„Bleichenbacher's RSA-Signature Forgery" – Solving Prof. Stamp's Challenge Number 21

5th January 2008

IT-Security und Kryptography
Supervisor: Bernhard Esslinger
University Siegen, Germany 2007

Steffen Rumpf
steffen.rumpf@withouthat.org

Contents

In the year 2006 *Daniel Bleichenbacher* demonstrates a attack on RSA-signatures with weak keys. This work describes this attack, how it is implemented and solves the challenge number 21 from *Prof. Mark Stamp.*

This is the summary of a seminar paper written on the university of Siegen (Germany) in the summer semester 2007. The original work (in german) can be obtained by the author of this paper.

1 Introduction

It is to be sayed, that there are many articles over this attack to be found on the web (s. [Izu07, NIS06, Ram06]). But most of them describe the attack only on the basis and do not show realy whats the clou of it. So I will try to explain how the attack works and why its not needed to know the private key of the sender. At first I will describe the basic functionality and then describe the mathematical basis. This paper is written according to [Fin06].

2 Bleichenbachers Attack – Basics

2.1 RSA

A RSA-key consists of serveral parts. The RSA-Modul N, the public exponent e and the private exponent d. To get this parts you have to select a public exponent e, which must be in case of this attack the value of 3. According to this e you have to choose two distinct odd primes p and q so that p-1 and q-1 have no common divisor with e. Then you can compute N by multiplying p and q ($N = p * q$). After that we can compute the totient of N ($\Phi(N) = (q - 1) * (p - 1)$). At last we can compute our private exponent d by sattisfying the concruence relation $de \equiv 1 \; mod \; \Phi(N)$ (s. [PKC93, P. 5]).

Let x be a encryption block (EB), which was created according to the PKCS#1 padding scheme. Than a signature can be created with the following function: $S = x^d \; mod \; N$. With this function the identity of the signer is secured, because the private exponent was used (excepting that no one other than the owner knows the private key).

To verify such a signature the inverse function is used: $x' = S^e \; mod \; N$. On the verification it must be checked that the resulting x is a valid EB and that the included hash value is similar to the hash value of the recieved message. Further more S^e must be larger then N because the full power of RSA lies in the modulus operation, so the overflow of this must be given. In case of the attack of *Daniel Bleichenbacher* this is not given, so the modulus operation is taken out of power and the verification will return the value of S^3 which is the forged EB'.

2.2 PKCS#1 Specification

The PKCS#1 specification defines the algorithm specific syntax of the private and public keys and the algorithms for the public key encryption and the signature process. Further more it describes a padding for this processes. Details to that specification can be found under [PKC93] and [PKC02].

PKCS#1 Padding Scheme

$$EB = 00\|BT\|PS\|00\|D^1 \tag{1}$$

Nomenclature:

EB = encryption block (used to build the signature)

BT = block type (one byte which defines the operation type)

PS = padding string (pads the EB to the length of n and consists only of bytes with the value 0xFF)

D = datablock (includes the OID[2] and the hash value of the message)

n = length of the RSA-Modulus (used to determ the length of the scheme)

00 = 0-Byte-Block (a zero byte block used as seperator)

The block type describes, wether the operation was a public or private key operation. In case of this attack (a signature process) the block type has the hexadecimal value 0x01. The padding string consists of many bytes so that the EB has the length of n. At last the datablock is an ASN.1 dataelement[3], which contains information over the used hash algorithm and the hash string[4]. This padding scheme is the basis of the described attack.

2.3 Forged PKCS#1 Padding Scheme

To perform the attack, the attacker attaches a aligned "garbage" value and truncates the padding string. This changes are made in order to have EB' a perfect cube. What

[1]|| is used for concatenation.
[2]ObjectIdentifier (OID) defines wich hash function is used.
[3]The definition can be found at [PKC02, S. 38]
[4]More information to ASN.1 can be found at [Gor92] and [Dub00]

this means will be shown in the next chapter. So it turns out, that the padding now has the following format:

Forged PKCS#1 padding scheme

$$EB' = 00\|BT\|PS\|00\|D\|G \tag{2}$$

Nomenclature:

G = Garbage (used to have the EB' as perfect cube)

The Function of EB as Perfect Cube

Considering, that the verification process of a signature is: $x' = S^e \ mod \ N$, where S is the signature, e the public exponent and N the RSA-modulus. So we can compute the cube root of the EB' and let it be our signature. So it turns out that this will avoid the modulus operation, because the value of S^e is the cube root of EB' and EB' even smaller as the N, because of the 00-byte-block at the beginning. The verfication process returns in this case S^e what was the value of the forged encryption block. This is described in detail, in the next chapter.

The implementations which accepts such a signature, failed to check that the datablock stands on the right side in the EB. They only searched the 00-byte-block between padding string and datablock and then read the ASN.1 data, which includes the length of itself.

After we have seen the basic procedure of the attack, I will now show what the mathematical background is.

3 Bleichenbacher's Attack

3.1 Number Representations of the PKCS#1 Padding Scheme

First of all we must remember, that the described EB is a octett string, which is binary coded. So the numerical value of this string is something like 2^x. The numerical value of EB can be described in general as:

$$2^x - 2^{de} + D * 2^{ds} + Garbage \tag{3}$$

D is the numerical value of the Datablock

N is a multiple of 3

n is the length of the RSA-modulus

x $= n - 15$; 15 Bits can be substracted, because the first two octetts in the PKCS#1 padding have together the value of 1 (00000000 00000001), the first 15 bits have the value of 0x00 and do not have any impact to the numerical value of the EB.

ds is the start position of the datablock

de is the end position of the datablock, respectiveley the new position of D from right

On the Crypto rump session *Bleichenbacher* gives an example with an 3072 Bit RSA-key, which make it more tangible, so I will use it to describe the mathematical background.

3.2 Example

First of all we have to initalize our variables:

n $= 3072$

x $= 3072 - 15 = 3057$

ds $= 2072$

de $= ds + 288 = 2360$

Later on we need:

N $= 2^{288} - D$

So we get following term:

$$2^{3057} - 2^{2360} + D * 2^{2072} + Garbage \tag{4}$$

On this values we can see that the datablock D is moved 2072 Bits from the right side and so we have, according to the given description before, a forged signature (Note: Along to the PKCS#1 spec, the datablock normaly stands on position 1 of the padding). This means in a schematic view to the octett string:

00000000 00000001 11111111 . . . 00000000 00110000 . . . 01101001 XXXXXXXX . . .

2^{3072} 2^{3057} 2^{2360} 2^{2072} Garbage $\rightarrow$

BT PS 0 Byte D

In the next step we put into term (4) the from N to D transformed definition of N, so that we get:

$$2^{3057} - 2^{2360} + (2^{288} - N) * 2^{2072} + Garbage \tag{5.1}$$

$$2^{3057} - 2^{2360} + 2^{2360} - N * 2^{2072} + Garbage \tag{5.2}$$

$$2^{3057} - N * 2^{2072} + Garbage \tag{5.3}$$

After that we can get the cube root of that, which broken implementations accept as a valid signature:

$$\sqrt[3]{2^{3057} - N * 2^{2072} + Garbage} = 2^{1019} - (N * 2^{34}/3) \tag{6}$$

Considering the description before, we now have our value which will be turn out the modulus operation because it is lesser than the RSA-modulus. To give a demonstration of that we can go backwards by just using the binomial formula of the 3rd grade.

$$(A - B)^3 = A^3 - 3A^2B + 3AB^2 - B^3 \tag{7}$$

So let 2^{1019} our A and $N * 2^{34}/3$ our B we get:

$$2^{3057} - N * 2^{2072} + (N^2 * 2^{1087}/3) - (N^3 * 2^{102}/27) \tag{8}$$

When we now compare our term (8) with our term (4), we see that the first part is identical and the garbage is described as a numeric value $(N^2 * 2^{1087}/3) - (N^3 * 2^{102}/27)$.

This short mathematical calculation was described from *Bleichenbacher* as "simple enough that it can be carried out by pencil an paper".

4 Prof. Stamp's Challenge

Among my term paper I had to code a example application to demonstrate the forgery. This implementation will go into a future release of the CrypTool[5] which is cordinated by my supervisor. The implementation is done in C# with .NET and the cryptography libary BouncyCastle[6].

One point to demonstrate the forgery was to solve the challenge from *Prof. Mark Stamp*. To solve this challenge, I had to forge a signature for a given padding and a given Message. The hash algorithm which has to be used was SHA-1 and the last limitation was that the signature should be verified with a 3072 bit key, which means that we have 3072 bit signature.

In difference to the given pattern I used the ObjectIdentifiers wich are specified in the PKCS#1 spec. But this makes no difference to the attack. To solve the challenge I used the given padding and attached the ObjectIdentifier for the used hash algorithm. After that I computed the hash value of the message, which was in case of the challenge "i forged the signature". This hash value was appended to the string too and then I choosed a number which I also appended. In my application this number was 125. The remaining bits of the signature are filled with zeros to pad it to n. Now I have to compute the cube root of this EB. For this I used the Newton-Method, which can be used to approximately calulate the roots of various grades. When I get this I had a forged signature.

In my implementation you have serveral methods to do the forgery and play around with different key sizes, positions of the datablock and hash algorithms.

[5]s. `http://www.cryptool.de`
[6]s. `http://www.bouncycastle.org`

5 Conclusion

The attack shows that during the implementation from specifications, the specification must be followed and understood, this means that all checks must be implemented. It must be observed, that an attacker must not follow the specification and can break it on other ways like shown here. The attacker forged the signature by simply manipulating the given pattern and the implementations failed to check, that the datablock was placed on the right side of the padding. They followed the specification and only searched the 00-byte-block between padding string and datablock, what is right when following the spec and considering that the given signature was created like described in the spec, and then read the ASN.1 data which includes the length of itself.

Further more it is advised to not use keys with a public exponent of 3 any more, to avoid this attack.

A Signature Examples

This signatures are a different to the given pattern from *Prof. Mark Stamp* because I used in my implementation the ObjectIdentifier given by the PKCS#1 spec. This ObjectIdentifier is the string: 30-21-30-09-06-05-2B-0E-03-02-1A-05-00-04-14 in difference to the given pattern, where the ObjectIdentifier was: fe-ad-17-63-de-bc-90-76-35-35-46-5d-6c-c6-19.

A.1 Correct Signature ($EB^d \; mod \; N$)

83-9D-47-B5-51-D4-2C-D2-CF-33-23-93-F0-15-F6-FD-5C-47-F3-C0-73-E0-67-42-1C-A9-34-99-05-AE-20-CB-60-A3-83-E7-72-D5-47-68-3E-90-

79-B9-34-07-88-3C-D4-51-05-07-DB-9D-5F-5F-88-3C-EF-18-1A-41-0D-57-25-06-7E-47-03-BE-3C-0D-BB-B0-EA-72-4C-A9-3B-CD-EE-E1-D4-

5D-7D-AC-2D-92-76-3C-AB-AD-94-C5-E7-9A-BC-A6-54-F3-65-AA-A4-DD-98-36-A7-10-1B-95-F7-75-4A-20-EE-1C-31-BE-5B-26-EE-38-E1-C4-

AA-1D-C0-DB-14-07-41-32-0A-16-AF-1E-F9-4C-DC-45-59-08-CA-7C-AB-AF-5A-9B-DE-2C-04-BE-CC-72-99-19-27-5A-C9-D8-E5-23-FF-FC-

D0-91-B2-7E-13-EA-DC-9C-38-DD-7B-79-4E-04-93-66-BE-D4-E7-DE-C7-BD-E5-C0-D8-B6-A7-CB-1E-BC-A0-CC-E7-2B-E6-10-AC-4D-DF-37-

5A-35-1E-AD-A5-4B-ED-77-9A-D0-76-56-E9-5E-C4-BF-9D-E0-FB-AC-3A-E5-02-EC-75-9E-28-8F-36-FA-61-41-0C-9D-49-B2-D7-E9-96-71-44-

A8-50-FB-D3-84-50-5B-45-52-23-D8-E5-07-1F-9D-96-2C-87-CF-2A-53-B3-F7-E3-42-87-66-14-13-E4-C0-28-67-3C-2C-C7-04-4F-F0-0A-F3-1E-

4B-98-51-B4-03-7D-E5-EB-00-0E-4D-57-B8-AD-1C-C4-46-95-5C-28-17-3C-FB-E6-B1-1A-11-FF-5B-29-FC-14-89-2D-EB-67-D3-17-2F-CF-26-

29-EF-1D-78-0C-19-56-B6-57-9C-CE-69-83-3F-7F-1F-34-B7-A7-B0-B1-98-5D-9A-C9-0B-A9-CD-CE-C0-5A-8E-31-EF-C0-8F-81-B7-F1-44-D8-

32-FF-10-57-E5-08-A7-CD-C4-9A-46-89-C0-65-B9

A.2 Decoded Correct Signature (EB)

Colormap: 00 || BT || PS || 00 || D

00-01-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-

FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-

FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-

FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-

FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-

FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-

FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-

FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-

FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-

FF-FF-FF-FF-FF-FF-00-30-21-30-09-06-05-2B-0E-03-02-1A-05-00-04-14-52-CB-03-62-8A-DA-97-B7-14-D0-1A-87-65-E5-EA-79-C4-98-04-46

A.3 Forged Signature ($\sqrt[3]{EB'}$)

```
00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-
00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-
00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-
00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-
00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-
00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-07-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-
FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-
FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FE-AA-EA-D6-EA-B6-
B2-B1-8E-BD-59-58-22-B1-55-5A-C5-C3-B9-59-D8-B9-23-74-F4-1B-C0-23-5F-32-87-E3-4D-06-20-05-B3-51-55-55-55
```

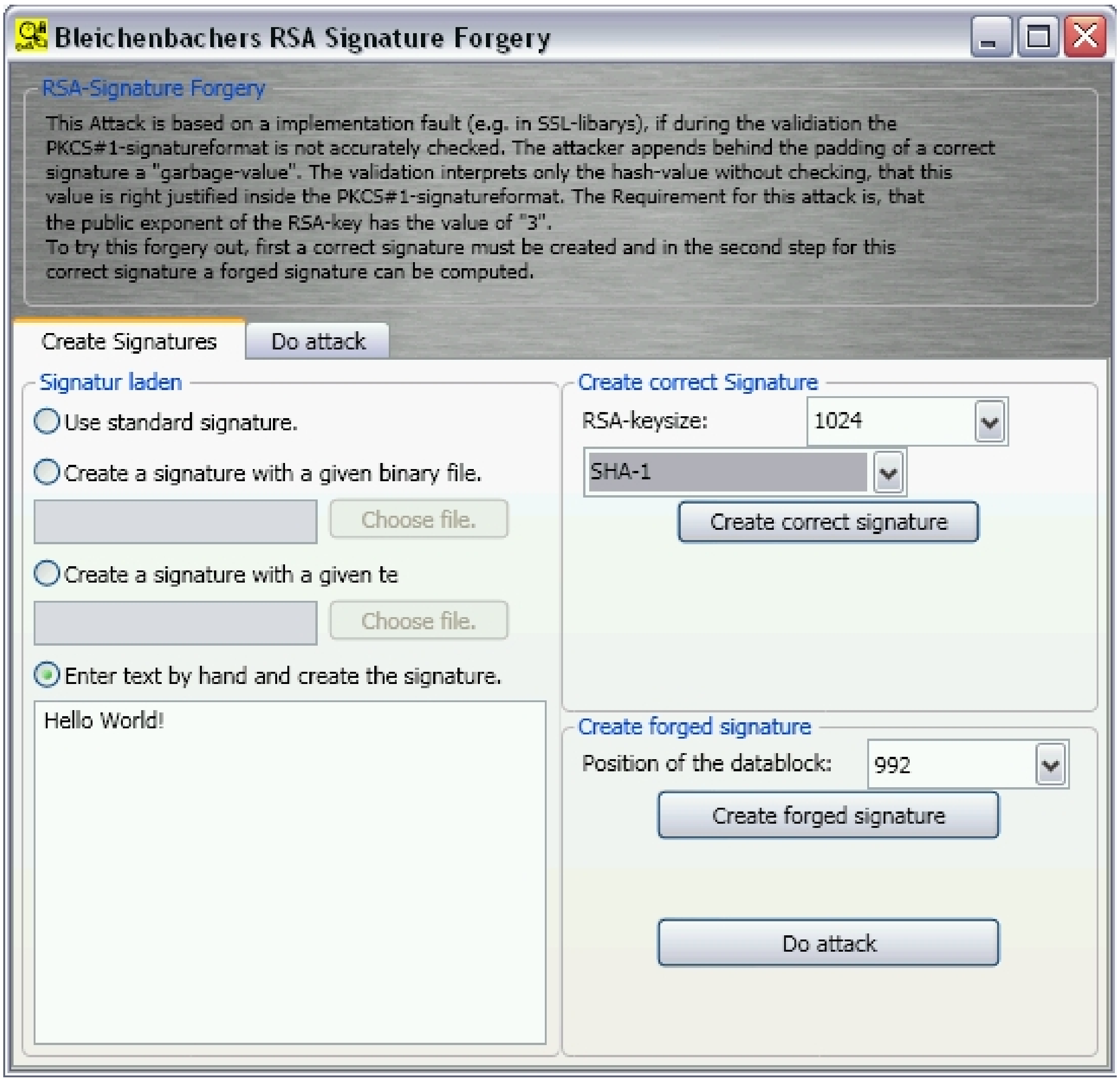
A.4 Decoded Forged Signature (EB')

Colormap: 00 || BT || PS || 00 || D || G

```
00-01-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-
FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-
FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-00-30-21-30-09-06-05-2B-0E-03-02-1A-05-00-04-14-52-CB-03-62-8A-DA-97-B7-14-D0-1A-87-65-
E5-EA-79-C4-98-04-46-7C-FF-FF-FF-C0-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-
00-00-00-00-00-00-00-00-00-00-00-00-00-2A-9A-A1-1C-BB-60-CB-35-CB-56-9D-DD-57-6C-27-29-6B-6B-AD-F8-20-BA-11-B6-8D-C1-24-31-3C-
8E-D2-14-B3-29-51-9B-BB-69-10-5B-B6-DD-E1-33-F3-77-FD-BD-21-38-31-D5-64-33-57-28-19-82-DA-AF-2C-43-86-C1-7B-CD-EF-73-3F-09-F5-
AD-F3-CA-2A-4F-75-95-55-55-57-FF-FF-FF-FF-FF-FF-FD-A2-85-69-4C-D9-34-7A-B7-52-8D-15-F9-D0-DB-F0-C7-E0-4A-BC-93-11-0B-78-5B-
A2-63-8D-BA-14-2B-E0-D2-14-82-7E-3E-0A-75-5B-F4-D4-A8-E3-DB-13-00-22-6F-E1-2F-12-41-B9-11-6F-1C-4F-E9-70-1C-FB-FE-3C-C3-70-
AF-70-46-44-D2-66-63-B4-25-9B-6C-09-2C-80-78-7E-CE-60-5B-51-2F-44-61-AC-73-54-11-9A-65-04-6D-B5-CD-4F-6C-8E-D2-E4-DE-89-5F-57-
30-39-D8-CF-5A-FB-75-EE-EF-EB-B4-25-ED
```

B Gui screenshot

B.1 Tab. 1



Bleichenbachers RSA Signature Forgery

RSA-Signature Forgery

This Attack is based on a implementation fault (e.g. in SSL-libarys), if during the validiation the PKCS#1-signatureformat is not accurately checked. The attacker appends behind the padding of a correct signature a "garbage-value". The validation interprets only the hash-value without checking, that this value is right justified inside the PKCS#1-signatureformat. The Requirement for this attack is, that the public exponent of the RSA-key has the value of "3".
To try this forgery out, first a correct signature must be created and in the second step for this correct signature a forged signature can be computed.

Create Signatures | Do attack

Signatur laden

- ○ Use standard signature.
- ○ Create a signature with a given binary file.
- ○ Create a signature with a given te
- ● Enter text by hand and create the signature.

Hello World!

Create correct Signature

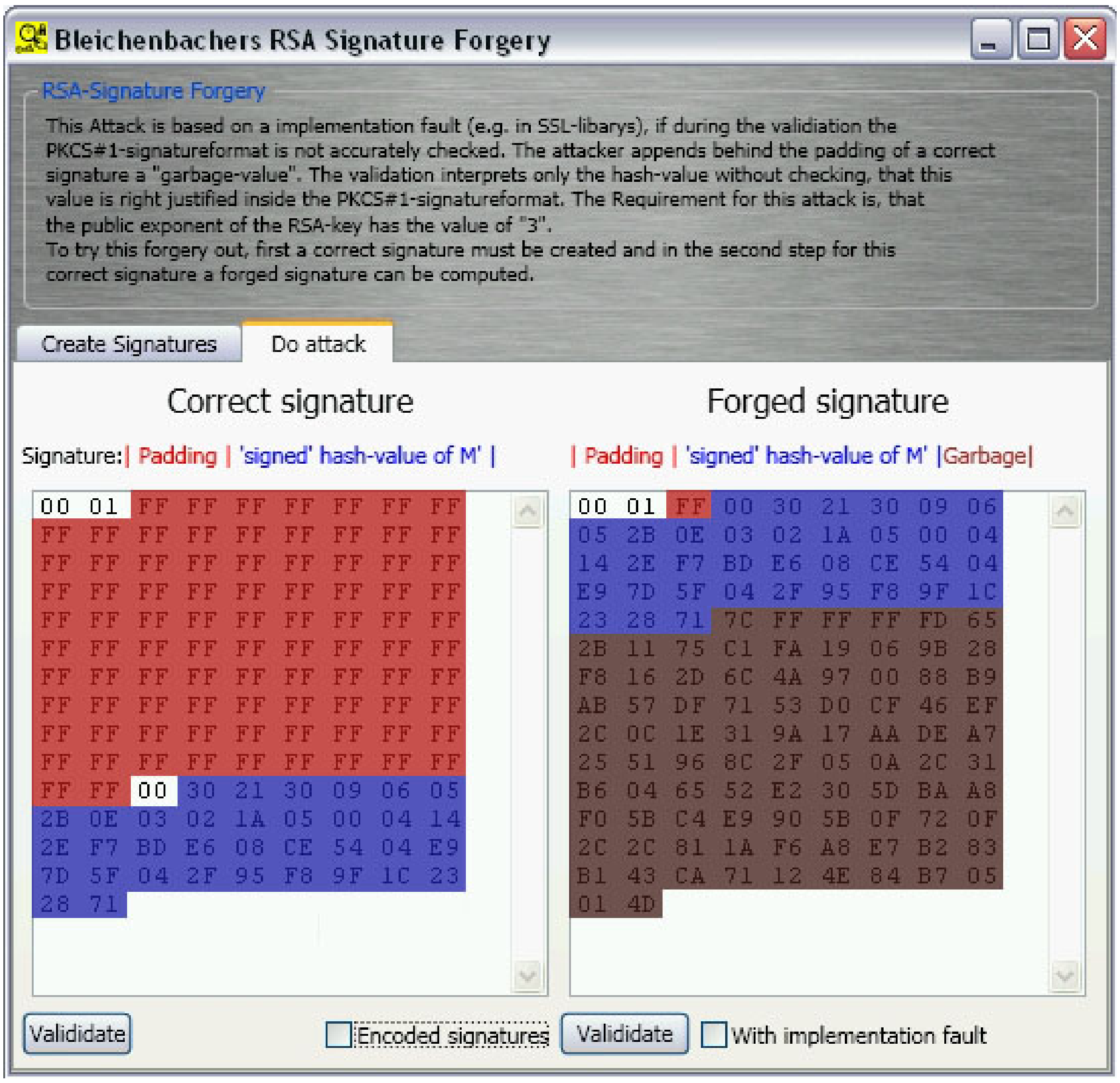
RSA-keysize: 1024

SHA-1

Create forged signature

Position of the datablock: 992

B.2 Tab. 2



Bleichenbachers RSA Signature Forgery

RSA-Signature Forgery

This Attack is based on a implementation fault (e.g. in SSL-libarys), if during the validiation the PKCS#1-signatureformat is not accurately checked. The attacker appends behind the padding of a correct signature a "garbage-value". The validation interprets only the hash-value without checking, that this value is right justified inside the PKCS#1-signatureformat. The Requirement for this attack is, that the public exponent of the RSA-key has the value of "3".
To try this forgery out, first a correct signature must be created and in the second step for this correct signature a forged signature can be computed.

Correct signature

Signature:| Padding | 'signed' hash-value of M' |

```
00 01 FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF
FF FF 00 30 21 30 09 06 05
2B 0E 03 02 1A 05 00 04 14
2E F7 BD E6 08 CE 54 04 E9
7D 5F 04 2F 95 F8 9F 1C 23
28 71
```

Forged signature

| Padding | 'signed' hash-value of M' |Garbage|

```
00 01 FF 00 30 21 30 09 06
05 2B 0E 03 02 1A 05 00 04
14 2E F7 BD E6 08 CE 54 04
E9 7D 5F 04 2F 95 F8 9F 1C
23 28 71 7C FF FF FF FD 65
2B 11 75 C1 FA 19 06 9B 28
F8 16 2D 6C 4A 97 00 88 B9
AB 57 DF 71 53 D0 CF 46 EF
2C 0C 1E 31 9A 17 AA DE A7
25 51 96 8C 2F 05 0A 2C 31
B6 04 65 52 E2 30 5D BA A8
F0 5B C4 E9 90 5B 0F 72 0F
2C 2C 81 1A F6 A8 E7 B2 83
B1 43 CA 71 12 4E 84 B7 05
01 4D
```

References

[Dub00] DUBUISSON, OLIVER: *ASN.1 – Communication between Heterogeneous Systems*. OSS Nokalva, 2000. – Free download: `http://www.oss.com/asn1/dubuisson.html`

[Fin06] FINNEY, HAL: *Bleichenbacher's RSA signature forgery based on implementation error*. `http://www.imc.org/ietf-openpgp/mail-archive/msg14307.html`, August 2006

[Gor92] GORA, WALTER: *ASN.1 – Abstract Syntax Notation One*. DATACOM-Verlag, 1992

[Izu07] IZU, TETSUYA ET AL.: Analysis of Bleichenbacher's Forgery Attack / Fujitsu Laboratories Ltd. 2007. – Forschungsbericht. – Second International Conference on Avaliablity, Reliability and Security (ARES'07)

[NIS06] NATIONAL INSTITUTE FOR STANDARDS AND TECHNOLOGY: An Attack on RSA Digital Signature. 2006. – Forschungsbericht. – `http://csrc.nist.gov/groups/ST/toolkit/documents/dss/RSAstatement_10-12-06.pdf`

[PKC93] RSA LABORATORIES: PKCS#1: RSA Encryption Standard. 1993. – Forschungsbericht

[PKC02] RSA LABORATORIES: PKCS#1 v2.1: RSA Cryptography Standard. 2002. – Forschungsbericht

[Ram06] RAMAZAN, Zulfikar: A common RSA implementation mistake explained / Symantec Corporation. 2006. – Forschungsbericht. – `http://www.symantec.com/enterprise/security_response/weblog/2006/10/a_common_rsa_implementation_mi.html`