Foundations of Security Analysis and Design,” Lecture Notes in Computer Science **2171**, Springer-Verlag, 2001 pp. 197–261.

[9] Lowe, G., *An attack on the Needham-Schroeder public-key authentication protocol*, Information Processing Letters **56** (1995), pp. 131–133.

[10] Lowe, G., *Analysing protocols subject to guessing attacks.*, in: J. Guttman, editor, *In Proceedings of the Workshop on Issues in the Theory of Security (WITS '02)*, Portland, Oregon, USA, 2002.

[11] Mao, W. and C. Boyd, *On the use of encryption in cryptographic protocols*, in: *Proceedings of 4th IMA Conference on Cryptography and Coding*, Cirencester, England, 1995, pp. 251–262.

[12] Meadows, C., *The NRL protocol analyzer: An overview*, Journal of Logic Programming **26** (1996), pp. 113–131.

[13] Menezes, A. J., P. C. van Oorschot and S. A. Vanstone, "Handbook of Applied Cryptography," Series on Discrete Mathematics and its Applications, CRC Press, 1997.

[14] Millen, J. K. and V. Shmatikov, *Symbolic protocol analysis with products and diffie-hellman exponentiation*, in: R. Focardi, editor, *16th IEEE Computer Security Foundations Workshop* (2003), pp. 47–61.

[15] Needham, R. M. and M. D. Schroeder, *Using encryption for authentication in large networks of computers*, Communications of the ACM **21** (1978), pp. 993–999.

[16] Paulson, L. C., *Proving properties of security protocols by induction*, in: *10th IEEE Computer Security Foundations Workshop* (1997), pp. 70–83.

[17] Pereira, O., *On the perfect encryption assumption*, in: P. Degano, editor, *Workshop on Issues in the Theory of Security (WITS 2000)*, Geneva, Switzerland, 2000, pp. 42–45.

[18] Shoup, V., *OAEP reconsidered*, in: J. Kilian, editor, *Advances in Cryptology—Crypto 2001*, Lecture Notes in Computer Science **2139** (2001), pp. 239–259.

[19] Song, D. X., *Athena, an automatic checker for security protocol analysis*, in: *12th IEEE Computer Security Foundations Workshop* (1999), pp. 192–202.

[20] Stubblebine, S. G. and V. D. Gligor, *On message integrity in cryptographic protocols*, in: J. McLean and R. Kemmerer, editors, *Symposium on Research in Security and Privacy* (1992), pp. 85–104.

[21] Stubblebine, S. G. and C. A. Meadows, *On searching for known and chosen cipher pairs using the nrl protocol analyzer*, in: H. Orman and C. Meadows, editors, *DIMACS Workshop on Design and Formal Verification of Security Protocols*, Piscataway, NJ, USA, 1997.

[22] Stubblebine, S. G. and C. A. Meadows, *Formal characterization and automated analysis of known-pair and chosen-text attacks*, IEEE Journal on Selected Areas in Communications **18** (2000), pp. 571–581, special issue on Network Security.

[23] Turuani, M., "Security of Cryptographic Protocols: Decidability and Complexity," Ph.D. thesis, Université Henri Poincaré — Nancy 1, Nancy, France (2003).

[24] Turuani, M. and M. Rusinowitch, *Protocol insecurity with finite number of sessions is NP-complete*, in: S. Schneider, editor, *14th IEEE Computer Security Foundations Workshop* (2001), pp. 174–187.