Adaptive Chosen Ciphertext Attack on the RSA PKCS#1 Standard

Brian Graversen

University of Aarhus, 2007

Contents

1 Introduction

In 1998 Daniel Bleichenbacher presented a paper on a practical attack against the implementation of the RSA PKCS#1 encryption standard[1], and the first chapter of this paper is dedicated to describing the attack, and talking about an actual implementation.

Throughout this paper we will let (n, e) denote the public key and (n, d) the corresponding private key, and let k be the byte length of n, giving us that

$$2^{8(k-1)} \leq n < 2^{8k}$$

Let us outline of the idea behind the attack. Consider the scenario where Alice wants to send a message m to Bob. Alice will encrypt the message using Bobs public key (n, e), such that the ciphertext $c = m^e \pmod{n}$. In this scenario this encryped message is intercepted by our attacker, and the attackers goal is to decrypt the message.

We will allow the attacker to send encrypted messages c_i to Bob, which Bob will either accept or reject, depending on whether the corresponding plaintext m_i is PKCS#1 conforming.

From the original ciphertext c, we will construct a series of ciphertexts, send them to Bob, and from the output we will eventually have enough information to decrypt the original message m. Each constructed ciphertext in this series depends on the outcome of the previous attempt, and we will show that each time we find a new ciphertext where the corresponding cleartext is PKCS#1 conforming, we come closer to finding m.

1.1 The PKCS#1 v1.5 Standard

A PKCS#1 conforming message is k bytes long, and consists of four blocks, of the following form

$$\text{00 02} \parallel \text{padding block} \parallel \text{00} \parallel \text{data block}$$

The first block consists of two bytes, 00 followed by the block type (02 for encryption). The second block is the padding block, and must be at least 8 characters long, and contain random values different from 00. The third block is the seperator between the padding block and the data block, and must always have the value 00. Finally comes the data block, containing the data to be encrypted.

Since we know the first two bytes of a PKCS#1 conforming message m, we note that for such a message, interpreted as an integer, it holds that (for $B = 2^{8(k-2)}$)

$$2B \leq m < 3B$$

This gives us the basic idea behind the attack. For given a ciphertext $c = m^e \pmod{n}$, we can pick a random value $s < n$, encrypt it using the public key, and multiply it with the ciphertext c to get a new ciphertext

$$c' = s^e m^e = (sm)^e \qquad \pmod{n}$$

and if the decrypted message sm is PKCS#1 conforming, we have gained some information about m, since we know s. The next chapter will describe this attack in details, and later we will show that it actually works (and how fast).

1.2 The Bleichenbacher PKCS#1 Attack

In the following we assume we are given a ciphertext c, and it is our goal to find the corresponding plaintext m. We have access to the receivers public key, and can send encrypted messages to an oracle that can check if the corresponding plaintext is PKCS#1 conforming.

The attack described by Daniel Bleichenbacher is divided into 4 steps, each iterations through the steps will find an integer s_i, and a corresponding set of intervals M_i, from which we can find the plaintext m.

Remark 1.1. Step 1 in the description below is not needed when we are talking about using the attack for decryption, but it is included for completion. Step 1 is normally used to find some initial ciphertext c_0 where the corresponding plaintext m_0 is PKCS#1 conforming, but since m is already PKCS#1 conforming, we will set $c_0 = c$ and the value $s_0 = 1$ in the first step and skip directly to step 2. If this attack where to be used for forging RSA signatures, step one would be needed to find the initial PKCS#1 conforming message. Of course such an attack would only be possible of the person who's signature we are forging uses the same keypair for encryption and signatures.

(1) Choose at random values s_0, and calculate $c_0 = c(s_0)^e \pmod{n}$ until a query to the oracle states that the corresponding plaintext to c_0, namely m_0 is PKCS#1 conforming. Once such a value s_0 has been found, store c_0 and set

$$\begin{aligned} M_0 &= [2B, 3B-1] \\ i &= 1 \end{aligned}$$

(2) If $i = 1$ do step (2a), otherwise if the number of intervals in $M_{i-1} > 1$ then do step (2b) otherwise do step (2c).

(2a) Find the smallest integer $s_i \geq \frac{n}{3B}$, such that the plaintext corresponding to the ciphertext $c_0(s_1)^e \pmod{n}$ is PKCS#1 conforming.

(2b) Find the smallest integer $s_i > s_{i-1}$, such that the plaintext corresponding to the ciphertext $c_0(s_i)^e \pmod{n}$ is PKCS#1 conforming.

(2c) Let $[a, b]$ denote the interval in M_{i-1}, and choose small integers r_i, s_i, such that

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

and

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

until the plaintext corresponding to the ciphertext $c_0(s_i)^e \pmod{n}$ is PKCS#1 conforming.

(3) After s_i has been found, we will reduce the previous set M_{i-1} and store the result as M_i. This is done by the following formula.

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

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

(4) If M_i only contains a single interval $[a, a]$ of length 1, then we are done. The resulting cleartext is $m = a(s_0)^{-1}$. If M_i contains more than one interval, or one interval with length > 1, then increase i by one and continue from step 2.

1.3 Analysis of the Attack

In this chapter we will do two things. We will first prove that the algorithm described actually terminates, and that it finds the message m. Next we will attempt to analyze the complexity of the algorithm, that is, how many attempts must we use before we can expect to find m.

From step 1 (blinding) we have $c_0 = c(s_0)^e \pmod{n}$, so if we can find the corresponding plaintext m_0, we can find m, since

$$m_0^e = (ms_0)^e \Rightarrow m_0 = ms_0 \Rightarrow m = m_0(s_0)^{-1}$$

Since Step 4 uses the final value a in the set M_i, and returns exactly $a(s_0)^{-1}$, we only need to show that $m_0 \in M_i \; \forall i$.

Since m_0 is PKCS#1 conforming, it is obviously in M_0 (the set of all PKCS#1 conforming messages). We will now look at Step 3, and use induction over i, to prove that $m_0 \in M_i$ given that $m_0 \in M_{i-1}$.

Assume that $m_0 \in M_{i-1}$, then there must exist some set $[a, b] \in M_{i-1}$, such that $m_0 \in [a, b]$. We know that $m_0 s_i$ is PKCS#1 conforming (this is how it was created in Step 2), giving us that

$$2B \leq m_0 s_i < 3B \qquad \pmod{n}$$

meaning that there exist some integer r, such that

$$2B \leq m_s s_i - rn \leq 3B - 1$$

From this we can deduce the following two formulas

(1) $as_i - (3B-1) \leq rn \leq bs_i - 2B$

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

Looking back on how M_i was defined, we have that $m_0 \in M_i$.

Now let us look at the complexity of the algorithm. For this purpose, we will calculate the chance Pr(P) that a message is PKCS#1 conforming. We recall that a message is PKCS#1 conforming (with regards to encryption) if the first two bytes are fixed as 00 02, followed by at least 8 bytes of non-zero padding. Now let

$$Pr(A) = \frac{B}{n}$$

Then Pr(A) is the chance that a random message has 00 02 as the first two bytes. Since we have that

$$2^{16}B = 2^{8k} > n > 2^{8(k-1)} = 2^8 B$$

it follows that

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

Actually since n is usually a multiply of 8, we are generally closer to 2^{-16} than 2^{-8}. Finally the probability that we have at least 8 padding characters followed by a 0 is

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

for $k = 64$ (a 512 bit RSA key) we have $Pr(P|A) = 0.18$, and as $k \to \infty$, $Pr(P|A) \to 0.97$. Combining these results (for worst case situations), we have that

$$Pr(P) = Pr(A)Pr(P|A) > 0.18 * 2^{-16}$$

and

$$Pr(P) = Pr(A)Pr(P|A) < 0.97 * 2^{-8}$$

For the rest of the text, we will assume worst case, and set $Pr(P) = 0.18 * 2^{-16}$.

Step 1, 2a and 2b simply tries different values (either at random or sequentially) to see if they are PKCS#1 conforming, so we can expect that we need $\frac{1}{Pr(P)}$ attempts, for each step, to find the s_i values.

Now let us consider the amount of intervals in M_i, and see if we can come to some conclusion as to how many times step 2b and 2c are executed.

First we notice that the lenghts of the intervals in M_i must be upper bounded by $\lceil \frac{B}{s_i} \rceil$, this follows from the construction of the intervals in step 3, where each interval I_r can be no larger than

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

If we use only the fact that $m_0 s_i$ is PKCS#1 conforming, we get from step 3, that there exist at most $\lceil \frac{s_i B}{n} \rceil$ intervals, since r can take no more than $\lceil \frac{s_i B}{n} \rceil$ values.

For M_i, where $i > 1$, we only keep the intervals (the intersecting part) that overlap with intervals from M_{i-1}. If the intervals in M_i are randomly distributed, we can expect that the probability that a given interval in M_i intersects with an interval from M_{i-1} (and thus is kept in M_i) would be upper bounded by

$$(\frac{1}{s_i} + \frac{1}{s_{i-1}})w_{i-1}$$

where w_{i-1} is the amount of intervals in M_{i-1}. Expanding this formula back to w_1, we find that w_i is upper bounded by

$$1 + 2^{i-1}s_i(\frac{B}{n})^i$$

We expect s_2 to be close to $\frac{2}{Pr(P)}$ ($s_0 = 1$, followed by 2 attempts at find a PKCS#1 conforming message), and we find that

$$\begin{array}{rclcl} s_2(\frac{B}{n})^2 & = & 2\frac{(\frac{B}{n})^2}{Pr(P)} & = & \frac{2B^2}{n^2 Pr(P)} \\ & = & \frac{2B^2}{n^2 Pr(P|A)Pr(A)} & = & \frac{2B^2}{n^2 Pr(P|A)\frac{B}{n}} \\ & = & \frac{2B}{nPr(P|A)} & < & \frac{2B}{0.18n} \\ & < & 0.05 & & \end{array}$$

Plug this into the formula for w_i, and we get that with a high probability, w_2 will be 1, and thus step 2b will be executed only once.

Let us take a look at step 2c.
We know that M_i only contains one interval $[a, b]$ and that $a \leq m_0 \leq b$, giving us

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

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

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

So we can expect to find a set s_i, r_i roughly every third r_i we try. The probability that $s_i \in [\frac{2B+r_i n}{m_0}, \frac{3B-1+r_i n}{m_0}]$ is roughly $\frac{1}{2}$, so we can expect to find a PKCS#1 conforming message after trying $\frac{2}{Pr(P|A)}$ ciphertexts.

Due to the construction of step 2c (the choice of r_i), the interval in M_i is divided roughly in half for each iteration, and thus we can expect to find m_0 with about

$$\frac{3}{Pr(P)} + \frac{16k}{Pr(P|A)}$$

chosen ciphertexts. For $Pr(P) = 2^{-16}$ and $k = 128$ (1024 bit RSA keys), we can expect the attack to succeed in roughly 2^{20} attempts.

1.4 Implementation Notes

I have implemented Bleichenbachers attack in Java, to test the effectiveness of the attack. Since I do not have legal access to a server on which I could attempt the attack, I opted to construct a fake scenario inside the program. The main program creates a client (Alice) and a server (Bob), and let Alice send an encrypted message to Bob. The message is intercepted by our attacker (the main program), which then uses Bob as the oracle, and attempts the attack described above.

Since both the attacker and the oracle shares resources, the program actually runs slower than would be expected in the wild (both encryption and decryption is done using the same resources). The attack is easily parallelized, so I take advantedge of the 2 CPU's in my machine, and for most of the attack, multithreading is used to great advantedge (step 2a and 2b).

On my testmachine, an Intel Core 2 powered machine, the attack could decrypt a message encrypted with a 1024 bit RSA key in a 8-12 minutes, taking roughly 50.000 chosen ciphertexts. The reason for the small amount of ciphertexts, is the first choice of implementation of the oracle that I made. the oracle was a bit too helpful, giving a BadPaddingException only if the first two bytes where wrong (the required 00 02), which speeds up the processes somewhat (an oracle that checks for full PKCS#1 conformity will actually reject messages that we could have used in this attack, since we only use the fact that we know the first two bytes). A few attempts with a more strict oracle (one which requires at least 8 bytes of not-null padding and one 00 byte), shows that the the attack slows down by several factors.

The final oracle, which checks for full PKCS#1 conformity, causes the attack to take around 20 minutes on my machine, and use around 200.000 chosen ciphertexts. These numbers matches the numbers stated in the Bleichenbacher paper[1], when taking into consideration that we are skipping step 1.

1.5 Other attack vectors

If we do not have access to an oracle, we may still be able to use the attack described above. When sending messages to the oracle, we no longer get information about the conformity of the messages, but instead we will measure the response times, using the assumption that a message that fails decryption due to padding (non-PKCS#1 conformity) will have a faster response time, than a message that is actually decrypted and processed by the application. Whether a timing attack can be made to work depends in many ways on the type of application/server the attack is used on, but it is not inconceivable that such an attack could be made to work.

1.6 Suggestions for fixing the problem

Even if the implementation of PKCS#1 v1.5 does not give away any information about the conformity of the messages, it is still might be possible to use timing attacks (as stated earlier), and Daniel Bleichenbacher suggests in his paper[1], that the use of plaintext-aware encryption schemes (a system where it is difficult to come up with valid ciphertexts without knowing the plaintext) should be considered. Luckily this is the case for v2.0 of the PKCS#1 standard, where OAEP (Optimal Asymmetric Encryption Padding) is used instead of the described PKCS#1 padding.

Though OAEP with RSA is proven secure against chosen ciphertext attacks, it might still be possible to use a similar attack as mentioned below.

1.7 Other Attacks

The revision of the PKCS#1 standard makes OAEP the recommended padding mechanism, to prevent the type of attack mentioned in the first chapter. But as we will see, it might still be possible to use a chosen ciphertext attack, at least if the implementation is flawed.

James Manger outlines such an attack in his paper[2], which uses an approach very similar to Bleichenbachers, but focuses on the two last steps in the decryption process, the step where the integer-to-octets conversion happens, and the OAEP decoding step. The PKCS#1 v2.0 standard states that it is important the the output in case of failure in these two steps be exactly the same, and James Mangers attack is an example of why this is important.

Danial Bleichenbacher has in 2006 presented another attack, where he shows it is possible to forge an RSA signature, if the public key has exponent 3, and the implementation of the signature validation is flawed.

When RSA PKCS#1 signature is validated, the signature is 'encrypted' using the senders public key, and then decoded according to the PKCS#1 specification, which looks like this

$$00\ 01 \parallel \text{padding block} \parallel 00 \parallel \text{HEADER HASH}$$

The HEADER is a ASN.1 sequence that contains information about the signature, for instance the type of hash function used (and thus the length of the hash value). In the attack, the padding is truncated, and some additional constructed garbage data is added to the end of the message, so it looks like this

$$00\ 01 \parallel \text{smaller padding block} \parallel 00 \parallel \text{HEADER HASH GARBAGE}$$

Bleichenbecher shows that it is possible, for keys with public exponent 3, to construct such a piece of garbage data, so the data can be signed without knowing the private key. The required weakness in the signature validation implementation, is that it trusts the ASN.1 header, and only reads as much data as the hash is supposed to be, and never checks that there is garbage data added to the message.

1.8 Conclusion

The strongest scheme can be made weak by a poor implementation (ie. giving away to much information to an attacker), and even widely used standards are often implemented with deviations from the standard (or just from poorly phrased standards), which can result in weaknesses not previously considered.

In Bleichenbachers paper[1], he writes that three different SSL servers where checked against the attack, and only one of these did not leak any information.

When professionals, who works with implementing security software on a day-to-day basis, can make these kinds of mistakes, it seems obvious that software houses that do not have security as a core competence should not attempt to implement security from scratch, but rather buy plug'n'play software or at least commercial crypto toolkits. Having a 3rd party provider for the cryptographic services needed for webservices, intranet portals, e-mail, etc. gives the business the option of chosing between several highly competent security companies, and replacing or upgrading broken components as needed, and for free they gain the benifit of a much more tested and used software (assuming that the company uses standard components).

References

[1] Bleichenbacher, D. *Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1.*

[2] Manger, J. *A Chosen Ciphertext Attack on RSA Optimal Asymmetric Encryption Padding (OAEP) as Standardized in PKCS #1 v2.0.*