Attacking LCCC Batch Verification of RSA Signatures

Martin Stanek*

Department of Computer Science
Faculty of Mathematics, Physics and Informatics
Comenius University
Mlynská dolina, 842 48 Bratislava, Slovak Republic
stanek@dcs.fmph.uniba.sk

Abstract. Batch verification of digital signatures is used to improve the computational complexity when large number of digital signatures must be verified. Lee at al. [2] proposed a new method to identify bad signatures in batches efficiently. We show that the method is flawed.

Key words: batch verification, digital signatures, RSA

1 Introduction

Batch verification of digital signatures provides better computational complexity when several signatures are verified together. Several batch verification algorithms have been proposed for various digital signature schemes, e.g. DSA or RSA.

Having n message/signature pairs $(m_1, s_1), \ldots, (m_n, s_n)$ the batch verification algorithm answers following question: "Are all the signatures correct (valid)?" In the negative case, further investigation (and computation) is necessary in order to identify bad signature or signatures. Lee at al. [2] proposed a method to identify bad RSA-type signatures in batches efficiently (we will refer their method as LCCC). However, the LCCC method is flawed. We show an explicit attack on the LCCC method for identifying single bad signature. In addition, we show that using this attack the LCCC method for identifying multiple bad signatures offers none or only marginal computational savings over straightforward divide-and-conquer approach. In the following section we describe the LCCC method, and present our attacks.

2 LCCC method and its security problems

Let N be a public RSA modulus, i.e. $N = pq$ for sufficiently large primes p, and q. Let e be a public exponent for this RSA instance, i.e. e is relatively prime to

* supported by VEGA 1/3106/06

$(p-1)(q-1)$. Signature s of a message m is valid if and only if $s^e \equiv m \pmod{N}$. In order to simplify notation, we use m instead of $H(m)$, i.e. m denotes a hash of actual message.

The LCCC method uses a standard "generic test" to test the validity of a batch $(m_1, s_1), \ldots, (m_n, s_n)$. The generic test (GT) can be instantiated as a Random Subset Test or Small Exponents Test, see [1]. The GT can be viewed as a probabilistic algorithm with security parameter l:

GT(x)
input: $x = ((m_1, s_1), \ldots, (m_n, s_n))$

---

return "true" whenever all signatures are valid
return "false" whenever x contains at least one bad
  signature (in this case with probability of mistake 2^{-l})

Since the choice of GT does not affect our analysis, we do not describe it in greater detail.

Lee at al. [2] proposed method for identifying single bad signature, and its generalization to multiple bad signatures.

2.1 Single bad signature

The algorithm called $\mathrm{DBI}_{\mathrm{basic}}$ is aimed at identifying single bad signatures in batch (if such signature exists), see Figure 1.

$\mathrm{DBI}_{\mathrm{basic}}(x)$
input: $x = ((m_1, s_1), \ldots, (m_n, s_n))$

---

if GT(x) **then return** "true";
$M \leftarrow \prod_{i=1}^{n} m_i$; $M^* \leftarrow \prod_{i=1}^{n} m_i^i$;
$S \leftarrow \prod_{i=1}^{n} s_i$; $S^* \leftarrow \prod_{i=1}^{n} s_i^i$;
find $k \in \{1, \ldots, n\}$ such that $\left(\frac{S^e}{M}\right)^k \equiv \frac{(S^*)^e}{M^*} \pmod{N}$; $(*)$
if k does not exist **then return** "false";
if GT($x \setminus (m_k, s_k)$) **then return** k (index of bad signature);
return return "false";

Fig. 1. $\mathrm{DBI}_{\mathrm{basic}}(x)$

The authors claim following properties of $\mathrm{DBI}_{\mathrm{basic}}$:

1. If all signatures in the batch x are valid, $\mathrm{DBI}_{\mathrm{basic}}(x)$ returns "true".
2. If there is exactly one bad signature in the batch x, $\mathrm{DBI}_{\mathrm{basic}}(x)$ returns the index of this bad message/signature pair.
3. If there are more then one bad signature in the batch x, $\mathrm{DBI}_{\mathrm{basic}}(x)$ returns "false" (This is the reason for the test $\mathrm{GT}(x \setminus (m_k, s_k))$).

The problem

We show that the property 2 can be easily attacked (and thus, Theorem 1 in [2] does not hold). Let $x = ((m_1, s_1), \ldots, (m_n, s_n))$ be a batch where all signatures are valid. Let j be an arbitrary even number from the set $\{1, \ldots, n\}$. Let us replace the pair (m_j, s_j) with pair $(m_j, -s_j)$. We denote this new batch as x'. Since the public exponent e is odd number, we get

$$(-s_j)^e \equiv -(s_j^e) \equiv -m_j \not\equiv m_j \pmod N.$$

Hence, batch x' contains exactly one bad signature.

When evaluating $\mathrm{DBI}_{\mathrm{basic}}(x')$, the property $(*)$ is satisfied for every even k from the set $\{1, \ldots, n\}$:

$$\text{left side:} \quad \left(\frac{S^e}{M}\right)^k \equiv \left(\frac{(-s_j)^e \cdot \prod_{i \in \{1,\ldots,n\}-\{j\}} s_i^e}{m_j \cdot \prod_{i \in \{1,\ldots,n\}-\{j\}} m_i}\right)^k \equiv$$

$$\left(\frac{(-s_j)^e}{m_j}\right)^k \equiv \left(\frac{s_j^e}{m_j}\right)^k \equiv 1 \pmod N$$

$$\text{right side:} \quad \frac{(S^*)^e}{M^*} \equiv \frac{((-s_j)^j)^e \cdot \prod_{i \in \{1,\ldots,n\}-\{j\}} (s_i^i)^e}{m_j^j \cdot \prod_{i \in \{1,\ldots,n\}-\{j\}} m_i^i}$$

$$\frac{((-s_j)^j)^e}{m_j^j} \equiv \frac{((s_j)^j)^e}{m_j^j} \equiv 1 \pmod N$$

When simplifying left and right sides of $(*)$ we make use of the fact that k and j are even numbers, respectively. Hence, the first tested even k (probably $k = 2$ when implemented in a standard for-loop) will be determined as an index of bad signature. If $k \neq j$, the subsequent test $\mathrm{GT}(x \setminus (m_k, s_k))$ returns "false". Let us summarize: the batch x' contains single bad signature, but $\mathrm{DBI}_{\mathrm{basic}}$ was unable to find it.

Remark 1. The $\mathrm{DBI}_{\mathrm{basic}}$ method cannot be easily fixed. Testing every candidate k satisfying $(*)$ will destroy intended efficiency of the method. Moreover, testing whether bad signature is just -1 multiple of the valid signature is computationally as demanding as simply checking the signature alone.

Remark 2. Modifying $(*)$ in such way that only odd exponents are used, i.e. $M^* \leftarrow \prod_{i=1}^n m_i^{2i-1}$, $S^* \leftarrow \prod_{i=1}^n s_i^{2i-1}$, and $(*)$ transforms into

$$\left(\frac{S^e}{M}\right)^{2k-1} \equiv \frac{(S^*)^e}{M^*} \pmod N,$$

would prevent our attack. However, the security of this modification should be investigated closer.

2.2 Multiple bad signatures

Lee at al. extended their $\mathrm{DBI}_{\mathrm{basic}}$ method to identify multiple bad signatures in a batch. The authors used divide-and-conquer approach and denoted their method DBI_α (see Figure 2). $\mathrm{DBI}_\alpha(x)$ returns the set of indices of bad signatures. For this reason, the value "true" can be viewed as an empty set in Fig. 2.

$\mathrm{DBI}_\alpha(x)$
input: $x = ((m_1, s_1), \ldots, (m_n, s_n))$

---

if $n = 1$ **then**
    if $\mathrm{GT}(x)$ **return** "true";
    else return $\{1\}$ (index of bad signature);
if $n = 2$ **then**
    if $\mathrm{GT}(x)$ **return** "true";
    else find $k \in \{1, 2\}$ such that $\left(\frac{(s_1 s_2)^e}{m_1 m_2}\right)^k \equiv \frac{(s_1 s_2^2)^e}{m_1 m_2^2} \pmod N$; $\quad (**)$
        if $k = 1$ **return** $\{1\}$;
        if $k = 2$ **return** $\{2\}$;
        else return $\{1, 2\}$;
/* case $n > 2$ */
if $\mathrm{GT}(x)$ **then return** "true";
$M \leftarrow \prod_{i=1}^n m_i$; $M^* \leftarrow \prod_{i=1}^n m_i^i$;
$S \leftarrow \prod_{i=1}^n s_i$; $S^* \leftarrow \prod_{i=1}^n s_i^i$;
find $k \in \{1, \ldots, n\}$ such that $\left(\frac{S^e}{M}\right)^k \equiv \frac{(S^*)^e}{M^*} \pmod N$; $\quad (*)$
if k exists **then**
    if $\mathrm{GT}(x \setminus (m_k, s_k))$ **then return** $\{k\}$;
/* k does not exist or $\mathrm{GT}(x \setminus (m_k, s_k)$ returns "false" */
divide x into α batch instances $(x_1, \ldots, x_\alpha)$ containing approx. $\frac{n}{\alpha}$ pairs each;
return $\mathrm{DBI}_\alpha(x_1) \cup \cdots \cup \mathrm{DBI}_\alpha(x_\alpha)$;

Fig. 2. $\mathrm{DBI}_\alpha(x)$

The attack described for $\mathrm{DBI}_{\mathrm{basic}}$ is not applicable to DBI_α. The reason is that for the case $n = 2$ there is single even number (thus $k = j$), and the correctness of the result from the test $(*)$ is checked again with GT.

However, the reason for DBI_α existence is its performance advantage over straightforward divide-and-conquer approach employing GT, see [3]. Lee at al. analyzed DBI_α, and implemented several experiments that support the claim of its superior performance.

The problem

The idea of our attack from Sect. 2.1 can be used to increase the DBI_α complexity. Let us illustrate this increase on case of single bad signature and $\alpha = 2$. Result can be easily generalized for multiple bad signatures and other values of α. Let $n = 2^m$, and let us assume that $(*)$ is tested in a standard for-loop.

The adversary modifies signature s_n to $-s_n$. Then the $(*)$ is satisfied for $k = 2$, but $\mathrm{GT}(x \searrow (m_2, s_2))$ returns "false", thus forcing division of x, and recursive calls of $\mathrm{DBI}_2(x_1)$ and $\mathrm{DBI}_2(x_2)$. $\mathrm{DBI}_2(x_1)$ requires one GT. However, $\mathrm{DBI}_2(x_2)$ requires two GT, since $(*)$ is satisfied for $k = 2^{m-1} + 2$, and leads to further recursive calls. Counting all GT's (regardless of their input size) performed during DBI_2 computation gives $3(m-1)$ invocations of GT. On the other hand, standard divide-and-conquer method requires only $2m + 1$ invocations of GT.

Remark 3. Using odd exponents, as proposed in Remark 2, would prevent this "complexity attack".

References

1. M. Bellare, J. A. Garay, T. Rabin: Fast Batch Verification for Modular Exponentiation and Digital Signatures, In *Advances in Cryptology – EUROCRYPT'98*, LNCS 1403, Springer-Verlag, pp. 236–250, 1998.
2. S. Lee, S. Cho, J. Choi, Y. Cho: Efficient Identification of Bad Signatures in RSA-Type Batch Signature, *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, Vol. E89-A, No. 1, pp. 74–80, 2006.
3. J. Pastuszak, D. Michalek, J. Pieprzyk, J. Seberry: Identification of Bad Signatures in Batches, In *Public Key Cryptography – PKC 2000*, LNCS 1751, Springer-Verlag, pp. 28–45, 2000.