Defective Sign & Encrypt in S/MIME, PKCS#7, MOSS, PEM, PGP, and XML

Don Davis*

Trust, but verify. – *Russian proverb*

Abstract

Simple Sign & Encrypt, by itself, is not very secure. Cryptographers know this well, but application programmers and standards authors still tend to put too much trust in simple Sign-and-Encrypt. In fact, every secure e-mail protocol, old and new, has codified naïve Sign & Encrypt as acceptable security practice. S/MIME, PKCS#7, PGP, OpenPGP, PEM, and MOSS all suffer from this flaw. Similarly, the secure document protocols PKCS#7, XML-Signature, and XML-Encryption suffer from the same flaw. Naïve Sign & Encrypt appears only in file-security and mail-security applications, but this narrow scope is becoming more important to the rapidly-growing class of commercial users. With file- and mail-encryption seeing widespread use, and with flawed encryption in play, we can expect widespread exposures.

In this paper, we analyze the naïve Sign & Encrypt flaw, we review the defective sign/encrypt standards, and we describe a comprehensive set of simple repairs. The various repairs all have a common feature: when signing and encryption are combined, the inner crypto layer must somehow depend on the outer layer, so as to reveal any tampering with the outer layer.

1 Introduction

Since the invention of public-key cryptography, cryptographers have known that naïve combinations of encryption and signature operations tend to yield insecure results [1, 2]. To guarantee good security properties, carefully designed security protocols are necessary. However, most security protocols of the past 25 years have focused on securing network connections, and relatively simple file-encryption problems have received surprisingly little attention from protocol designers.

Users and programmers prefer to think about security by analogy with familiar symmetric-key "secret codes." For mail-handling and file-handling, security designers have relied heavily on simple asymmetric encryption and signing, rather naïvely combined. Naïve sign & encrypt has surprisingly different security semantics from symmetric encryption, but the difference is subtle, perhaps too subtle for non-specialist users and programmers to grasp. Indeed, for senders, sign-and-encrypt guarantees the same security properties as symmetric-key cryptography gives. With both types of crypto, the sender is sure that:

- The recipient knows who wrote the message; and
- Only the recipient can decrypt the message.

The difference appears only in the *recipient's* security guarantees: the recipient of a symmetric-key ciphertext knows who sent it to him, but a "simple sign & encrypt" recipient knows only who *wrote* the message, and has no assurance about who *encrypted* it. This is because naïve sign & encrypt is vulnerable to "surreptitious forwarding," but symmetric-key encryption is not. Since users always will assume that sign & encrypt is similar to symmetric-key "secret codes," they will tend to trust naïve sign & encrypt too much.

The standards that exist for simple file-encryption, chiefly PKCS#7 [23] and S/MIME [20], tend to allow secure Sign & Encrypt implementations (i.e., such as would prevent surreptitious forwarding), but surprisingly, these file-security standards don't *require* fully-secure implementation and operation. Similarly, some important new security standards, such as the XML[1] security specifications [6, 26], offer only low-level "toolbox" APIs. Too often, both the established standards and the new ones allow insecure yet compliant implementations. Application programmers need more security guidance than these "toolbox" APIs offer, in order to build effective security into their applications. Without such guidance, programmers tend to suppose incorrectly that simply signing and

*Affiliations: Shym Technology, 75 Second Ave. Suite 610 Needham, MA 02494; Curl Corp., 400 Technology Sq., Cambridge, MA 02139; *ddavis@curl.com, don@mit.edu*

[1]Extended Markup Language

then encrypting a message or a file will give good security.

The limitations of naïve sign & encrypt probably were well-known to the designers of all of the standards we discuss here (see §4.6). The standards authors assumed, sometimes explicitly and sometimes implicitly, that applications programmers and end-users would understand that naïve sign & encrypt is not a complete security solution. Application programmers were expected to know how to bolster each standard's sign & encrypt feature with other protocol elements. At the same time, end-users were expected to make careful security judgments about any application they might use, so as to use the application's security features correctly, and so as not to over-rely on a product that offers only limited security. The standards authors' expectations may have been realistic ten years ago, before Everyman and the Acme Boot-Button Co. began using the Internet. It seems unfair to fault the standards designers for insufficient prescience, but now, these expectations are hopelessly outdated, and those standards cannot serve end-users well.

1.1 Surreptitious Forwarding

Why is naïve Sign & Encrypt insecure? Most simply, S&E is vulnerable to "surreptitious forwarding:" Alice signs & encrypts for Bob's eyes, but Bob re-encrypts Alice's signed message for Charlie to see. In the end, Charlie believes Alice wrote to him directly, and can't detect Bob's subterfuge. Bob might do this just to embarass Alice, or Charlie, or both: [2]

$$A \to B \quad : \quad \{\{\text{``}\mathit{I\ love\ you}\text{''}\}^a\}^B \tag{1}$$
$$B \to C \quad : \quad \{\{\text{``}\mathit{I\ love\ you}\text{''}\}^a\}^C \tag{2}$$

Here, Bob has misled Charlie to believe that "Alice loves Charlie." More serious is when Bob undetectably exposes his coworker Alice's confidential information to a competitor:

$$A \to B \quad : \quad \{\{\text{``}\mathit{sales\ plan}\text{''}\}^a\}^B \tag{3}$$
$$B \to C \quad : \quad \{\{\text{``}\mathit{sales\ plan}\text{''}\}^a\}^C \tag{4}$$

In this case, Alice will be blamed conclusively for Bob's exposure of their company's secrets.

Further, when Alice signs a message to Bob, Alice may be willing to let Charlie see that message, but not to *sign* the same message for Charlie:

$$A \to B \quad : \quad \{\{\text{``}\mathit{I.O.U.\ \$10K}\text{''}\}^a\}^B \tag{5}$$
$$B \to C \quad : \quad \{\{\text{``}\mathit{I.O.U.\ \$10K}\text{''}\}^a\}^C \tag{6}$$

If every user could be relied upon to understand that Sign & Encrypt is vulnerable to surreptitious forwards, then Alice wouldn't have to worry about Bob forwarding her message to Charlie. But in reality, when Charlie gets Alice's message via Bob, Charlie very likely will assume that Alice sent it to him directly. Thus, even if Alice doesn't care whether Bob divulges the message, she may be harmed if Bob is able to forward her signature surreptitiously.

1.2 Don't Sign Ciphertexts

Interestingly, naïve Encrypt-then-Sign isn't any better than Sign & Encrypt. In this case, it's easy for any eavesdropper to replace the sender's signature with his own, so as to claim authorship for the encrypted plaintext:

$$A \to\!| \, B \quad : \quad \{\{\text{``}\mathit{my\ idea}\text{''}\}^B\}^a \tag{7}$$
$$C \to B \quad : \quad \{\{\text{``}\mathit{my\ idea}\text{''}\}^B\}^c \tag{8}$$

Note that Charlie has to block Bob's receipt of Alice's original message, before sending the re-signed ciphertext.

Another problem with Encrypt-then-Sign arises, when Alice uses RSA or El Gamal encryption. In a sequel to Abadi's "Robustness Principles" paper [1], Anderson showed that Encrypt&Sign is dramatically weaker than had been thought [2]. Suppose Alice uses RSA keys to send Bob an E&S message:

$$A \to B \quad : \quad \{\{msg\}^B\}^a \tag{9}$$

Then Bob can pretend that Alice encrypted and signed an *arbitrary* message msg', of his choice. To alter Alice's plaintext, Bob uses the factors of his own RSA modulus n_B to calculate the discrete logarithm x of Alice's message msg, using as base Bob's arbitrary message msg':

$$\{msg'\}^x \quad = \quad msg \pmod{n_B} \tag{10}$$

Now, Bob needs only to certify (xB, n_B) as his public key, in order to make Alice's original ciphertext signature valid for Bob's new encryption $\{msg'\}^{xB}$:

$$B \to B \quad : \quad \{\{msg'\}^{xB}\}^a \tag{11}$$

[2]Notation: "A" is Alice's public key, and "a" is her private key. Thus, $\{msg\}^A$ is an encrypted ciphertext, and $\{msg\}^a$ is a signed message. We assume that the asymmetric-key cryptosystem behaves similarly to RSA [21], so that a signature is a private-key encryption.

Anderson's attack has two minor limitations:

- Each modulus factor must be short enough (~120 digits, or ~400 bits) to allow a discrete-log calculation [13];
- Bob's new public exponent xB will be obviously unusual, in that it will be a full-length bitstring, instead of the usual small integer value.

So, it might seem that Alice should be safe from this attack, as long as Bob's public key B is substantially longer than 240 digits (800 bits). Unfortunately, Alice cannot tell, without factoring Bob's RSA key-modulus, whether Bob used three or more prime factors to prepare his RSA key-pair [22]. If Bob has a large-modulus key-pair made up from several small factors, then Alice's naïve use of Encrypt & Sign would still leave her vulnerable to Bob's substituted-ciphertext attack.

Thus, whenever we want to sign a ciphertext, Anderson's attack forces Alice to sign, along with her ciphertext, either the plaintext itself or Bob's public key B:

$$A \rightarrow B \quad : \quad \{\{msg\}^B, \#msg\}^a \qquad (12)$$
$$A \rightarrow B \quad : \quad \{\{msg\}^B, \#B\}^a \qquad (13)$$

The two formats offer different advantages: signing the plaintext alongside the ciphertext gives non-repudiation, while signing the encryption key is more easily understood as a defense against Anderson's attack. In either format, Bob can still alter B and msg simultaneously, so that $\{msg'\}^{B'}$ is the same as Alice's ciphertext $\{msg\}^B$. But, in order to preserve Alice's signature, Bob now also has to choose msg' to have the same hash value as the one Alice signed, and this is too difficult.

Of course, Encrypt-then-Sign isn't very useful anyway, because only the illegible ciphertext, not the plaintext, would be non-repudiable. In what follows, for simplicity, we'll mostly ignore Encrypt & Sign, and we'll concentrate on analyzing and fixing Sign & Encrypt's defects.

1.3 Purpose of the Paper

This paper intends to fill the gap between the "do-it-yourself" toolbox APIs and the "out-of-the-box" secure-networking standards:

- Section 2 describes the problem's technical and social scope,
- Section 3 analyzes the problem cryptographically,
- Section 4 reviews several standards that accept naïve Sign / Encrypt as secure, and
- Section 5 presents a comprehensive variety of simple solutions.

Our goal is to help security standards offer a variety of *secure* ways to sign and encrypt messages. Application programmers should not be constrained by "one size fits all" protocols, but they also shouldn't have to understand the nuances of cryptographic design.

2 Problem Scope

Why is this old and easy problem worth discussing at this late date? Though designing a secure Sign & Encrypt protocol is easy for cryptographers, it's a different class of engineer who faces this problem nowadays. Application programmers have to rely on crypto vendors and crypto standards, in order to learn how to write crypto applications. Unfortunately, the vendors and standards have left untended a big gap in their support for application programmers. Current security standards don't give application programmers a simple recipe for file-encryption problems.

2.1 Technical Scope

Secure session protocols have attracted a lot of research attention, and several effective session-security protocols have been standardized, so naïve Sign & Encrypt is not a problem in session security. Session-security standards, like Kerberos [19], TLS [5], and SET [28], give straightforward, out-of-the-box solutions. For files and one-way messaging, though, current security standards give developers only a kind of "toolbox" support, with a variety of security options, but with no clear or firm guidance about how to combine the options to make Sign & Encrypt an effective security solution. Providing only toolbox-style cryptographic protocols is appropriate for a low-level mechanism like IPSEC [12], but for user-visible applications like secure e-mail, programmers need "turnkey" cryptography, not only cryptographic toolkits.

Thus, naïve Sign & Encrypt has come to characterize file-handling and e-mail security applications. PKCS#7 [23], CMS[3] [9], S/MIME [20], and PGP[4] [29], all suffer from this defect. Further, the W3C's[5] XML-Signature & XML-Encryption Working Groups have explicitly set themselves the task

[3] Cryptographic Message Syntax.
[4] Pretty Good Privacy
[5] The World Wide Web Consortium, see http://w3.org .

of supplying XML with S/MIME-style security. The demand for simple file-security and message-security is big and growing, so widespread use of these naïve Sign & Encrypt security models will lead to widespread exposures.

2.2 Social Scope

Increasingly, secure applications are being designed and built by application programmers, not by cryptographers. Several factors have obliged mainstream application programmers to undertake public-key protocol design:

- Commercial PKI is in widespread deployment;
- Secure networking standards don't address file-encryption;
- Demand for cryptographers greatly exceeds the supply.

So, when application programmers need file-encryption help, they can seek help from crypto vendors and from crypto standards. Unfortunately, the vendors and the standards both offer either high-level secure connections, or low-level "toolkit" mechanisms. Neither offering makes file-encryption easy. The available standards specifications for file-encryption intend to support security applications, but the specifications tend to standardize only low-level APIs for cryptographic primitives, so as to leave designers as much flexibility as possible.

3 Defective Standards

The delicacy of naïve Sign & Encrypt is a well-known issue in S/MIME. Similar flaws appeared in 1986 in the first version of the PGP message-format [30], and in 1988 in X.509v1 [14]. X.509's flaw was discovered in 1989 by Burrows et al. [4], and a correct repair was proposed in 1990 by I'Anson and Mitchell [11]. Unfortunately, more recent workers have failed to apply I'Anson's simple repair correctly; PEM and PKCS#7 suffer from a defective version of I'Anson's repaired Sign & Encrypt, and the same defect is now codified by S/MIME. In parallel with these developments, PGP independently retained the same naïve Sign & Encrypt defect. The current protocols' flaw is substantially similar to the original flaws in X.509 and PGP. So, the historical flow of inheritance is:

- Zimmermann described a naïve RSA-based Sign & Encrypt protocol, which later became PGP;
- X.509v1 codified a flawed, naïve Encrypt & Sign, independently of PGP;
- Burrows et al. and I'Anson described a workable Sign & Encrypt protocol for X.509;
- PEM applied X.509's cryptography to e-mail transport, using naïve S&E instead of I'Anson's repaired S&E;
- Three standards extended and generalized PEM:
  1. MOSS extended PEM to support MIME-encoded e-mail, by adding naïve Sign & Encrypt for e-mail attachments;
  2. PKCS#7 generalized PEM to non-mail file-handling applications, but preserved the S&E flaw intact;
  3. CMS and S/MIME carried PKCS#7's generality and the flawed S&E back to the e-mail community.
- Today, the nascent XML security standards expressly intend to support naïve Sign & Encrypt.

These relationships aren't as complicated as they look, because MOSS, PKCS#7, and S/MIME are all descended from PEM, and through PEM from X.509, while PGP and XML are completely independent efforts.

In the rest of this section, we discuss the defective standards in the chronological order listed above.

3.1 PGP and OpenPGP

PGP is similar to PEM and simpler than S/MIME, in that PGP provides only three security options: Sign, Encrypt, and Sign & Encrypt. Of these security options, we are only interested in PGP's Sign & Encrypt (we will discuss only Sign & Encrypt in the other standards' subsections, too).

PGP's message-format had several similarities with later features of PEM and S/MIME:

- symmetric-key encryption for message bodies;
- unformatted message-bodies;
- independent crypto layers.

In our discussion, we'll omit PGP's use of symmetric-key ciphers for bulk encryption, because it is irrelevant to our surreptitious forwarding attack.

PGP's strongest security option is naïve Sign & Encrypt, so PGP is vulnerable to surreptitious forwarding:

$$A \rightarrow B \quad : \quad \{\{\text{“}\textit{The deal is off.}\text{”}\}^a\}^B \qquad (14)$$
$$B \rightarrow C \quad : \quad \{\{\text{“}\textit{The deal is off.}\text{”}\}^a\}^C \qquad (15)$$

Here, Alice has cancelled a deal with Bob, so Bob gets even with her later, by re-encrypting and redirecting Alice's signed message to her next business partner, Charlie.

Note that PGP's plaintext message-bodies are unformatted, containing no names for the sender or recipient. Because PGP doesn't allow formatted message bodies, an extra signature layer, or signed attributes, PGP doesn't admit any of the protocol repairs we describe below for S/MIME and PKCS#7 (see §§ 3.5, 3.6, & 5.1).

3.2 X.509, Version 1

The first version of X.509 included a simple protocol for secure message-exchange, employing secure message "tokens" with the following structure:

$$A \rightarrow B \quad : \quad \{Bob,\ \#msg,\ \{msg\}^B\}^a \qquad (16)$$

Burrows et al. [4] pointed out that C could readily replace A's signature with his own, leading B to attribute A's message to C:

$$C \rightarrow B \quad : \quad \{Bob,\ \#msg,\ \{msg\}^B\}^c \qquad (17)$$

(See also Eqn.7). So, I'Anson and Mitchell [11] offered a repaired token-structure for X.509:

$$A \rightarrow B \quad : \quad \{\{\#(Bob,\ msg)\}^a,\ msg\}^B \qquad (18)$$

Unfortunately, I'Anson's cryptographic notation was hard to understand,[6] and his text didn't emphasize exactly what made his corrected token secure:

> This modification involves no additional effort as far as token construction is concerned, and it is simply to require that the encryption of enc-Data is done *after* the signature operation instead of before.

I'Anson's text incorrectly implied that he had only replaced E&S with S&E. In fact, his repair worked only because he made Alice sign her recipient's name, *Bob*, along with her message. This signed name proved Alice's intent to write for Bob. If Alice's signature hadn't included Bob's name, then I'Anson's new token would have been just a naïve Sign & Encrypt, fully vulnerable to surreptitious forwarding.

Clearly, I'Anson's paper influenced the early PKI standards community, because PKCS#1 and various later RFCs cited the paper. Though PEM and later mail standards didn't cite I'Anson, they followed his paper's advice: PEM, PKCS#7, and CMS provided Sign & Encrypt as a basic operation, and S/MIME explicitly deprecated Encrypt & Sign. We suggest that had I'Anson explained the necessity of signing the recipient's name, the later standards would have used Sign & Encrypt correctly.

Note that X.509's original Encrypt & Sign token (cf. Eqn. 16, above) could have been fixed without signing first, by the simple addition of the sender's name, similar to I'Anson's signed recipient-name:

$$A \rightarrow B \quad : \quad \{\#msg,\ \{Alice,\ msg\}^B\}^a \qquad (19)$$

This repair, like I'Anson's, blocks Burrow's signature-replacement attack (cf. Eqn. 17), because Bob can now detect Charlie's replacement: if the signer's certificate doesn't match Alice's name inside the plaintext, then Bob can conclude that the message was tampered with. This repair also repairs Encrypt & Sign's non-repudiation problem, since Alice signs her plaintext explicitly. Finally, this repair also blocks Anderson's plaintext-replacing attack (see § 1.2).

3.3 PEM

Privacy-Enhanced Mail was the first notable secure-email standard for the Internet. PEM was designed and specified in the late 1980's and early 1990's [15]. The first version of PEM relied exclusively on symmetric-key cryptography, but as X.509's PKI specification settled, later versions of PEM increasingly emphasized public-key cryptography. It seems likely that PEM's over-reliance on naïve Sign & Encrypt led PEM's descendants MOSS, PKCS#7, and S/MIME to follow suit. Indeed, the later specifications tried hard to support backward-compatibile interoperation with PEM.

For our purposes, PEM provides essentially only two variants of mail security; a message can be signed only, or it can be signed and then encrypted. Like PGP, and like PEM's descendants PKCS#7, CMS, and S/MIME, PEM applies its signature and encryption steps to the message-body, i.e., *not* to the SMTP header, the "From: / To:" header, or to the "encapsulated header," which carries a PEM message's keys and names. PEM has no notion of signing or authenticating ancillary attributes, and also doesn't support extra crypto layers, so the repairs we discuss below for S/MIME and PKCS#7 (see §§ 3.5 & 3.6) won't work for PEM. To prevent surreptitious forwarding, a PEM message's author would have to include the recipient's name directly in the message-body. Of course, it could be very difficult for the receiving PEM mail-client to find the recipient's name in the body, so as to check automatically for surreptitious forwarding.

Today, PEM is not widely used, and PEM's vulnerability to surreptitious forwarding is mostly just

[6] In Eqn.16, we've simplified the X.509 token's structure, by leaving out various nonces and other parameters.

a matter of historical interest. But PEM's accomplishment and influence were great, because PEM successfully achieved platform-independent cryptographic interoperation, at a time when the still-new Internet was a much more heterogenous affair than it is today.

3.4 MOSS

MOSS extended PEM's cryptography in three principal ways:

1. By adding cryptographic support for MIME-formatted multipart messages (popularly known as attachments);
2. By allowing encryptions and signatures to be applied in any order, like S/MIME;
3. By decoupling secure mail from the monolithic X.500 public-key infrastructure, which had failed by the mid-1990's.

Like PEM, MOSS was eclipsed by S/MIME and by PGP, and is little heard-of today.

MOSS had another feature, one very valuable for our purposes: unlike the other secure e-mail protocols, MOSS explicitly provided by default for a sender Alice to be able to sign her message-header, along with her message-body. MOSS is the only e-mail standard that gives users such an out-of-the-box mechanism for signing the recipient-list. (S/MIME's ESS feature did allow header-signing, but this was explicitly intended as a link-oriented security feature for military mail servers. See the discussion of ESS, in the last half of §3.6.)

Header-signing was easy for MOSS to provide, because MOSS treated the header as just another "part" in the message. If Alice's MOSS message carried her signature and encryption on *both* the message-body and the message-header, Alice's MOSS message and her recipients would be fairly well-protected against surreptitious forwarding. Unfortunately, MOSS made header-signing an optional feature, and the MOSS RFCs don't discuss why header-signing is valuable. As specified, MOSS is as vulnerable to our attack as the other e-mail protocols are.

It's worth noting that even when Alice does choose to sign MOSS's header, MOSS's cryptography still relies too much on Bob's sophistication about e-mail security:

- When Bob receives Alice's MOSS message, he does have to read Alice's signed header, so as to make sure that Alice intended to send the message to him.
- Further, when Alice's cc-list is long, Bob still has to read the signed header, but this step is neither as automatic nor as reliable as one would like.
- Finally, if Alice's mail-client doesn't bother to sign her mail-headers, Bob probably won't notice, so he'll still be vulnerable to surreptitiously-forwarded messages.

All of these issues would vanish, if MOSS had made header-signing mandatory. Bob's e-mail reader presumably would automatically scan the header, looking for Bob's decryption-key's "name form," and if this search were to fail, the MOSS mail-reader would raise an error-message warning Bob.

3.5 PKCS#7

PKCS#7 was created as a file-oriented adaptation and extension of PEM's platform-independent cryptographic features. Accordingly, PKCS#7 inherited naïve Sign & Encrypt from PEM.

In order to bolster PKCS#7's Sign & Encrypt security, how might a PKCS#7 author securely attach names to a file or message? Each PKCS#7 message has SignerInfo and RecipientInfo fields, but the specification does not allow these fields to be signed or encrypted. PKCS#7 does provide for application-defined "authenticated attributes," though, so a PKCS#7 application could create a signed "To-List" attribute, so as to prove to recipients that they are the author's *intended* recipients. But crucially, PKCS#7 does not require or even suggest that for effective security, such a signed "To-list" should accompany the message. Further, PKCS#9 [24], which defines various attributes for PKCS#7 messages, similarly fails to provide any attributes for holding senders' or recipients' names.

Note also that in order to use authenticated attributes for repairing PKCS#7 Sign and Envelope, one must separately apply the signature and encryption steps, instead of using the Signed-and-Enveloped construct. This is because the combined construct doesn't support attributes at all [23]:

> Note. The signed-and-enveloped-data content type provides cryptographic enhancements similar to those resulting from the sequential combination of signed-data and enveloped-data content types. However, since the signed-and-enveloped-data content type does not have authenticated or unauthenticated attributes, nor does it provide enveloping of signer information other than the signature, the sequential combination of signed-data and enveloped-data content

types is gnerally preferable to the Signed-AndEnvelopedData content type, except when compatibility with the ENCRYPTED process type in Privacy-Enhanced Mail is intended.

Thus, for PKCS#7's simple Signed-and-Enveloped message, the protocol affords no cryptographically secure naming. The only way a Signed-and-Enveloped recipient can know that he is intended to see the message, and that no surreptitious forwarding has occurred, is for the sender to include the recipient's name within the message-body.

3.6 S/MIME and CMS

S/MIME is a set of secure email standards, which specify not only how to encrypt and sign messages, but also how to handle keys, certificates, and crypto algorithms. CMS is the specification that describes the data-formats and procedures needed for encryption and signatures. CMS is mostly identical to PKCS#7, from which it descends.

The S/MIME specification itself acknowledges that CMS' Sign & Encrypt isn't very secure, but the S/MIME specification fails to discuss the main defect. Further, the document tells implementors nothing about how to shore up Sign & Encrypt. Instead, the S/MIME specification merely cautions users and implementors not to over-rely on a message's security:

> 1. "An S/MIME implementation MUST be able to receive and process arbitrarily nested S/MIME within reasonable resource limits of the recipient computer.
>
> 2. "It is possible to either sign a message first, or to envelope[7] the message first. It is up to the implementor and the user to choose. When signing first, the signatories are then securely obscured by the enveloping. When enveloping first, the signatories are exposed, but it is possible to verify signatures without removing the enveloping. This may be useful in an environment where automatic signature verification is desired, as no private key material is required to verify a signature.
>
> 3. "There are security ramifications to choosing whether to sign first or to encrypt first. A recipient of a message that is encrypted and then signed can validate that the encrypted block was unaltered, but cannot determine any relationship between the signer and the unencrypted contents of the message. A recipient of a message that is signed-then-encrypted can assume that the signed message itself has not been altered, but that a careful attacker may have changed the unauthenticated portion of the encrypted message" [sic].
>
> – [20] *Sec. 3.5, "Signing and Encrypting."*

This excerpt is the S/MIME specification's only discussion of Sign & Encrypt's limitations. Several features in the excerpt deserve comment:

- Paragraph 2 presents the security issues as a tradeoff between confidentiality and ease of verification;
- Paragraph 3 hints that an attacker can replace the external signature in an encrypted-then-signed message,
- But there's no mention that sign-then-encrypt is vulnerable to surreptitious forwarding, by replacement of the outermost encryption layer. (In paragraph 3, "unauthenticated portion" seems to refer not to the unauthenticated ciphertext, but to unauthenticated plaintext.)
- The excerpt presents only the choice between signing first and encrypting first. There's no mention of repairing either option's defects.

S/MIME *is* flexible enough to allow the Sign & Encrypt defect to be repaired. In the specification excerpt above, the first paragraph provides that every S/MIME application must be able to process Sign/Encrypt/Signed messages and Encrypt/Sign/Encrypted messages. Either S/E/S or E/S/E suffices to reveal any alteration of the sender's crypto layers, as long as the receiving client knows how to detect the alterations (See §§ 5.2 & 5.3, below).

Note that our S/E/S double-signing only superficially resembles S/MIME's optional "triple-wrapping" feature; the two are different in mechanism and in purpose. S/MIME's Enhanced Security Services specification [7] provides specialized security-related message-attributes, in support of certain features such as signed receipts and secure mailing-lists. In order to support the ESS features, some mail servers will apply an extra signature to the ciphertext of an end-user's Signed-and-Encrypted message:

[7]The S/MIME, CMS, and PKCS#7 specification documents use the verbs "encrypt" and "envelope" interchangeably.

> *1.1* Triple Wrapping Some of the features of each service use the concept of a "triple wrapped" message. A triple wrapped message is one that has been signed, then encrypted, then signed again. The signers of the inner and outer signatures may be different entities or the same entity. Note that the S/MIME specification does not limit the number of nested encapsulations, so there may be more than three wrappings.
>
> *1.1.1* Purpose of Triple Wrapping Not all messages need to be triple wrapped. Triple wrapping is used when a message must be signed, then encrypted, and then have signed attributes bound to the encrypted body. Outer attributes may be added or removed by the message originator or intermediate agents, and may be signed by intermediate agents or the final recipient. [...]
>
> The outside signature provides authentication and integrity for information that is processed hop-by-hop, where each hop is an intermediate entity such as a mail list agent. The outer signature binds attributes (such as a security label) to the encrypted body. These attributes can be used for access control and routing decisions.

Triple-wrapping allows mail servers to securely annotate messages on-the-fly ("hop-by-hop"), primarily for the benefit of other mail-servers. In contrast, in our S/E/S repair, Alice applies her outer signature, without any extra attributes, to her own Signed & Encrypted message, as the basic CMS specification allows. Similarly, only Alice's intended S/E/S recipient Bob would validate her inner and outer signatures. In sum, our S/E/S is an end-to-end security feature, while ESS uses triple-wrapping to support link-oriented security features.

Further, ESS triple-wrapping and S/E/S serve different purposes. Though the first two ESS paragraphs do mention that an end-user like our Alice might apply an outer signature herself, the ESS document gives no reason that she might do so, except to attach signed attributes to the ciphertext. The ESS document nowhere suggests that triple-wrapping might be necessary to repair a security defect in Sign & Encrypt. In fact, the ESS specification committee did *not* intend triple-wrapping to be a repair for the surreptitious-forwarding defect. Instead, the ESS specification was written to fulfill the U.S. Dept. of Defense's purchasing criteria for secure e-mail, which demanded server-oriented security features [8].

Besides S/E/S, another S/MIME repair option comes from the CMS specification, which is a core piece of the S/MIME standards suite. Like PKCS#7, CMS provides for "signed attributes," which offer a different way to prevent crypto alterations. Suppose the sender includes a signed "To-List" attribute, and suppose the recipient knows how to process and interpret such an attribute. Then the recipient can identify who intended him to receive the message, and no attacker can profit by replacing the outer crypto layers. Unfortunately, like the PKCS#7 specification, the CMS specification does not stipulate or even suggest such naming attributes, though the specification does suggest other signed attributes.

These S/MIME repairs are cumbersome, and they only barely meet the e-mail industry's needs. Crucially, because the specification neither requires any repair, nor even mentions that some features can serve as repairs, the repairs' interpretations aren't standardized, and different vendors' S/MIME applications can't readily interoprate with full Sign & Encrypt security.

3.7 XML Security

At this writing (Spring 2001), the XML-Signatures draft specification [6] is nearing completion, and the allied XML-Encryption Working Group [26] is just starting its work. Both groups have explicitly committed to producing low-level "toolkit" specifications, which will describe how to combine basic public-key operations with a rich array of XML document-structuring features. In particular, both groups are very unwilling to stipulate any high-level security behavior, such as how to sign and encrypt with full security.

To some extent, this is proper: these standards are intended to support as broad a class of applications as possible, including document preparation and handling, financial applications, wire protocols, and potentially even intricate cryptographic security protocols. The Secure XML Working Groups say that they don't want to require secure high-level behavior in their specifications, because they don't want to constrain how low-level applications will use XML's security features. The WGs explicitly hope that a higher-level XML security specification, with out-of-the-box "idiot-proof" security, will be built someday to follow on the current WGs' specifications. But for now, certainly, the XML-Signatures draft specification is most suitable for use only by experienced security engineers and cryptographers, and not for application programmers who don't want to specialize in security.

4 Analysis

We propose that users of file-security and mail-security need simple security semantics, and that symmetric-key semantics are sufficient for most users and most applications' needs. Further, symmetric-key semantics are natural and easy for unsophisticated users to understand.

In this section, we present three overlapping views of what's wrong with naïve Sign & Encrypt. Then, we summarize and discuss several arguments in defense of the naïve Sign & Encrypt standards. Finally, we discuss how this flaw survived several standards-review committees' deliberations.

4.1 Asymmetric Security Guarantees

At first glance, naïve Sign & Encrypt seems quite secure, because message-author Alice gets the security guarantees she needs: her signature proves her authorship, and she knows who can read the message. The reader, Bob, doesn't get the same guarantees, though. He knows who wrote the message, but he doesn't know who encrypted it, and therefore doesn't know who else besides Alice has read the message. Note the asymmetry:

- When A sends B a signed & encrypted message, A knows that only B can read it, because A trusts B not to divulge the message, but –
- When B receives A's signed & encrypted message, B can't know how many hands it has passed through, even if B trusts A to be careful.

Seen this way, the flaw in naïve Sign & Encrypt is that B gets no proof that it was A who encrypted the message. In hindsight, this is obvious: public key algorithms usually don't automatically authenticate the encryptor of a message.

Certainly, in some applications, it's neither necessary nor feasible to give a recipient any assurance that only the sender has seen the message-plaintext. Thus, for example, mail-security applications do need the flexibility to waive full end-to-end symmetric-key semantics. But, whenever possible, and by default, mail- and file-security applications should give end-users easy-to-understand security guarantees.

4.2 Symmetric-Key Semantics

Users tacitly expect public-key file-encryption to offer the same security semantics that a symmetric key offers. Thus, another way to describe the Sign & Encrypt problem is that whether signing or encryption is applied first, naïve Sign & Encrypt fails to duplicate the security meaning of a symmetric-key ciphertext. When B receives a symmetric-key ciphertext from A, B can safely assume that:

- A sent the message,
- No-one else has seen the plaintext,
- A intended B to receive the plaintext.

With naïve Sign & Encrypt, these assumptions can break down, because the recipient may have to rely on the crypto layer to supply the intended recipient's names. That is, the problem arises when:

- The message plaintexts don't mention the sender's and target's names;
- The sender's and recipient's names are important for understanding the message or its security import;
- The recipient *assumes* that the signer encrypted the message.

Under these conditions, an attacker can successfully and surreptitiously forward a naïvely signed and encrypted message.

4.3 Sign & Encrypt Must Cross-Refer

We suggest that the messaging standards all erred by treating public-key encryption and digital signatures as if they were fully independent operations. This independence assumption is convienient for writing standards and for writing software, but it is cryptographically incorrect. When independent operations are applied one on top of another, then the outermost crypto layer can undetectably be replaced, and security is weakened.

In [1], Abadi and Needham presented a simple best-practice rule for protocol design:

> When a principal signs material that has already been encrypted, it should not be inferred that the principal knows the content of the message. On the other hand, it is proper to infer that the principal that signs a message and then encrypts it for privacy knows the content of the message.

In [2], Anderson and Needham presented their plaintext-substitution attack against Encrypt-then-Sign (see § 1.2), and they strengthened Abadi's prescription:

> Sign before encrypting. If a signature is affixed to encrypted data, then . . . a third party certainly cannot assume that the signature is authentic, so nonrepudiation is lost.

These principles were well-understood soon after X.509's defect was discovered (if not before), and to be fair, they were published after the early versions of PEM, PKCS#7 and S/MIME were published. But PKCS#7 and S/MIME have been revised since Abadi's and Anderson's papers became well-known, so the updated standards could have been repaired. Nevertheless, the e-mail standards still treat the Sign & Encrypt problem as a user-interface issue: "There are security ramifications to choosing whether to sign first or encrypt first..." [20].

Though signing and encryption are not independent of one another, the defective standards treated crypto operations as independent content-transformations, converting "content" to "content." Conceptually, this makes it easy for users and programmers to layer crypto operations in arbitrary depth and in arbitrary order. By this device, the standards authors sought to avoid constraining application developers' designs.

With such independent operations, though, it's hard to fulfill the recipient's security expectations. In order to work properly together, the signature layer and the encryption layer actually must refer to one another, so as to achieve basic symmetric-key security guarantees that users expect. The recipient needs proof that the signer and the encryptor were the same person, which necessarily entails either signing the recipient's identifier (in Sign & Encrypt), or encrypting the signer's identifier (in Encrypt & Sign). Once such cross-references are in place, an attacker can't remove and replace the outermost layer, because the inner layer's reference will reveal the alteration.

In Section 5, "Repair Options," we present five ways to give the recipient this cross-referenced proof of the encryptor's identity. In each of these five repairs, the sender identifies the outermost operation's key-holder, inside the innermost content, so as to bind the sender's and recipients' names together. For example, one repair for Sign & Encrypt puts the decrypting recipient's name inside the signed plaintext message:

$$A \rightarrow B \quad : \quad \{\{\text{``}\mathit{To:\ Bob}\text{''},\ \mathit{msg}\}^{a}\}^{B} \qquad (20)$$

This repair is straightforward for a user or an implementor to do, but it's hard for a standards specification to stipulate that different crypto operations must be tied together like this, without breaking the full generality of the content-transformation model.

4.4 Trust and Risk

A common defense of naïve Sign & Encrypt is that users have to be careful about whom they trust, or equivalently, that users should carefully assess risk when putting sensitive material under cryptographic protection. In this view, the recipient of a signed and encrypted message should not invest more trust in the message than the technology and the sender's reputation can support. This argument seems very plausible, but it turns out not to address the problems with naïve Sign & Encrypt.

B has no way to gauge the risk that the message has been divulged to people unknown to A and B. To gauge the risk, B would have to know how trustworthy are the people who have surreptitiously forwarded the message along from A towards B. Thus, in general, one can't assess the privacy of a decrypted plaintext, and shouldn't trust its privacy, unless one knows who encrypted it. In sum: if we accept the Trust and Risk argument, then the encryption step of Sign & Encrypt is quite pointless from the receiver's point-of-view.

4.5 Security and Ease-of-Use

Another common defense of S/MIME's naïve Sign & Encrypt is that "Users shouldn't trust unsigned information" about the signer's intended recipients. This argument misses the point of S/MIME's weakness, by supposing that users are over-relying on the unsigned SMTP header to identify the sender's intended recipients. The users' mistake is more subtle, though; they're over-relying on the encrypting-key's certificate, as a secure record of the sender's intended recipient.

It's unrealistic to expect today's users to catch such a subtle point. When X.509, PEM, and S/MIME were designed, PKI users were expected to be system administrators and other fairly sophisticated users; now, though, with the modern Internet and with electronic commerce in play, we can't expect most users to understand any cryptographic nuances at all.

A similar defense of the defective secure mail standards is that the specifications aren't actually broken, because "Applications can and should put names into the content, if that's what they want." This argument assumes that application programmers shouldn't try to incorporate cryptographic security into programs in the first place, unless they understand security and cryptography well enough to design security protocols. Further, the argument insists that no security standard can be so complete as to prevent ignorant programmers from "shooting themselvers in the foot."

A ready answer to this argument is "SSL." The SSL specification gives fairly complete security, out-of-the-box. Further, non-specialist programmers are able to set up secure SSL connections for their applications, without having to patch the SSL protocol on their own.

4.6 How Did This Happen?

According to the authors of the PEM [16, 17], S/MIME [8, 10], and XML-Security [25] standards, those working groups explicitly discussed surreptitious forwarding, and yet deliberately left the flaw unrepaired. The committees accepted this cryptographic neglect for several reasons:

- *Optional Coverage*: All of the specifications *allow* senders to put the recipient's name, or the whole mail header, into the message-body before signing. In addition, some protocols explicitly provide an optional mechanism for signing the mail header or the recipient-list.
- *Contextual Repair*: In the same way, the PEM committee's discussion explicitly decided that the message's context would *usually* solve the problem. For example, Alice's signed "Dear Bob" salutation would reveal any re-encryption.
- *Out of Scope*: The PEM committee noted that surreptitious forwarding is a type of replay, and that no e-mail mechanism can prevent e-mail replay. Thus, to the PEM committee, it seemed inappropriate to worry about surreptitious forwarding of signed-and-encrypted mail.

More recently, the XML-Signature and XML-Encryption working groups explicitly decided, from the outset of their work, to emulate S/MIME's security. Both groups decided not to address S/MIME's and PKCS#7's vulnerability to surreptitious forwarding, for three related reasons:

1. XML-Signature and XML-Encryption are explicitly low-level protocols. Thus, the XML-security standards mustn't force higher-level protocols to follow a particular cryptographic model.
2. The W3C intends that for XML documents, format specifications and semantics specifications should generally be kept separate. Accordingly, surreptitious forwarding, being an issue of Sign & Encrypt "semantics," should be treated in a separate XML Security Semantics specification.
3. A document-format working group shouldn't try to resolve questions about minute details of cryptographic implementation, because such discussions invariably become time-wasting "ratholes."

Thus, the XML-Security working groups seem to intend their specifications to be accepted as strictly "low-level" cryptographic primitives. It's hard, though, to reconcile this "low-level" label with these working groups' early proposal to emulate S/MIME, since S/MIME claims to offer high-level, comprehensive, and secure messaging.

It's hard to blame the secure-mail standards groups for having made a cryptographic mistake. Clearly, they all worked in good faith to promote secure and usable technologies. Further, it's important to acknowledge how hard it is to write networking standards in general, and mail-related standards in particular. As hard as it is to design cryptographic security protocols, cryptographic difficulty is only a formal or mathematical affair, and is very different from the difficulty of designing workable networking protocols for real-world deployment. In any design of a concrete security protocol, many hard problems have to be solved simultaneously, including:

- Flexibility for application programmers;
- Flexibility for network admins and sys-admins;
- Interoperation with other protocols;
- OS platform differences;
- Scaling;
- Server statelessness;
- Exportability;
- Time-to-market.

Clearly, each of the secure e-mail standards committees tried to codify a cryptographically correct protocol. The worst that can be said of these working groups is that they underestimated the subtlety of adding cryptography to their already-burdened portfolio.

5 Repair Options

We present five independent and equivalently-secure ways to fix the naïve Sign & Encrypt problem:

1. Sign the recipient's name into the plaintext, or
2. Encrypt the sender's name into the plaintext, or
3. Incorporate both names; or
4. Sign again the signed-&-encrypted message; or
5. Encrypt again the signed ciphertext.

In each case, the signing layer and the encryption layer become interdependent, binding the sender's name, in one layer, to the recipient's name in the other layer. Any one of these alternatives suffices to establish that Alice authored both the plaintext and the ciphertext. Note though that an effective security standard should require not only that the author must *provide* one of these five proofs, but also that the recipient must *demand* some such proof as well. That is, if a naïve Sign & Encrypt message arrives without proof that the signer and encryptor were the same person, then the application software should warn the recipient that the message's privacy and/or authenticity are suspect.

5.1 Naming Repairs

Perhaps part of the reason naïve Sign & Encrypt seems secure is that with many common payload messages, S&E *is* secure. For example, even if Alice just signs and encrypts the text "Dear Bob, The deal is off. Regretfully, Alice," then Alice's message is secure, albeit only accidentally so. The presence of names under both crypto layers is crucial, but including both names is not strictly necessary:

1. If Alice wants to use Sign & Encrypt, then she needs to enclose only Bob's name, because this will link the outer layer's key to the inner layer.

   $$A \rightarrow B \quad : \quad \{\{Bob,\ msg\}^a\}^B \tag{21}$$

   By signing Bob's name into her message, Alice explicitly identifies him as her intended recipient. This is equivalent to I'Anson's repair for X.509v1, as discussed above in Section 3.1.

2. If Alice prefers instead to use Encrypt & Sign, then she should encrypt her own name along with her message, and should sign her message-plaintext outside the ciphertext, so as to block Anderson's plaintext-replacement attack:

   $$A \rightarrow B \quad : \quad \{\{Alice,\ msg\}^B,\ \#msg\}^a \tag{22}$$

   Again, this links the outer layer's key-pair to the inner layer, and prevents an attacker from replacing Alice's signature. Encrypting the sender's name works in a subtle way to prove that Alice performed the encryption: The enclosed name shows that the encryptor intends for the outer signature to carry the same name (Alice's). The outer signature, in turn, says that Alice did indeed touch the ciphertext. Therefore, Bob knows that Alice performed the encryption.

3. If Alice encloses *both names* in the message-body, she can avoid having to pay attention to cryptographic choices early on, while she's formatting her message text. She can send to Bob in either of two ways:

   $$\begin{aligned} A \rightarrow B \quad &: \quad \{\{\text{"}A \rightarrow B\text{"},\ msg\}^B, \#msg\}^a \\ A \rightarrow B \quad &: \quad \{\{\text{"}A \rightarrow B\text{"},\ msg\}^a\}^B \end{aligned} \tag{23}$$

   These two-name formats might be suitable for a flexible standards-specification like S/MIME, in which the layers of crypto can be applied in any order. Always enclosing both names with the message is simpler than judging on the fly which names to enclose, depending on the choice of cryptographic wrappings.

These repairs are rational examples of Martín Abadi's and Catherine Meadows' rule-of-thumb for designing security protocols:

- Abadi: "If the identity of a principal is essential to the meaning of a message, it is prudent to mention the principal's name explicitly in the message." (Principle 3 in [1])

- Meadows: "In general, it's safer to include names explicitly inside crypto protocols' messages." [18]

5.2 Sign/Encrypt/Sign

Surprisingly, we can get an effective repair for S&E, if Alice signs and encrypts the plaintext, and then she signs the ciphertext, too:[8]

$$A \rightarrow B \quad : \quad \{\{\{msg\}^a\}^B\ ,\ \#B\}^a \tag{24}$$

(Here, $\#B$ means Alice hashes Bob's key, not his name.) This message means:

- Inner Signature: "Alice wrote the plaintext;"

- Encryption: "Only Bob can see the plaintext;"

- Outer Signature: "Alice used key B to encrypt."

Bob can conclude not only that Alice wrote the message, but that she also encrypted it. Seen another way, S/E/S is a variation on including the sender's name inside the plaintext, which then is encrypted and signed (see Sec. 5.1, bullet 2). The inner signature's key links the encryption-layer to the outer signature's layer. Alice signs Bob's key, so as to protect herself from Anderson's plaintext-substitution attack.

[8]For notational simplicity, we represent these signatures as $\{stuff\}^a$, instead of as $stuff$, $\{\#stuff\}^a$.

5.3 Encrypt/Sign/Encrypt

Conversely, Alice can get the same security guarantees by re-encrypting her ciphertext's signature:

$$A \rightarrow B : \{\{\{msg\}^B \ , \ \#msg\}^a\}^B \qquad (25)$$

This message means:

- First Encryption: "Only Bob sees the plaintext;"
- Signature: "Alice wrote the plaintext and the ciphertext;"
- Outer Encryption: "Only Bob can see that Alice wrote the plaintext and ciphertext."

Bob cannot forward the message without invalidating Alice's signature. The outer encryption serves to prevent an attacker from replacing Alice's signature. As with S/E/S, E/S/E is a variant of including the recipient's name inside the plaintext, which is then signed and encrypted (see Sec. 5.1, bullet 1). Alice signs her plaintext along with her ciphertext [27], so as to protect herself from Anderson's plaintext-substitution attack. At the same time, Alice's signed plaintext gives Bob non-repudiation.

5.4 Costs and Advantages

Of course, the naming repairs and the double-signed repairs offer different trade-offs. The naming repairs bring no performance cost, but they do require new standards, and those standards would arguably be more intricate than the current standards (because interdependence of layers conflicts with arbitrary nesting of layers). The double-signed repairs are quite expensive in speed, but they have two virtues:

- Double-signing is quite compatible with the existing CMS and S/MIME specifications. The only change double-signing would bring is that the standard would have to require that the recipient check the innermost layer's key against the outermost layer's key.
- For some applications, double-signing may be preferable to having to put names into message-bodies or payloads.

Overall, it's clear that the simplest repair is to add the recipient's name, then Sign & Encrypt(§ 5.1, bullets 1 and 3). The other solutions all require an extra hash of the message or of the encrypting key, so as to block Anderson's plaintext-replacement attack.

6 Conclusions

We have presented a forensic history of how naïve Sign & Encrypt, an insecure cryptographic primitive, has come to be widely trusted, standardized, and implemented, despite its insecurity. The notion that naïve Sign & Encrypt is secure seems to have arisen with PGP's first description in 1986. This mistake was reinforced by a misstatement in a paper that proposed several repairs for X.509v1. Since then, all of the leading standards for file-encryption and for secure e-mail have relied on naïve Sign & Encrypt. Some of these defective standards can be fixed easily, but for others, the repair would become intricate. Secure-session protocols and authentication protocols typically do not rely on naïve Sign & Encrypt, so they are not affected by this paper's findings.

The weakness of naïve Sign & Encrypt is somewhat subtle, but it is easily fixed in several ways. The repairs all show that Signing and Encryption should not be viewed as independent operations; the repairs presented here all rely on linking the outer operation's key to the inner operation's payload. This realization, that public-key operations are not necessarily so independent as they're commonly thought to be, and that coupling two layers together is a profitable primitive, may prove to be a novel and useful axiom for beginning protocol designers and analysts.

7 Acknowledgements

I have had profitable discussions about these ideas with many expert critics: Martín Abadi, Ross Anderson, Marc Branchaud, Dave Carver, Dan Geer, Peter Gutmann, Philip Hallam-Baker, Paul Hoffman, Russ Housley, Steve Kent, Norbert Leser, John Linn, Ellen McDermott, Joseph Reagle, Ed Simon, Win Treese, Charlie Reitzel, Ralph Swick, and Henry Tumblin. I thank all of these people for their patient attention.

References

[1] M. Abadi and R. Needham, "Prudent Engineering Practice for Cryptographic Protocols," *Digital SRC Research Report* #125 (June 1, 1994).

[2] R. Anderson and R. Needham, "Robustness Principles for Public Key Protocols," in *Lecture Notes in Computer Science* 963, Don Coppersmith (Ed.), Advances in Cryptology - CRYPTO '95, pp. 236-247. Springer-Verlag, 1995.

[3] S. Crocker, N. Freed, J. Galvin, S. Murphy, Internet RFC 1848 "MIME Object Security Services," October 1995. http://www.faqs.org/rfcs/rfc1848.html

[4] M. Burrows, M. Abadi, and R. Needham, "A Logic of Authentication," *Proc. R. Soc. Lond. A* 426(1989) pp. 233-271.

[5] T. Dierks and C. Allen, Internet RFC 2246 "The TLS Protocol Version 1.0," January 1999. ftp://ftp.isi.edu/in-notes/rfc2246.txt

[6] D. Eastlake, J. Reagle, and D. Solo (Editors), "XML-Signature Syntax and Processing: W3C Working Draft 18-September-2000," http://www.w3.org/TR/xmldsig-core/

[7] P. Hoffman, Internet RFC 2634 "Enhanced Security Services for S/MIME," June 1999. ftp://ftp.isi.edu/in-notes/rfc2634.txt

[8] Paul Hoffman, personal communication.

[9] R. Housley, Internet RFC 2630 "Cryptographic Message Syntax," June 1999. ftp://ftp.isi.edu/in-notes/rfc2630.txt

[10] Russ Housley, personal communication.

[11] C. I'Anson and C. Mitchell, "Security Defects in CCITT Recommendation X.509 - The Directory Authentication Framework," *ACM Comp. Comm. Rev.*, (Apr '90), pp. 30-34.

[12] B. Fraser, T.Y. Ts'o, J. Schiller, M. Leech, "IP Security Protocol (IPSEC) Charter," http://www.ietf.org/html.charters/ipsec-charter.html

[13] A. Joux and R. Lercier, "Discrete logarithms in GF(p)", April 17 2001. Announcement on the Number Theory Mailing List NMBRTHRY. http://www.medicis.polytechnique.fr/ /˜lercier/talk/ecc99.ps.gz .

[14] International Telegraph and Telephone Consultative Committee (CCITT). Recommendation X.509: The Directory - Authentication Framework. In *Data Communications Network Directory, Recommendations X.500-X.521*, pp. 48-81. Vol. 8, Fascicle 8.8 of *CCITT Blue Book*. Geneva: International Telecommunication Union, 1989.

[15] J. Linn, Internet RFCs 989, 1040, 1113, 1421, "Privacy Enhancements for Internet Electronic Mail: Part 1: Message Encryption and Authentication Procedures," February 1987, January 1988, August 1989, February 1993. ftp://ftp.isi.edu/in-notes/rfc{989,1040}.txt ftp://ftp.isi.edu/in-notes/rfc{1113,1421}.txt

[16] John Linn, personal communication.

[17] Stephen Kent, personal communication.

[18] C. Meadows, "Verification of Security Protocols," lecture at 1996 RSA Cryptographers' Colloquium, Palo Alto, CA.

[19] C. Neuman and J. Kohl, Internet RFC 1510 "The Kerberos Network Authentication Service (V5)", September 1993. ftp://ftp.isi.edu/in-notes/rfc1510.txt

[20] B. Ramsdell, Internet RFC 2633 "S/MIME Version 3 Message Specification," June 1999. ftp://ftp.isi.edu/in-notes/rfc2633.txt

[21] R. Rivest, A, Shamir, L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Comm. ACM*, v. 21, **2**, Feb. '78, pp. 120-126.

[22] RSA Laboratories, "PKCS#1: RSA Cryptography Standard," version 2.0. Amendment 1: "Multi-Prime RSA." http://www.rsasecurity.com/rsalabs/pkcs/pkcs-1/

[23] RSA Laboratories, "PKCS#7: Cryptographic Message Syntax Standard," Version 12.5, Nov. 1, 1993. http://www.rsasecurity.com/rsalabs/pkcs/pkcs-7/

[24] RSA Laboratories, "PKCS#9 v2.0: Selected Object Classes and Attribute Types," February 25, 2000. http://www.rsasecurity.com/rsalabs/pkcs/pkcs-9/

[25] Joseph Reagle, personal communication.

[26] J. Reagle, "XML Encryption Requirements," W3C Working Draft 2001-April-20, http://www.w3.org/TR/2001/WD-xml-encryption-req-20010420 .

[27] Ed Simon, personal communication.

[28] Visa International and MasterCard, "Secure Electronic Transactions Protocol Specification," http://www.setco.org/set_specifications.html .

[29] P. Zimmermann, "The Official PGP User's Guide," MIT Press (1995).

[30] P. Zimmermann, "A Proposed Standard Format for RSA Cryptosystems," IEEE Computer 19(9): 21-34 (1986).