Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1

Daniel Bleichenbacher

Bell Laboratories
700 Mountain Ave.
Murray Hill, NJ 07974
E-mail: bleichen@research.bell-labs.com

Abstract. This paper introduces a new adaptive chosen ciphertext attack against certain protocols based on RSA. We show that an RSA private-key operation can be performed if the attacker has access to an oracle that, for any chosen ciphertext, returns only one bit telling whether the ciphertext corresponds to some unknown block of data encrypted using PKCS #1. An example of a protocol susceptible to our attack is SSL V.3.0.
Keywords: chosen ciphertext attack, RSA, PKCS, SSL

1 Overview

In this paper, we analyze the following situation. Let n, e be an RSA public key, and let d be the corresponding secret key. Assume that an attacker has access to an oracle that, for any chosen ciphertext c, indicates whether the corresponding plaintext $c^d \bmod n$ has the correct format according to the RSA encryption standard PKCS #1.

We show how to use this oracle to decrypt or sign a message. The attacker carefully prepares ciphertexts that are sent to the oracle. Combining the returns from the oracle, the attacker gradually gains information on c^d. The chosen ciphertexts are based on previous outcomes of the oracle. Thus, this technique is an example of an adaptive chosen-ciphertext attack. Usually, a chosen ciphertext attack is based on the theoretical assumption that the attacker has access to a decryption device that returns the complete decryption for a chosen ciphertext. Hence, if a public-key cryptosystem is susceptible to a chosen-ciphertext attack, that often is considered to be only a theoretical weakness. However, the attack shown in this paper is practical, because it is easy to get the necessary information corresponding to the oracle reply. The attack can be carried out if, for example, the attacker has access to a server that accepts encrypted messages and returns an error message depending on whether the decrypted message is PKCS conforming.

This paper is organized as follows. We describe the RSA encryption standard PKCS #1 in Section 2. In Section 3, we describe and analyze our chosen-ciphertext attack. Different situations in which this attack can be carried out are listed in Section 4. We then analyze the vulnerability of SSL to our attack in Section 5. In Section 6, we report experiments with the technique. In Section 7, we conclude by offering recommendations.

2 PKCS #1

In this section, we describe briefly the RSA encryption standard PKCS #1; refer to [11] for details. Currently, there are three block formats: Block types 0 and 1 are reserved for digital signatures, and block type 2 is used for encryption. We describe only block type 2, because it is relevant for this paper.

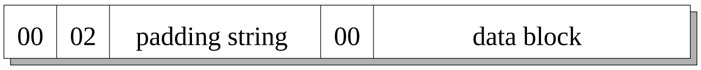

Fig. 1. PKCS #1 block format for encryption. The first two bytes in this format are constant. The length of the padding block can vary.

Let n, e be an RSA public key, and let p, q, d be the corresponding secret key (i.e, $n = pq$ and $d \equiv e^{-1} \pmod{\varphi(n)}$). Moreover, let k be the byte length of n. Hence, we have $2^{8(k-1)} \leq n < 2^{8k}$. A data block D, consisting of $|D|$ bytes, is encrypted as follows. First, a padding string PS, consisting of $k-3-|D|$ nonzero bytes, is generated pseudo-randomly. Here, $|D|$ must not exceed $k-11$; in other words, the byte length of PS is a least 8. Now, the encryption block $EB = 00||02||PS||00||D$ is formed (Figure 1), is converted into an integer x, and is encrypted with RSA, giving the ciphertext $c \equiv x^e \pmod{n}$. The representation of the ciphertext is not important for this paper.

We are, however, interested in how the receiver parses a ciphertext. First, he gets an integer x' by decrypting the ciphertext with his private key. Then, he converts x' into an encryption block EB$'$. Now he looks for the first zero byte, which indicates the ending of the padding string PS and the start of the data block D. The following definition specifies when this parsing process is successful.

Definition 1. An encryption block EB consisting of k bytes – that is,

$$\mathrm{EB} = \mathrm{EB}_1||\ldots||\mathrm{EB}_k$$

is called PKCS *conforming* – if it satisfies the requirements of block type 2 in PKCS #1. In particular, EB must satisfy the following conditions:

- $\mathrm{EB}_1 = 00$.
- $\mathrm{EB}_2 = 02$.
- EB_3 through EB_{10} are nonzero.
- At least one of the bytes EB_{11} through EB_k is 00.

We also call a ciphertext c PKCS conforming if its decryption is PKCS conforming.

Note that the definition of *conforming* does not include possible integrity checks. We show in Section 3 that it should not be possible for an attacker to decide whether a chosen ciphertext is PKCS conforming. It is sometimes possible for an attacker to do so even if the data block contains further integrity checks.

3 Chosen-Ciphertext Attacks

In a chosen-ciphertext attack, the attacker selects the ciphertext, sends it to the victim, and is given in return the corresponding plaintext or some part thereof. A chosen-plaintext attack is called *adaptive* if the attacker can chose the ciphertexts depending on previous outcomes of the attack.

It is well known that plain RSA is susceptible to a chosen-ciphertext attack [5]. An attacker who wishes to find the decryption $m \equiv c^d \pmod{n}$ of a ciphertext c can chose a random integer s and ask for the decryption of the innocent-looking message $c' \equiv s^e c \bmod n$. From the answer $m' \equiv (c')^d$, it is easy to recover the original message, because $m \equiv m's^{-1} \pmod{n}$.

Another well-known result is that the least significant bit of RSA encryption is as secure as the whole message [8] (see also [1]). In particular, there exists an algorithm that can decrypt a ciphertext if there exists another algorithm that can predict the least significant bit of a message given only the corresponding ciphertext and the public key. Håstad and Näslund recently extended this result to show that all individual RSA bits are secure [9].

Hence, it is not necessary for an attacker to learn the complete decrypted message in a chosen-ciphertext attack: Single bits per chosen ciphertext may be sufficient.

The result reported in this paper is similar. We assume that the attacker has access to an oracle that, for every ciphertext, returns whether the corresponding plaintext is PKCS conforming. We show that we can use this oracle to compute $c^d \pmod{n}$ for any chosen integer c. Theoretically, we can use Håstad's and Näslund's algorithm [9] to find c. In this paper, we describe a different algorithm that has as its goal to minimize the number of chosen ciphertexts; thus, we show the practicality of the attack. That is, we are not trying to generalize the attack; rather, we would like to take advantage of specific properties of PKCS #1. In particular, the algorithm relies on the facts that the first two bytes of the PKCS #1 format are constant, and that we know these two bytes with certainty when a ciphertext is accepted. Also, we use heuristic arguments in our the analysis of the algorithm to approximate the number of expected chosen ciphertexts, rather than finding an upper bound.

3.1 Description of the Attack

First, we give a short overview over the attack; then, we describe the attack in detail.

Assume that the attacker wants to find $m \equiv c^d \pmod{n}$, where c is an arbitrary integer. Basically, the attacker chooses integers s, computes

$$c' \equiv cs^e \pmod{n},$$

and sends c' to the oracle. If the oracle says that c' is PKCS conforming, then the attacker knows that the first two bytes of ms are 00 and 02. For convenience, let

$$B = 2^{8(k-2)}.$$

Recall that k is the length of n in bytes. Hence, that ms is PKCS conforming implies that

$$2B \leq ms \bmod n < 3B.$$

By collecting several such pieces of information, we can eventually derive m. Typically, 2^{20} chosen ciphertexts will be sufficient, but this number varies widely depending on numerous implementation details.

The attack can be divided into three phases. In the first phase, the message is blinded, giving a ciphertext c_0 that corresponds to an unknown message m_0. In the second phase, the attacker tries to find small values s_i for which the ciphertext $c_0(s_i)^e \bmod n$ is PKCS conforming. For each successful value for s_i, the attacker computes, using previous knowledge about m_0, a set of intervals that must contain m_0. We elaborate this process later. The third phase starts when only one interval remains. Then, the attacker has sufficient information about m_0 to choose s_i such that $c_0(s_i)^e \bmod n$ is much more likely to be PKCS conforming than is a randomly chosen message. The size of s_i is increased gradually, narrowing the possible range of m_0 until only one possible value remains.

Now we describe this attack in detail. The variable M_i will always be a set of (closed) intervals that is computed after a successful s_i has been found, such that m_0 is contained in one of the intervals of M_i.

Step 1: Blinding. Given an integer c, choose different random integers s_0; then check, by accessing the oracle, whether $c(s_0)^e \bmod n$ is PKCS conforming. For the first successful value s_0, set

$$\begin{aligned} c_0 &\leftarrow c(s_0)^e \bmod n \\ M_0 &\leftarrow \{[2B, 3B-1]\} \\ i &\leftarrow 1. \end{aligned}$$

Step 2: Searching for PKCS conforming messages.

Step 2.a: Starting the search. If $i = 1$, then search for the smallest positive integer $s_1 \geq n/(3B)$, such that the ciphertext $c_0(s_1)^e \bmod n$ is PKCS conforming.

Step 2.b: Searching with more than one interval left. Otherwise, if $i > 1$ and the number of intervals in M_{i-1} is at least 2, then search for the smallest integer $s_i > s_{i-1}$, such that the ciphertext $c_0(s_i)^e \bmod n$ is PKCS conforming.

Step 2.c: Searching with one interval left. Otherwise, if M_{i-1} contains exactly one interval (i.e., $M_{i-1} = \{[a, b]\}$), then choose small integer values r_i, s_i such that

$$r_i \geq 2\frac{bs_{i-1} - 2B}{n} \tag{1}$$

and

$$\frac{2B + r_i n}{b} \leq s_i < \frac{3B + r_i n}{a}, \tag{2}$$

until the ciphertext $c_0(s_i)^e \bmod n$ is PKCS conforming.

Step 3: Narrowing the set of solutions. After s_i has been found, the set M_i is computed as

$$M_i \leftarrow \bigcup_{(a,b,r)} \left\{ \left[\max\left(a, \left\lceil \frac{2B + rn}{s_i} \right\rceil\right), \min\left(b, \left\lfloor \frac{3B - 1 + rn}{s_i} \right\rfloor\right) \right] \right\} \tag{3}$$

$$\text{for all } [a, b] \in M_{i-1} \text{ and } \frac{as_i - 3B + 1}{n} \leq r \leq \frac{bs_i - 2B}{n}.$$

Step 4: Computing the solution. If M_i contains only one interval of length 1 (i.e., $M_i = \{[a, a]\}$), then set $m \leftarrow a(s_0)^{-1} \bmod n$, and return m as solution of $m \equiv c^d \pmod{n}$. Otherwise, set $i \leftarrow i + 1$ and go to step 2.

Remarks. Step 1 can be skipped if c is already PKCS conforming (i.e., when c is an encrypted message). In that case, we set $s_0 \leftarrow 1$. However, step 1 is always necessary for computing a signature, even if we do not wish to get a blind signature.

In Step 2.a, we start with $s_1 = \lceil n/(3B) \rceil$, because, for smaller values $m_0 s_1$ is never PKCS conforming.

We use condition (1) because we want to divide the remaining interval in each iteration roughly in half.

We can often improve the attack by using more information. For example, we have not used the fact that any PKCS-conforming message $m_0 s_i$ contains at least one zero byte. Moreover, if the attack is performed in a client–server environment, where both parties use the message $m_0 s_i$ to generate session keys, we might be able to find this message by exhaustive search if we already knew a sufficient portion of it.

3.2 Analysis of the Attack

We now analyze the correctness of the attack and approximate the complexity of, and, in particular, the number of oracle accesses necessary for, this attack. We must make a few heuristic assumptions; hence, we cannot give a rigorous proof of our result.

First, we approximate the probability $\Pr(P)$ that a randomly chosen integer $0 \leq m < n$ is PKCS conforming. Let $\Pr(A) = \frac{B}{n}$ be the probability that, for a randomly chosen integer, the first two bytes are 00 and 02, respectively. Since we have $2^{16}B > n > 2^8 B$, it follows that

$$2^{-16} < \Pr(A) < 2^{-8}.$$

The RSA modulus is usually chosen to be a multiple of 8; hence, $\Pr(A)$ will usually be close to 2^{-16}. The probability that the padding block PS contains at least 8 non-zero bytes followed by a zero byte is

$$\Pr(P|A) = \left(\frac{255}{256}\right)^8 \cdot \left(1 - \left(\frac{255}{256}\right)^{k-10}\right).$$

Assuming a modulus n of at least 512 bit (i.e. $k \geq 64$), we have

$$0.18 < \Pr(P|A) < 0.97;$$

hence, we have

$$0.18 \cdot 2^{-16} < \Pr(P) < 0.97 \cdot 2^{-8}.$$

Next, we explain why our algorithm finds m_0 and thus m. We prove that $m_0 \in M_i$ for all i by induction over i. Since m_0 is PKCS conforming, we have $2B \leq m_0 \leq 3B - 1$, and so, trivially, $m_0 \in M_0$.

Now assume that $m_0 \in M_{i-1}$. Hence, there exists an interval $[a, b] \in M_{i-1}$ with $a \leq m_0 \leq b$. Since $m_0 s_i$ is PKCS conforming, there exists an integer r such that $2B \leq m_0 s_i - rn \leq 3B - 1$, and hence $as_i - (3B - 1) \leq rn \leq bs_i - 2B$. We also have

$$\frac{2B + rn}{s_i} \leq m_0 \leq \frac{3B - 1 + rn}{s_i}.$$

Hence, it follows from the definition of M_i that m_0 is contained in one of the intervals.

Now we analyze the complexity of the attack. The messages in step 1 are chosen randomly; therefore, this step needs about $1/\Pr(P)$ accesses to the oracle on average to find s_0. We assume again that, on average, we need $1/\Pr(P)$ accesses to the oracle to find s_i for $i \geq 1$ in step 2.a and 2.b. (See also the remark at the end of this section.)

Let ω_i be the number of intervals in M_i. Using heuristic arguments, we can expect that ω_i will satisfy the following equation for $i \geq 1$.

$$\omega_i \leq 1 + 2^{i-1} s_i \left(\frac{B}{n}\right)^i \tag{4}$$

Indeed, the length of an interval in M_i is upper bounded by $\left\lceil \frac{B}{s_i} \right\rceil$. The knowledge that $m_0 s_i$ is PKCS conforming alone would lead to $\lceil \frac{s_i B}{n} \rceil$ intervals of the form

$$I_r = \left[\left\lceil \frac{2B + rn}{s_i} \right\rceil, \left\lfloor \frac{3B - 1 + rn}{s_i} \right\rfloor\right], \tag{5}$$

since r can take at most $\lceil \frac{s_i B}{n} \rceil$ values in equation (3).

In particular, M_1 will contain about $\lceil \frac{s_1 B}{n} \rceil$ intervals. If $i > 1$, then each of the intervals I_r or a fraction of I_r is included in M_i if I_r overlaps with one interval of M_{i-1}. No interval I_r can overlap with two intervals in M_{i-1}. If intervals I_r were randomly distributed, then the probability that one intersects with M_{i-1} would be upper bounded by

$$\left(\frac{1}{s_i} + \frac{1}{s_{i-1}}\right)\omega_{i-1}.$$

Hence, we get Equation (4) by taking into account that one interval must contain m_0. In our case, we expect s_2 to be approximately $2/\Pr(P)$, and we have $2(B/n)^2/\Pr(P) = 2B/(n\Pr(P|A)) < 2B/(0.18n) < 1/20$. Hence, w_2 is 1 with high probability. Thus, we expect that Step 2.b will be executed only once.

Now we analyze Step 2.c. We have $M_i = \{[a, b]\}$; hence, $a \le m_0 \le b$, and thus

$$\frac{2B + r_i n}{b} \le \frac{2B + r_i n}{m_0} \le s_i \le \frac{3B - 1 + r_i n}{m_0} \le \frac{3B - 1 + r_i n}{a}.$$

The length of the interval $[\frac{2B+r_i n}{b}, \frac{3B-1+r_i n}{a}]$ is

$$\frac{3B - 1 + r_i n}{a} - \frac{2B + r_i n}{b} \ge \frac{3B - 1 + r_i n}{m_0} - \frac{2B + r_i n}{m_0} \ge \frac{B-1}{m_0} \ge \frac{1}{3}\frac{B-1}{B}.$$

Therefore, we can expect to find a pair r_i, s_i that satisfies (2) for about each third value of r_i that is tried. Thus, it seems easy to find such pairs r_i, s_i that satisfy (1) and (2) just by iterating through possible values for r_i.

The probability that $s_i \in [\frac{2B+r_i n}{m_0}, \frac{3B-1+r_i n}{m_0}]$ is roughly 1/2. Thus, we will find a PKCS-conforming s_i after trying about $2/\Pr(P|A)$ chosen ciphertexts.

Since the remaining interval in M_i is divided in half in each step 2.*c*, we expect to find m_0 with about

$$3/\Pr(P) + 16k/\Pr(P|A)$$

chosen ciphertexts, where k denotes the size of the modulus in bytes. For $\Pr(P) = 0.18 \cdot 2^{-16}$ and $k = 128$ (which corresponds to a 1024-bit modulus), we expect that the attack needs roughly 2^{20} chosen ciphertexts to succeed. The bit length of the modulus is usually a multiple of 8; hence, $\Pr(P)$ is close to $0.18 \cdot 2^{-16}$, as assumed previously.

Remarks. The probabilities in this section were computed under the assumption that the values s_i are independent of each other. We made that assumption to allow a heuristic analysis of the algorithm. However, the assumption may be wrong in special cases. For example, let us assume that m_0 and $s_i m_0$ are both PKCS conforming with padding strings of similar length; that is, we have, for some integer j,

$$\begin{aligned} m_0 &= 2 \cdot 2^{8(k-2)} + 2^{8j}\mathrm{PS} + D \\ s_i m_0 &= 2 \cdot 2^{8(k-2)} + 2^{8j}\mathrm{PS}' + D'. \end{aligned}$$

Then, $(2s_i - 1)m_0$ is PKCS conforming with high probability, since

$$(2s_i - 1)m_0 = 2 \cdot 2^{8(k-2)} + 2^{8j}(2\mathrm{PS}' - \mathrm{PS}) + 2D' - D$$

often is PKCS conforming too. We believe that such relations generally help the attacker, but it in certain situations the attack might require many more chosen ciphertexts than our analysis indicates.

Usually, the bit size of the RSA modulus is a multiple of 8. This choice is a good one, because, for such a modulus, $\Pr(P)$ is small. A modulus with a bit length $8k - 7$ would make the attack much easier, because, in that case, only about 2^{13} chosen messages would be necessary.

4 Access to an Oracle

In this section, we describe three situations in which an attacker could get access to an oracle.

4.1 Plain Encryption

Let us assume that a cryptographic protocol starts as follows. Alice generates a message m (e.g., a randomly chosen key). She encrypts it with PKCS #1, without applying any further integrity checks, and sends the ciphertext to Bob. Bob decrypts the message. If the format of the message is not PKCS conforming, then he returns an error; otherwise, he proceeds according to the protocol.

If Eve impersonates Alice, she can easily send messages to Bob and check them for conformance. Note that Eve's attack works even when the protocol includes strong authentication at a later step, since Eve has obtained useful information before she has to respond with an authenticated message.

Note that the RSA encryption standard PKCS #1 [11, page 8, note 3] recommends that a message digest be included before an RSA operation, but for only the signing procedure. Even though the standard mentions that an encrypted message does not ensure integrity by itself, the standard does not indicate where such an integrity check should be included.

4.2 Detailed Error Messages

Thus far, we have shown that a reliable integrity check is an important part of an RSA encryption. One way to include such a check is to let the sender sign the message with his private key, before he encrypts it with the receiver's public key. Then, an attacker can no longer hope to create a correct message by accident. Her attack will nonetheless be successful when, in the case of a failed verification, the receiver returns an error message that gives detailed information about where the verification failed. In particular, it would compromise security to return different error messages for a message that is not PKCS conforming and for a message where only the signature verification failed.

4.3 A Timing Attack

Certain applications combine encryption and signatures. In such cases, a reliable integrity check often is part of the signature, but is not included in the encryption. Let us assume that an encrypted message c is decrypted and verified as shown in the following pseudo-code:

1. Let $m \equiv c^d \pmod{n}$ be the RSA-decryption of c.
2. If m is not PKCS conforming, then reject.
3. Otherwise, verify the signature of m.
4. If the signature is not correct, then reject; otherwise, accept.

An attacker will not be able to generate a chosen ciphertext c such that this message has a correct signature. However, she will be able to generate messages such that c sometimes passes the check in step 2 and is rejected only after the signature is checked. Hence, by measuring the server's response time, an attacker could determine whether c is PKCS conforming. This *timing attack* is much easier to perform than is Kocher's timing attack [10], which measures the time difference of single modular multiplications – a small fraction of the time used for one exponentiation. In our case, however, we have to distinguish between performing only an decryption and performing both an decryption and a signature verification. In the worst case, the time for the signature verification could be significantly longer than the time for the decryption – when, for example, we have a 512-bit encryption key because of export restrictions, but we use a 2048-bit key to ensure strong authentication. In addition, the attacker can chose what signing key is sent to the server.

5 SSL V.3.0

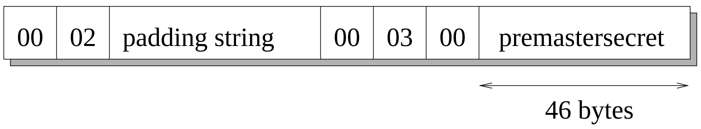

Fig. 2. SSL block format. Unlike the PKCS format, this format contains the SSL version number. Moreover, the length of the data block is constant.

The situation discussed in this paper arises in SSL V.3.0 [7] during the handshake protocol. In particular. the client and server first exchange the messages `client.hello` and `server.hello`, which, among other information exchanges, select the cryptographic routines. After that, the client and server may send their public keys and certificates. The client then generates a random secret bit string called `pre_master_secret`, encrypts that secret bit string with RSA (if that mode was chosen earlier), and sends the resulting ciphertext to the server. The server decrypts the ciphertext. If the plaintext is not PKCS conforming, the server sends an alert message to the client and closes the connection; otherwise, the server continues the handshake protocol. Finally, the client has to send a `finished` message, which contains strong authentication. In particular, the client has to know the `pre_master_secret` to compute that message.

Because an attacker must generate a `finished` message that depends on the `pre_master_secret`, she cannot complete the handshake protocol successfully. However, she does not have to complete it; she gets the necessary information – namely, whether her chosen message is PKCS conforming – before the protocol is finished.

There are details of SSL V.3.0 that might hinder this attack if they are implemented the right way. Figure 2 shows the format of the message containing the `pre_master_secret` before the latter is encrypted with RSA. It contains the version number of the protocol, the purpose of which is to detect *version-rollback attacks,* in which an attacker tries to modify the `hello` messages such that both client and server use the compatibility mode and hence use the Version 2.0, instead of Version 3.0, protocols. One implementation that we analyzed [12] checks the version number only if the server is running in the compatibility mode, because otherwise obviously no rollback attack has occurred.

A much more secure implementation would check the version number in all modes, and, if it identified a mismatch, would send back to the client the same error alert as it sends in the case of a decryption error. The result would be that a randomly generated message would be accepted with a probability of about 2^{-40}; although such a protocol still could not be called *secure,* the attack shown in this paper would at least be impractical.

The SSL documentation does not specify clearly the error conditions and corresponding alerts. As a result, different implementations of SSL do not react consistently with one another in error situations.

6 Experimental Results

We implemented the algorithm described in Section 3 and verified experimentally that this algorithm can decrypt a PKCS #1 encrypted message given access to an oracle that, for any ciphertext, indicates whether the the corresponding plaintext is PKCS conforming. We tested the algorithm with different 512-bit and 1024-bit keys. The algorithm needed between 300 thousand and 2 million chosen ciphertexts to find the message. We implemented our own version of the oracle, rather than using an existing software product.

Finney checked three different SSL servers [6] to find out how carefully the servers analyze the message format and what error alerts are returned. One of the servers verified only the PKCS format. The second server checked the PKCS format, message length, and version number, but returned different message alerts, thus still allowing our attack. Only the third server checked all aspects correctly and did not leak information by sending different alerts.

7 Conclusion

We have shown a chosen-ciphertext attack that can be carried out when only partial information about the corresponding message is leaked. We conclude not only that it is important to include a strong integrity check into an RSA encryption, but also that this integrity check must be performed in the correct step of the protocol – preferably immediately after decryption. The phase between decryption and integrity check is critical, because even sending out error messages can present a security risk. We also believe that we have provided a strong argument to use plaintext-aware encryption schemes, such as the one described by Bellare and Rogaway [3]. Note that plaintext awareness implies security against chosen-ciphertext attacks [2, 3]. In particular, Version 2 of PKCS #1, which makes use of [3], is not susceptible to the attack described in this paper.

It is a good idea to have a receiver check the integrity of a message immediately after decrypting that message. Even better is to check integrity before decrypting a message, as Cramer and Shoup show is possible [4].

Acknowledgments

I thank Markus Jakobsson, David M. Kristol, and Jean-François Misarsky, as well as the members of the program committee, for all their comments and suggestions. I am grateful for the cooperation of the people at RSA Laboratories. I thank Hal Finney for telling me about his experiments on different SSL servers. I am also grateful to Lyn Dupré for editing this paper.

References

1. W. Alexi, B. Chor, O. Goldreich, and P. Schnorr. Bit security of RSA and Rabin functions. *SIAM Journal of computing*, 17(2):194–209, Apr. 1988.
2. M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryptions schemes. In H. Krawczyk, editor, *Advances in Cryptology – CRYPTO '98*, Lecture Notes in Computer Science. Springer Verlag. (in press).
3. M. Bellare and P. Rogaway. Optimal asymmetric encryption. In A. D. Santis, editor, *Advances in Cryptology – EUROCRYPT '94*, volume 950 of *Lecture Notes in Computer Science*, pages 92–111, Berlin, 1995. Springer Verlag.
4. R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In H. Krawczyk, editor, *Advances in Cryptology – CRYPTO '98*, Lecture Notes in Computer Science. Springer Verlag. (in press).
5. G. I. Davida. Chosen signature cryptanalysis of the RSA (MIT) public key cryptosystem. Technical Report TR-CS-82-2, Departement of Electrical Engineering and Computer Science, University of Wisconsin, Milwaukee, 1982.
6. H. Finney. personal communication.
7. A. O. Freier, P. Karlton, and P. C. Kocher. *The SSL Protocol, Version 3.0.* Netscape, Mountain View, CA, 96.

8. S. Goldwasser, S. Micali, and P. Tong. Why and how to establish a private code on a public network. In *Proc. 23rd IEEE Symp. on Foundations of Comp. Science*, pages 134–144, Chicago, 1982.
9. J. Håstad and M. Näslund. The security of individual RSA bits. manusrcipt, 1998.
10. P. C. Kocher. Timing attacks on implementations of Diffie–Hellman RSA, DSS, and other systems. In N. Koblitz, editor, *Advances in Cryptology – CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113, Berlin, 1996. Springer Verlag.
11. RSA Data Security, Inc. *PKCS #1: RSA Encryption Standard.* Redwood City, CA, Nov. 1993. Version 1.5.
12. E. A. Young. SSLeay 0.8.1. url = `http://www.cryptsoft.com/`