Breaking RSA-based PIN Encryption with thirty ciphertext validity queries

N.P. Smart

Dept. Computer Science,
University of Bristol,
Merchant Venturers Building,
Woodland Road,
Bristol, BS8 1UB,
United Kingdom.
nigel@cs.bris.ac.uk

Abstract. We show that one can recover the PIN from a standardised RSA-based PIN encryption algorithm from a small number of queries to a ciphertext validity checking oracle. The validity checking oracle required is rather special and we discuss whether such oracles could be obtained in the real world. Our method works using a minor extension to the ideas of Bleichenbacher and Manger, in particular we obtain information from negative, as well as positive, responses from the validity checking oracle.

1 Introduction

Despite advances in provably secure cryptographic systems over the last decade or so, there are still a large number of systems deployed which do not use provably secure algorithms. This is mainly due to legacy reasons, and the problems of replacing or updating already deployed systems. In this paper we focus on a particular example of a non-provably secure encryption method based on RSA, namely the PIN encryption method in the EMV card payment system.

The RSA algorithm, being the earliest public key algorithm, was deployed in a number of systems before the advent of the provable security methodology. Probably the most famous example was the early adoption of the PKCS-v1.0 encryption method employed in SSL. This was famously attacked by Bleichenbacher [2], who used a ciphertext validity checking oracle, for the PKCS-v1.0 padding scheme, to recover the underlying RSA plaintext for a given challenge. The number of queries required by Bleichenbacher was linear in the bit-size of the RSA key, and always strictly larger. Hence, the number of challenges to the oracle was relatively large.

With the advent of provably secure systems such as OAEP [1] it appeared that RSA based systems could be deployed in a provably secure manner. However, Manger [7] showed how an extension to Bleichenbacher's attack could be deployed against RSA-OAEP when the ciphertext validity checking oracle returned different error messages when it failed for different events (similar data can be obtained via timing attacks on poor implementations). In particular the

OAEP padding scheme when applied to RSA can result in two possible errors occurring on decrypting an invalid ciphertext. The first error is that upon inverting the RSA function one obtains a number which is too large for the OAEP padding mechanism. The second error is that the OAEP padding mechanism returns an invalid ciphertext result. The security proof for OAEP assumes that the first error does not occur. But Manger, using an oracle which tells him whether the first error occurs as opposed to the second, managed to break the RSA-OAEP scheme, again with a number of ciphertext queries which was linear in (but strictly larger than) the bit-size of the public key.

In our paper we take a similar approach to Manger, in that we look at one out of a possible set of failure events. We examine the PIN encryption method in the EMV standard [4, 5]. This method is used to RSA encrypt four digit PIN numbers from a trusted terminal to a smart card. The smart card then decrypts the RSA ciphertext, recover the PIN and checks it against it's internally held PIN value. On decrypting the RSA ciphertext the card performs three validity checks. We assume, much as Manger does, that the failure or success of one of these checks, leaks to the attacker.

We then adapt the method of Bleichenbacher and Manger to this situation. In particular a number of simplifications can be performed. Firstly we are not interested in recovering the full RSA plaintext, we only care about the underlying PIN. Luckily, for the attacker, each PIN defines a distinct interval of possible RSA plaintexts. So our goal is to determine which of 10000 possible intervals the plaintext lies in. Secondly, and much for the same reason, we are able to use the failure of an oracle queries to reduce the possibilities for the underlying PIN. In the Bleichenbacher and Manger attacks only positive oracle responses are used to reduce the space of underlying plaintexts.

We end the paper by discussing the practical impact of our attacks, and point out that it is highly unlikely that our method will provide a means for an attacker to obtain PINs for the EMV system. However, our work does once again point out the need for old legacy systems to be replaced with new ones (EMVCo is already engaged in a process to update their standards due to the need to increase key sizes). In addition our method of using negative responses from the validity checking oracle may be useful in other applications.

Acknowledgements

The author was partially supported by the eCrypt-2 Network of Excellence, and a Royal Society Wolfson Merit Award.

2 PIN Encryption Method

We assume an RSA modulus N of t bits in length, where t is a precise multiple of eight. We write $k = t/8$ for the number of bytes in N. In this section we outline how PIN encryption and verification is performed in the EMV standard [4, 5] (the same method is used in the equivalent ISO standard [6]) First we cover the

creation of the PIN block, then we discuss how an RSA message is formatted, and finally we discuss verification. Full details can be found in [4, 5]. For later use we write

$$f(\mathfrak{a}, \mathfrak{b}, \mathfrak{u}, \mathfrak{r}) = \mathfrak{a} \cdot 2^{8(k-1)} + \mathfrak{b} \cdot 2^{8(k-9)} + \mathfrak{u} \cdot 2^{8(k-17)} + \mathfrak{r}$$

where $\mathfrak{a}$ is a one-byte value, $\mathfrak{b}$ and $\mathfrak{u}$ are eight bytes values and $\mathfrak{r}$ is a $k-17$ byte value.

PIN Block Format: A PIN block is defined to be a sequence of eight bytes, or equivalently sixteen nibbles. A PIN is assumed to be a sequence of between four and twelve integers in the range '0' to '9'. The PIN block is then defined to be (in nibbles)

C	N	P	P	P	P	P/F	P/F	P/F	P/F	P/F	P/F	P/F	P/F	F	F

where

- C is the number 2.
- N encodes the length of the PIN, i.e. it is a number between 0 and 12.
- P is a PIN digit between 0 and 9.
- F is a filler digit of 15 (i.e. the nibble with all ones set).

In the common case of a four digit PIN, $P_1 \| P_2 \| P_3 \| P_4$, then the value of the PIN block (as an integer) is given by

$$\begin{aligned}\mathfrak{b} &= 2 \cdot 2^{60} + 4 \cdot 2^{56} + (P_1 \cdot 2^{12} + P_2 \cdot 2^8 + P_3 \cdot 2^4 + P_4) \cdot 2^{40} + (2^{40} - 1) \\ &= (2^{61} + 2^{58} + 2^{40} - 1) + (P_1 \cdot 2^{12} + P_2 \cdot 2^8 + P_3 \cdot 2^4 + P_4) \cdot 2^{40}.\end{aligned}$$

RSA Message Format: Before encrypting a PIN the encryptor obtains an 8-byte nonce $\mathfrak{u}$ from the decryptor. This is used to avoid replay attacks. The encryptor then generates the pin block $\mathfrak{b}$ and creates a $k-17$ byte random number $\mathfrak{r}$. The message $\mathfrak{m}$ is then created as

$$\mathfrak{m} = f(127, \mathfrak{b}, \mathfrak{u}, \mathfrak{r}).$$

The encryptor then generates

$$\mathfrak{c} = \mathfrak{m}^e \pmod{N}$$

and sends $\mathfrak{c}$ to the verifier.

PIN Verification: The verifier, who knows the RSA secret key d, on obtaining a ciphertext $\mathfrak{c}'$ first obtains the underlying RSA plaintext $\mathfrak{m}'$, via

$$\mathfrak{m}' = \mathfrak{c}^d \pmod{N}.$$

Then the verifier recovers the values of $\mathfrak{a}', \mathfrak{b}', \mathfrak{u}', \mathfrak{r}'$ from the equation

$$\mathfrak{m}' = f(\mathfrak{a}', \mathfrak{b}', \mathfrak{u}', \mathfrak{r}').$$

Then he performs the following tests:

1. Return fail if $\mathfrak{u} \neq \mathfrak{u}'$, i.e. the nonce recovered is not equal to the nonce sent.
2. Return fail if $\mathfrak{a}' \neq 127$.
3. Return fail if the PIN in the PIN block $\mathfrak{b}'$ is not equal to the expected PIN.

we call the above tests Test 1, Test 2, and Test 3 in what follows. Note, that no mention is made in [4] of checking whether the PIN block is formatted correctly, only that the recovered PIN is correct.

In this paper we consider the situation when Test 2 is always executed for every ciphertext passed to the verifier and the attacker can determine whether the Test 2 returns fail or not. Later in Section 4 we shall discuss how realistic this is.

We note that there is a trivial attack on the PIN system, given a card one could run through all 10000 possible PIN blocks, obtain 10000 nonces, and then form 10000 ciphertexts. Each ciphertext is then passed to the card for checking, until one is returned as valid, in which case the attacker has determined the correct PIN. However, as we shall discuss later, the card has mechanisms built into it to protect against checking too many invalid PIN numbers. Hence, the question is how; few challenge ciphertexts are needed to recover a PIN?

3 The "Attack"

Our method is a simple extension of the methods of Bleichenbacher [2] as applied to the PKCS-v1.0 encryption scheme, and the method of Manger [7] as applied to the PKCS-v1.2 OAEP encryption scheme. In the original methods of Bleichenbacher and Manger the attacker is given a validity oracle which given a ciphertext returns whether the underlying plaintext lies in a given range (the range depending on the padding scheme being used). The attacker uses the positive responses from the oracle to essentially half the interval in which the plaintext behind a target ciphertext lies.

The main difference in our attack is that since we are only trying to determine one of a small number of PINs, rather than recovering the whole text, we are able to make use of negative results from our validity oracle. This comes at the expense of increasing the number of possible intervals for our target message. However, this does not result in an exponential blow-up since we have a small number of ranges in which a valid ciphertext could lie.

We describe our method in a bottom-up fashion by first describing Algorithm 1. This takes as input an interval $[a, b]$, and an integer s such that we know bounds L and U such that

$$L \leq s \cdot m \pmod{N} \leq U,$$

for a message $m \in [a, b]$. The algorithm works by evaluating all integers r such that

$$L \leq s \cdot m - r \cdot N \leq U,$$

which leads to

$$s \cdot a - U \leq s \cdot m - U \leq r \cdot N \leq s \cdot m - L \leq s \cdot b - L,$$

since $m \in [a, b]$. Then we deduce new bounds on m, for each of these values of r, from

$$L + r \cdot N \leq s \cdot m \leq U + r \cdot N.$$

Algorithm 1: UpdateInterval(a, b, L, U, s, N)

Input: a, b, L, U, s, N
Output: A list of intervals, $List$
$List \leftarrow \{\}$
for r **from** $\lceil (s \cdot a - U)/N \rceil$ **to** $\lfloor (s \cdot b - L)/N \rfloor$ **do**
  $a' \leftarrow \max(a, \lceil (L + r \cdot N)/s \rceil)$
  $b' \leftarrow \min(b, \lfloor (U + r \cdot N)/s \rfloor)$
  if $a' \leq b'$ **then**
    $List \leftarrow List \cup \{[a', b']\}$
return $List$

To describe the our next Algorithm 2, we assume we have an oracle $\mathcal{O}$ which on input of an RSA ciphertext $\mathfrak{c}'$ will return whether the underlying RSA message $\mathfrak{m}'$ lies in the interval $[\mathcal{L},\mathcal{U}]$, for specific fixed integers $\mathcal{L}$ and $\mathcal{U}$. Algorithm 2 takes as input a ciphertext $\mathfrak{c}$ a list of intervals $List$, for which we know that the underlying message $\mathfrak{m}$ lies in one of the intervals contained in $List$, and a "test" integer s.

Algorithm 2: UpdateList$(List, s, \mathfrak{c}, N, e)$

Input: $List, s, \mathfrak{c}, N, e$
Output: A new list of intervals, $List'$
$List' \leftarrow \{\}$
$\mathfrak{c}' = s^e \cdot \mathfrak{c} \pmod N$
$flag \leftarrow \mathcal{O}(\mathfrak{c}')$
forall $[a, b] \in List$ **do**
  if $flag$ **then**
    $List' = List' \cup$ UpdateInterval$(a, b, \mathcal{L},\mathcal{U}, s, N)$
  else
    $List' = List' \cup$ UpdateInterval$(a, b, 0, \mathcal{L}, s, N)$
    $List' = List' \cup$ UpdateInterval$(a, b,\mathcal{U}, N, s, N)$
return $List'$

Notice that UpdateList will create at most two intervals for every one in $List$ if the oracle returns $false$. This is the main difference between our method and that of Bleichenbacher and Manger. However, this only works due to the

nature of the underlying message we are trying to recover. It may appear that the size of $List'$ could become very large if UpdateList is repeatedly called, but we control the size of $List'$ using another list of intervals $TList$, which contains one interval for each PIN number. The $TList$ is created by Algorithm 3 which takes as input the value of the nonce $\mathfrak{u}$ which underlies in the target ciphertext $\mathfrak{c}$. In Algorithm 3 we assume that PINs are four digits in length. Notice, that all the intervals created are distinct.

Algorithm 3: CreateTList($\mathfrak{u}, k$)

Input: $\mathfrak{u}, k$
Output: $TList$
$TList \leftarrow \{\}$
forall $p_1, p_2, p_3, p_4 \in [0, 9]$ **do**
  $\mathfrak{b} \leftarrow$ the PIN block for this PIN.
  $a \leftarrow f(127, \mathfrak{b}, \mathfrak{u}, 0)$
  $b \leftarrow f(127, \mathfrak{b}, \mathfrak{u}, 2^{8(k-17)})$
  $TList \leftarrow TList \cup \{[[a, b], [p_1, p_2, p_3, p_4]]\}$
return $TList$

To filter the intervals we then use Algorithm 4. This takes a set of intervals $List$ for which one contains the target message, and a list $TList$ of intervals which contain the PINs. It then forms two new lists $List'$ and $TList'$. Clearly if Algorithm 4 ever finds returns $TList'$ containing only one element, then we have found the PIN.

Algorithm 4: Filter($List, TList$)

Input: $List, TList$
Output: $List', TList'$
$List' \leftarrow \{\}, \quad TList' \leftarrow \{\}$
forall $[a, b] \in List$ **do**
  $flag \leftarrow false$
  forall $[[a', b'], PIN] \in TList$ **do**
    if $a' \leq b$ *and* $a \leq b'$ **then**
      $TList' \leftarrow TList' \cup \{[[a', b'], PIN]\}$
      $flag \leftarrow true$
  if $flag$ **then**
    $List' \leftarrow List' \cup \{[a, b]\}$
return $List', \ TList'$

In Algorithm 5 we describe the main method, which takes as input a ciphertext $\mathfrak{c}$ corresponding to a PIN encryption (and the corresponding nonce $\mathfrak{u}$) and outputs the corresponding PIN. In this algorithm we choose the value of s to use in the call to the oracle so as to hopefully half the number of possible messages

on each call to the oracle. For a particular value of r we require that

$$\mathcal{L} \leq s \cdot m - rN \leq \mathcal{U}$$

and so we must have

$$\lceil (\mathcal{L} + r \cdot N)/b \rceil \leq s \leq \lfloor (\mathcal{U} + r \cdot N)/a \rfloor,$$

which explains the choice of s in Algorithm 5. However, the key is the choice of r.

If we assume an oracle call is successful then the range of the new maximum possible interval is given by $(\mathcal{U} - \mathcal{L})/s$. Which given our above bound on s is upper bounded by

$$b\left(\frac{\mathcal{U} - \mathcal{L}}{\mathcal{L} + r \cdot N}\right).$$

But we really want this last value to be less than $(b-a)/2$. Thus we require

$$b\left(\frac{\mathcal{U} - \mathcal{L}}{\mathcal{L} + r \cdot N}\right) \leq \frac{b-a}{2},$$

which implies

$$r \geq \frac{2 \cdot b \cdot (\mathcal{U} - \mathcal{L})/(b-a) - \mathcal{L}}{N}.$$

Algorithm 5: Main$(\mathfrak{c}, \mathfrak{u}, N, e)$

Input: $\mathfrak{c}$
Output: PIN
$TList \leftarrow$CreateList$(\mathfrak{u}, k)$
$\mathfrak{b}_0 \leftarrow$ the pin block corresponding to the PIN: $0, 0, 0, 0$
$\mathfrak{b}_9 \leftarrow$ the pin block corresponding to the PIN: $9, 9, 9, 9$
$a \leftarrow f(127, \mathfrak{b}_0, \mathfrak{u}, 0)$
$b \leftarrow f(127, \mathfrak{b}_9, \mathfrak{u}, 2^{8(k-17)})$
$List \leftarrow \{[a, b]\}$
$r \leftarrow 0$
while $\#TList > 1$ **do**
  $a \leftarrow \min\{a : [a, b] \in List\}$
  $b \leftarrow \max\{b : [a, b] \in List\}$
  $r \leftarrow \max(r + 1, \lceil (2 \cdot b \cdot (\mathcal{U} - \mathcal{L})/(b-a) - \mathcal{L})/N \rceil)$
  $s \leftarrow \lceil (\mathcal{L} + r \cdot N)/b \rceil$
  if $s \leq \lfloor (\mathcal{U} + r \cdot N)/a \rfloor$ **then**
    $List \leftarrow$UpdateList$(List, s, \mathfrak{c}, N, e)$
    $List, TList \leftarrow$Filter$(List, TList)$
$PIN \leftarrow TList[1][1]$
return PIN

To run the algorithm, which works for any oracle $\mathcal{O}$, although possibly not very efficiently, we need to specify the values of $\mathcal{L}$ and $\mathcal{U}$, which depend on precise validity checking oracle $\mathcal{O}$ that we use.

3.1 Experimental Results

We assume the validity checking oracle simply returns true if the leading byte of the plaintext is equal to 127, and it returns false otherwise. This means that our values for $\mathcal{L}$ and $\mathcal{U}$ are given by

$$\mathcal{L} = 127 \cdot 2^{8(k-1)},$$
$$\mathcal{U} = 127 \cdot 2^{8(k-1)} + 2^{8(k-1)} - 1.$$

If we use the parameters for RSA keys as defined in the EMVCo specification, then we find that the RSA key size is only 896 bits, i.e. $k = 112$. This is the size of the RSA modulus for the chip-and-pin cards public key.

We ran a series of 1000 experiments to emulate the above attack and recorded how many oracle queries were needed to recover the PIN. The percentages can be found in Table 1. Notice, that the number of oracle queries is incredibly low by modern cryptographic standards (even for a modulus of 896 bits). The reason is that although the message appears to be padded with a large amount of randomness, this randomness is not mixed in with the plaintext (like it would be with OAEP). In particular the combination of this with the small number of possible plaintexts (i.e. 10000) leads to a highly reduced number of oracle queries.

Table 1. Percentage of attacks with a given number of oracle queries

Range	%		Range	%		Range	%
$0-4$	0.0		$5-9$	0.3		$10-14$	6.9
$15-19$	25.9		$20-24$	28.2		$25-29$	13.7
$30-34$	7.4		$35-39$	4.6		$40-44$	3.0
$45-49$	2.5		$50-54$	0.3		$55-59$	1.3
$60-64$	1.2		$65-69$	0.6		$70-74$	0.4
$75-79$	0.5		$80-84$	0.1		$85-89$	0.0
$90-94$	0.1		$95-99$	0.1		$100-104$	0.2
$105-109$	0.1		$110-114$	0.1		$115-119$	0.4
$120-124$	0.1		$125-129$	0.1		$130-134$	0.2
$135-139$	0.1		$140-144$	0.1		≥ 145	1.5

So we see that 75% of all PINs can be recovered with less than 30 calls to the oracle.

4 Practical Attack Considerations

We now consider what are the practical consequences of the above analysis. Firstly we consider how the information which our oracle provides may (or may not) be obtained in real life. Then we consider, assuming the oracle is available, how one can obtain the required number of oracle queries.

4.1 Is the oracle practical?

Upon performing the RSA decryption function the card needs to execute three validity checking steps as explained earlier. The EMV standard [4] mentions these in the order 1, 2, 3 as specified earlier, however no warnings are given that executing them in a different order will make a difference to the security. A PIN encryption passes only if *all* tests pass.

In [8] the tests are given in a different order, namely 2, 1, 3. Indeed it might be tempting to implement the padding checking algorithm in this order with Test 2 before the other operations since then one is dealing with the leading byte first.

In performing three tests there are two basic ways this can be done. The first method tests the three conditions in sequence and aborts on the first occurrence of a failure, we call this the sequential method. In the second method, one tests all three conditions and then returns failure at the end, we call this the parallel method. The parallel method is usually considered "best practice" since it avoids any timing analysis which could result from the sequential method. However, any padding test is susceptible to a possible simple power analysis style attack in which the attacker observes whether the test passes or fails.

In the following table we give for each method of testing, and each order of the tests whether the timing analysis or simple power analysis *could* result in an attack against PIN encryption. We make no claim as to whether such an attack could be carried out in practice, only that it is possible if such an implementation choice is made. In all cases we ignore the trivial attack which requires 10000 ciphertext queries.

Order	Method	Timing	SPA
1, 2, 3	Sequential	-	-
1, 3, 2	Sequential	-	-
2, 1, 3	Sequential	✔	✔
2, 3, 1	Sequential	✔	✔
3, 2, 1	Sequential	-	-
3, 1, 2	Sequential	-	-
1, 2, 3	Parallel	-	✔
1, 3, 2	Parallel	-	✔
2, 1, 3	Parallel	-	✔
2, 3, 1	Parallel	-	✔
3, 2, 1	Parallel	-	✔
3, 1, 2	Parallel	-	✔

We note that if Test 2 is not performed first in the sequential method then it is highly unlikely for the attacker to be able to apply a Bleichenbacher/Manger style attack above, since the card is highly likely to abort after performing Test 1. This means that is would be better practice to implement the sequential test in the order 1, 2, 3, rather than the parallel test. However, it is clear from our analysis that any card which executes Test 2, for every ciphertext passed to it, is susceptible.

4.2 Can one obtain this many queries?

So from now on we assume that an attacker can obtain access to an oracle which tells him whether Test 2 passes or fails for a particular ciphertext. Whilst 30 queries may seem small compared to the 10000 needed to brute force the PIN, or the few thousand needed in the case of Bleichenbacher and Manger on more elaborate padding schemes, the card architecture does protect against executing even 30 queries.

The card usually has two counters within it; a PIN try counter n and a PIN decipherment error counter m. The card which will lock as soon as n invalid PINs have been tested (or m decryption errors have occured) in a row. According to sources within the industry, the value of m is usually quite large and is independent of n, i.e. decrementing m does not decrement n, hence the PIN try counter only applies upon a valid decryption. The value of n is not stated in the standard but in the field it is generally set to be equal to three, whilst the use of a value for m is not alluded to at all within the standard.

In the following we assume the most pessimistic situation for the attacker, in that we assume that a PIN decipherment error is treated as a PIN try error and that the value of the PIN try counter is set to three. This means an attacker can execute at most two invalid ciphertext queries for every valid PIN number entered by the user. It is not unreasonable, given how most people use their cards, to assume that this can be increased to four, since users are used to things "going wrong" (for example by typing their PIN in incorrectly) hence at a electronic point of sale terminal they would not treat as suspicious the request for a second PIN entry request.

Hence, obtaining 30 such ciphertext queries for attacking a particular user would only be possible if one was able to find a user who repeatedly used his card at the attackers terminal. Such an attack might be economic for a high net-worth individual, but would still require around 7-8 such visits to obtain the PIN with a 75% chance of success.

Alternatively, a high through-put supermarket could easily blame a software glitch for requiring all customers in one day to enter their PINs three times. This would give a total of 6 such queries per card. With enough customers passing through the door one would expect at least one PIN to be recovered within a few hours of trading.

5 Conclusion

We have presented a variant of Bleichenbacher and Manger's method to attack PIN encryption via RSA. The attack can be considered highly theoretical, since implementing it in practice would require a lot of work and would require the card to provide a validity checking oracle for Test 2. What is interesting however is the small number of oracle queries which result in a full PIN recovery.

In some sense theoretically our method could be improved. Information theoretically we are only trying to recover 13.28 bits of information, since we are

trying to recover a 10000 bit PIN number. Each oracle query returns one bit of information, i.e. does a certain multiple of the message lie in a given interval? It would be interesting whether the attack could be improved, for example by a better choice of parameters or by using a different oracle, so as to reduce the number of oracle queries even further.

However, finally we note that PIN encryption from the keypad to the card is not implemented in many geographic locations. For example it was decided in the UK chip-and-pin system to not implement PIN encryption so as to enable cheaper cards, but we note that this comes at an expense in security as the attacks on unencrypted PINs in [3] point out.

References

1. M. Bellare and P. Rogaway. Optimal asymmetric encryption. In *Advances in Cryptology - EUROCRYPT 1994*, Springer-Verlag LNCS 950, 92–111, 1995.
2. D. Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In *Advances in Cryptology – CRYPTO '98*, Springer-Verlag LNCS 1462, 1–12, 1998.
3. S. Drimer, S.J. Murdoch and R. Anderson. Thinking inside the box: system-level failures of tamper proofing. *IEEE Symposium on Security and Privacy*, 281–295, 2008.
4. EMV. *Integrated circuit card specifications for payment systems, Book 2. Security and Key Management.* Version 4.2, June 2008. Available from `www.emvco.com`.
5. EMV. *Integrated circuit card specifications for payment systems, Book 3. Application Specification.* Version 4.2, June 2008. Available from `www.emvco.com`.
6. ISO 9564-2. *Banking – Personal Identification Number management and security – Part 2: Approved algorithm(s) for PIN encipherment.* 2005. Available from `www.iso.org`.
7. J. Manger. A chosen ciphertext attack on RSA Optimal Asymmetric Encryption Padding (OAEP) as standardized in PKCS # 1 v2.0. In *Advances in Cryptology – CRYPTO 2001*, Springer-Verlag LNCS 2139, 230–238, 2001.
8. C. Radu. *Implementing electronic card payment systems.* Artech House Publishers, 2002.