Study of RSA Performance in Java Cards

G. Bernabé and N. Clarke

Centre for Security, Communications and Network Research,
Plymouth University, Plymouth, UK
e-mail: info@cscan.org

Abstract

In a near future smart card may have to use RSA 4096 bits on Java cards, so it may be interesting study the viability of RSA (both for private key and public key operations) in smart cards fopr such key lengths.
In this paper, some measurements of RSA performances carried out (using keys lengths from 512 to 2048 bits) for a recent Java card are presented and some regression analyzes as well as some statistical tests have been performed in order to calculate some estimations of performances for operations with RSA 4096 bits. This study shows that it may be realistic for smart cards' manufacturers to implement RSA 4096 bits in today's smart cards (notably Java cards), mainly for smart cards implementing TLS and certificate checking with public key operations.

Keywords

Smart cards, performance, RSA, Java Card, TLS

1 Introduction

Actually, different applications use RSA cryptosystem in smart cards: traditional applications such as banking (EMV protocol [EMV]), but also some Internet applications: particularly for Web authentication with TLS when the client is authenticated with his certificate and his private key (also called mutual authentication), in this case the smart card performs a RSA operation with its private key (e.g.: [Aussel J.-D., 2007], [PKCS11], [CAPI]). Some other applications (few for the moment) also implement the entire TLS Handshake protocol in the smart card for the client side (e.g.: [Urien P., 2009] and [Lu H.K., 2007]), therefore, some operations with the public keys of the certificate authorities (CA) are required so that the card can check the server's certificate. There are already some CA on the Internet that sign certificates with RSA 4096 bits, for a full list: see [List of CA, 2011], actually more than 10 CA using RSA 4096 are included in Web browsers, at least for *Firefox* and *Internet Explorer* (CA using RSA 4096 are mainly those of governments but also few others like *Microsoft Internet Authority*: used by services such as *Outlook* for emails).

However, it appears that no Java cards (name given to the smart cards implementing Java Card specifications [Java Card]) implement RSA up to 4096 bits today, at least it hasn't been possible to find the specifications of such a smart card on the Internet. In fact RSA 4096 is part of the version 3 of Java Card, but even some recent smart

cards implementing Java Card 3 don't implement RSA up to 4096 bits (e.g.: *G&D SmartCafe expert 6.0*, Java Card 3.0.1: [G&D SC 6.0, 2011]).

So when considering certificate checking in the smart card for some Internet applications, this can cause a problem of compatibility between the server and the smart card, when the smart card has to check the server's certificate.

The aim of this study is to assess the viability of RSA cryptosystem for smart cards, and to determine if it makes sense to implement RSA 4096 in today's Java cards, in particular in the context of TLS and certificate checking in the card. Some measurements of the times of signature/verification and encryption/decryption (using both private key and public key) have been carried out for the card G&D SmartCafe 3.2 (Java Card 2.2.1), then an attempt has been performed to correlate these measurements with the theoretical time complexity of RSA, in order to estimate the performances of RSA for longer keys (4096 bits).

RSA 2048 bits in smart cards is pretty fast, but when increasing the length of the key, the time of private keys operations grows quite quickly.

Furthermore RSA 4096 will be probably required in a near feature (if it is not replaced by other cryptosystems) for the other applications that already use RSA in smart cards.

In order to estimate the times of operations for RSA 4096 bits, a regression analysis is conducted, however the extrapolation to a key length of 4096 bits is possible only if the size of the registers of the RSA coprocessor is big enough to store such lengths of data, otherwise the estimations for RSA 4096 for this chip will be distorted. The chip of the smart card used for the tests is *NXP P5CD144*, and it contains a FameXE RSA coprocessor. According to the specifications of this chip: [NXP P5CD144, 2008], this FameXE coprocessor supports RSA with an operand length of up to 8-kbit (up to 4-kbit with intermediate storage in RAM only). So it is very probable that an extrapolation using a regression analysis is valid.

This study is organized as follows: a review of theory and time complexity of RSA is presented in **2.**, then some measurements carried out for a recent smart card equipped with Java Card are analyzed and correlated in **3.** and **4.**. Finally, some estimations of RSA operations with keys bigger than 2048 are calculated in **4.**, and the viability of RSA for smart cards is discussed in **5.**.

2 Background

In this part, some theoretical concepts on RSA are reminded before presenting the analysis of measurements.

2.1 RSA cryptosystem

Below is the definition of RSA theorem:

p and q are two secret prime numbers and $N=p \cdot q$;
e is an integer satisfying: $\gcd(e,(p-1)(q-1))=1$; and $1<e<(p-1)(q-1)$;
d is an integer satisfying: $ed \equiv 1 \pmod{(p-1)(q-1)}$; and $m \in \mathbb{Z}_N$;

encryption (using public key) **decryption** (using private key)

$$c \equiv m^e \pmod{N} \quad (1) \qquad m' \equiv c^d \equiv m^{ed} \pmod{N} \quad (2)$$

with m the plaintext and c the ciphertext then m' equals m

The proof of this theorem can be found in many books, this one by example: [Hoffstein J., Pipher J. and Silverman J.H., 2008], which is generally quite detailed for the proofs of theorems, with many exercises for undergraduate students.

A speed improvement of RSA operations with private key can be done using the Chinese remainder theorem (CRT), a method has been introduced by J.-J. Quisquater in 1982. As explained in [Smart N., 2002], using CRT, operations with private key can be also calculated as follows (equivalent to formula (2)):

$$\begin{cases} m_1 \equiv c^{d_1} \pmod{p} \\ m_2 \equiv c^{d_2} \pmod{q} \end{cases} \text{where}$$

$$\begin{cases} d_1 \equiv d \pmod{(p-1)} \\ d_2 \equiv d \pmod{(q-1)} \end{cases} \Rightarrow m = m_1 + ((m_2 - m_1)(p^{-1} \pmod{q}) \pmod{q})p$$

d_1, d_2 and $p^{-1}[q]$ (the inverse of p modulo q), are likely to be pre-computed and stored with the private key.

This is equivalent to calculate $m \equiv c^d \pmod{N}$.

Thanks tho the CRT, this method substitutes the main modular exponentiation for two sub modular exponentiations with smaller exponents, this permits to speed up the time of calculation by about 4 times (sizes of p and q are generally close).

(security considerations of RSA are not developed here, as explained before, the focus is on time complexity)

Time complexity of RSA

As we can see, the calculations done by RSA operations are some modular exponentiations, efficient methods for implementing this calculations have been introduced many years ago, "square-and-multiply" algorithm (also called binary exponentiation), combined with the method of Montgomery multiplication [Koç C. K., 1994].

Using these methods, the basic time complexity for RSA operations with CRT is about:

$$\frac{3k^3}{8} + \frac{k^2+3k}{2}, \quad \textbf{(3)}$$

k being the length (in bits) of the exponent. For RSA without CRT the time complexity is: $\frac{3k^3}{2}$.

According to some recent articles (e.g.: [Huang Z. and Li S., 2011]), today "square-and-multiply" combined with Montgomery multiplication are still the 2 algorithms the most used for RSA, including in smart cards. However many improvements of these 2 algorithms for speeding up the operations have been published this last decade (e.g.: [Alia G. and Martinelli E., 2002], [Sepahvandi S. *et al.*, 2009] or [Huang Z. and Li S., 2011]...), so it's not easy to know which ones exactly are implemented in a given smart card.

Moreover the history of smart cards is made of perpetual sophistication of algorithms in order to counter a variety of physical and logical attacks, this has begun in about 1996 after the first timing attacks of Paul Kocher, until now with more complex attacks combining physical and logical attacks (e.g.: [Sato H. *et al.*, 2005], [Amiel F. *et al.*, 2009], [Markantonakis K. *et al.*, 2009]).

There are also some alternatives to these two algorithms, by example this article: [Wu C.-L. *et al.*, 2006] reviews different other algorithms that are very efficient.

As a result, the basic algorithms have been enhanced, so the time complexity from **(3)** isn't the real one. However, as far as I know, when using square-and-multiply algorithm combined with Montgomery multiplication or some of their variants, the time complexity for private key operations stays in $O(k^3)$. So it is likely to be the complexity of the algorithm implemented in the smart card used for the tests of this study. According to the papers previously mentioned: because of the constraints of embedded devices like smart cards, square-and-multiply algorithm combined with Montgomery multiplication or some of their variants have been much more popular for smart cards until now. So the hypothesis is made that the time complexity of RSA operations with private key for the smart card used for these tests is of the order of $O(k^3)$.

Concerning operations with public keys, the time complexity is much more inferior, because a common practice is to use a small exponent e, generally equal to 65537 ([Hoffstein J., Pipher J. and Silverman J.H., 2008], page 121), this is possible because it is admitted that it doesn't weaken the security of RSA. Then, complexity with public key is in $O(k^2)$.

3 Measurements

3.1 Methodology of measurements

In order to measure the times of RSA operations, the method used for this study is sometimes used by researchers for measuring smart card performances. The measure

is performed on the host PC using the Java API of OCF [OCF, 2003]. First it consists in measuring the time for the card to execute the“empty loop”: the empty loop is the program containing the operation we want to measure but without the line of code corresponding to the operation we want measure. Secondly it consists in measuring the time for the card to execute the “full loop”: the full loop is the same program but containing the line of code corresponding to the operation we want to measure. Finally we can subtract the time of “empty loop” from the time of “full loop” to obtain the time of the operation (method used by [Cordry J., 2009] and [Rehoui K., 2005]).

For this study, all the other methods of the framework from [Cordry J., 2009] have not been used, because the measures were precise enough. However using this framework could have permitted to improve precision of measures, by modifying the bytecode to isolate again more the operation to measure and by using other statistical tests to refine the measures.

The tests have been carried out for the card *G&D SmartCafe 3.2* (Java Card 2.2.1). The smart card reader used is *Gemalto GemPC USB-SL*, which is USB full speed: 12Mbps

The measurements are performed with the following Java code, using the Java API of OCF [OCF, 2003]:

```
start=System.currentTimeMillis();
sendAPDU(cmd,true);
time=System.currentTimeMillis();
```

These measurements are probably relatively approximate because of the work environment and the possible noise, furthermore due to the high level API of Java Card language, the lowest operation that it's possible to measure for RSA is the one corresponding to this line of code (Java Card):

```
cipherRSA.doFinal(data,(short)dataOffset,byteRead,data,(s
hort)dataOffset);
```

So it's not easy to know which operations are added by the API in addition to the RSA operation, but this factor of imprecision has been limited to its maximum because the RSA algorithm used is ALG_RSA_NOPAD, this algorithm (part of the API of Java Card) doesn't add any random number or padding to the data, so it only performs the modular exponentiation of RSA.

The card never sends the data resulting of its operations, so the measure is not affected by the time for the PC to display the result (of course it has been verified beforehand that when displaying the response of the card, the encrypted/decrypted or signed/verified data is correct).

The probable noise is potentially problematic (tasks, processes of the OS...), but when running the minimal number of programs necessary to perform the measures (basically none except the shell to run Java), it has been noticed that time measurements are minimalistic. Furthermore the time of the "empty loop" is so small in comparison with the time of the "full loop" (in the case of private key operations), that the noise should be negligible. Also, in the case of RSA operations with private key, the difference of times of operation between two lengths of keys is so important that it would have been probably not very useful to get more precise measures. However concerning public key operations, the measured times are very small, and using the framework from [Cordry J., 2009] could have been probably useful for a better precision.

It has been decided not to make more than 20 measures for each operation as the standard deviation and variance of the distribution are very small proportionally to the order of the measurements.

One pair of RSA keys (private key CRT + public key) has been generated for each length of key (the card supports 9 lengths of key, 8 of them have been used). For each pair, 20 measures have been carried out for each operation with a given key. This makes a total of: 20 measures for the signature with the private key RSA CRT, 20 measures for the encryption with the public key, 20 measures for the verification with the private key RSA CRT and 20 measures for the decryption with the public key (this makes 20*4*8=640 measures), in addition there is a test in empty loop for each measure (making a total of 2*640=1280 measures). The data to process is always the size of the key (modulus).

3.2 Results

Here are the results of the measurements:

key size (bits) / stat	512	768	896	1024	1280	1536	1984	2048
signature (ms) private CRT	63.9	95	115.9	142.7	210.1	304.3	543.2	585
encryption (ms) public key	14.1	15.5	16	16.7	18.1	19.9	23.9	24.8

Table1: Performances of RSA for ALG_RSA_NOPAD in *G&D SmartCafe 3.2*, Java Card 2.2.1

For a given key length, performances with private key for signature and verification (with ALG_RSA_NOPAD) are almost identical so only the results for signature are mentioned, the same for encryption/decryption.

The curve of RSA signature grows quite quickly, the more the length of the key increases the more the growth of the curve is important.

When changing the scale of the time axis and representing also the encryption with the public key, we can see that the blue curve is really soaring by comparison with the red one that almost stagnates: the time of encryption with the public key only inched up from 14.1ms to 24.8ms whereas the length of the key increased from 512 to 2048 bits.

Considering that the complexity of operations with private key CRT is in $O(k^3)$, and operations with public key in $O(k^2)$, according to the measurements, for k=2048: the time of operations with public key is more than 20 times faster than with private key CRT. Then if we want to estimate the time of operations for RSA 4096 bits with public key for this smart card, we know that it will be at least more than 20 times faster than with private key CRT of the same length.

4 Analysis of results

4.1 Estimation of the model: first approach

So under the hypothesis that the time complexity of RSA CRT operations with private key is $O(k^3)$, then the equation of the blue curve is of the form: $t=ax^3 + bx^2 + cx + d$ with a, b, c and d constants, and t the time of the operation.

In order to find the constants using the measurements of Table1, the following system has been solved with *Mathematica* software (4 points have been chosen, 4 are required as there are 4 unknowns in the system):

$$\begin{cases} t_1=ax_1^3 + bx_1^2 + cx_1 + d \\ t_2=ax_2^3 + bx_2^2 + cx_2 + d \\ t_3=ax_3^3 + bx_3^2 + cx_3 + d \\ t_4=ax_4^3 + bx_4^2 + cx_4 + d \end{cases}$$

with $x_1 = 512; x_2 = 896; x_3 = 1280; x_4 = 1948;$

and $t_1; t_2; t_3; t_4$

the corresponding time values from Table1

The result is: $a = 4.50537\times10^{-8}$; b = 0.000021989; c = 0.0358071; d = 33.7555; **(4)**

Using this model just found, we can observe that the estimations for the other points are quite close to the values of measurements:

x / value	768	1024	1536	2048
t measured	95	142.7	304.3	585
t from model	94.6	141.6	303.9	586.3

An estimation for x=4096 is t=3645.41 ms.

4.2 Least squares polynomial curve fitting

Now, using the method of least squares in the case of a polynomial curve, we may approximate better the model. So we apply directly the method of least squares to fit a distribution: considering a set of points $M_i(x_i,y_i)$, and $f(x)=c_0+c_1x+\ldots+c_px^p$; where p is the degree of the polynomial curve fit, and c_k the researched coefficients with $\{k \in \mathbb{N}, k \in [0,p]\}$.

So it consists in minimising Δ where:

$$\Delta=\sum_{i=1}^{n}[y_i-f(x_i)]^2=\sum_{i}^{n}\left[y_i-\sum_{k=0}^{p}c_k\cdot x_i^k\right]^2$$

We set out, $\{\exists k \in \mathbb{N}, k \in [0,p]\}: \frac{\partial\Delta}{\partial c_k}(c_0,\ldots,c_k)=2\sum_{i=1}^{n}\left[y_i-f(x_i)\cdot -x_i^k\right]^2=0$

$\frac{\partial\Delta}{\partial c_k}(c_0,\ldots,c_k)$ is the partial derivative of Δ with respect to the variable c_k at the point $(c_0,\ldots,c_k)$.

This gives a system of p+1 equations: $\sum_{i=1}^{n}x_i^k\cdot f(x_i)=\sum_{i=1}^{n}y_i\cdot x_i^k$

We write $S_k=\sum_{i=1}^{n}x_i^k$ and $W_k=\sum_{i=1}^{n}y_i\cdot x_i^k$, then the matrix system can be written as follows:

$$\begin{matrix} n & S_1 & S_2 & \cdots & S_p \\ S_1 & S_2 & S_3 & \cdots & S_{p+1} \\ S_2 & S_3 & S_4 & \cdots & S_{p+2} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ S_{p-1} & S_p & S_{p+1} & \cdots & S_{2p-1} \\ S_p & S_{p+1} & S_{p+2} & \cdots & S_{2p} \end{matrix} \cdot \begin{matrix} c_0 \\ c_1 \\ c_2 \\ \cdots \\ c_{p-1} \\ c_p \end{matrix} = \begin{matrix} W_0 \\ W_1 \\ W_2 \\ \cdots \\ W_{p-1} \\ W_p \end{matrix}$$

The proof that the result is minimalistic can be performed by calculating the Taylor series of Δ.

Here, we take p=3. Using the function "NonlinearModelFit" of *Mathematica* software we can solve this system. The model found is:

$$f(x)=4.39713\times10^{-8}x^3+0.0000247368x^2+0.0341446x+34.0995 \quad \textbf{(5)}$$

4.3 Estimations

Using *Mathematica*, the statistical ANOVA table generated for the model is:

	d.f.	sum of squares	mean squares
function 4	820935.	205234.	
error	4	1.142	0.285501
sum	8	820936.	
corrected sum	7	290435.	

The estimated error variance is 0.285501.

The mean predicted values are in the table below with the corresponding confidence intervals for a probability of 95%:

Observed	Predicted	Standard Error	Confidence Interval
63.9	63.9679	0.518619	{62.528,65.4078}
95	94.8313	0.311777	{93.9657,95.6969}
115.9	116.182	0.315459	{115.306,117.058}
142.7	142.216	0.299743	{141.384,143.048}
210.1	210.548	0.314431	{209.675,211.421}
304.3	304.254	0.434188	{303.048,305.459}
543.2	542.609	0.346651	{541.646,543.571}
585	585.492	0.422988	{584.318,586.667}

Then, an estimation of t for x=4096 is: f(x)=3610.66.

5. Review

So an estimated value for private key's operations with RSA CRT 4096 bits is about 3.6 seconds (for the smart card used for these tests), and for k=3072, the estimation of t is about 1.6 second.

(to be noted that these are just some estimations, under the hypothesis set in the previous parts, the most probable source of imprecision of the estimations is due to the high-level API of Java Card, used for RSA operations. So even if the time

complexity of RSA inside the coprocessor is $O(k^3)$, it's not easy to know exactly which operations are added by the Java Card API in addition to RSA)

In the context of TLS in the smart card for the client side (at least for the entire Handshake), the only time constraint is that the Handshake mustn't be longer than 10 seconds because it is the timeout set on most of the TLS servers, however for convenience reasons, much less is suitable.

For TLS without client authentication with certificate (optional), the client just performs some public key operations, so as explained previously: RSA 4096 bits operations with public key should be at least 20 times faster than with private key CRT. So based on the previous estimation, we can conclude that RSA is totally viable (as 3610÷20 < 200 ms) for the future of TLS in the smart card when the client is not authenticated by certificate. RSA 4096 bits having a key space of more than 128 bits of security, it is then considered secure for more than 30 years, according to the report on algorithms and key lengths ECRYPT II [ECRYPT II, 2011].

Also some other applications implement the server side of TLS protocol in smart card (according to the specifications of Java Card 3), in this case the card must perform an operation with its private key, so for cypher suites with RSA, private key's operations for RSA 4096 will be required in a near future.

We can suppose that in the future, with hardware improvements, the performances will be better, as RSA 2048 bits will be still secure for more than 10 years (according to [ECRYPT II, 2011]) and according to some reports [Briggs J.S. and Beresford R.A., 2001], smart cards seem to follow a kind of Moore's law, but not the same as computers, the period to double hardware performances is of the order of 5 years, rather than 18 months (for traditional computers).

However “some” physicians [Kaku M., 2011] point out that Moore's law should reach its limit by 2020, so cryptography adapted to smart cards will be likely an interesting challenge in the future for cryptographers and computer scientists. Also, the fact that the power of computers grows more quickly than the one of smart cards, presents another challenge for the future of smart cards because the key space of cryptographic algorithms grows in function of the performances of computers.

5 Conclusion

So this study shows that it can make sense for smart card manufacturers to implement RSA up to 4096 bits in today's smart cards, mainly for Java cards implementing TLS and certificate checking (given that some Ca on the Internet use now RSA 4096).

Also, the same study should be performed to estimate the time for the card to generate the 4096 bits key, because the time to generate the keys is noticeably much more longer than the times of signature/verification and encryption/decryption.

A credible alternative to RSA is elliptic curve cryptography (ECC), which provides better performances than RSA for private key operations, however in the case of public key operations, RSA is more efficient than ECC (according to different papers, but it would be interesting to do the test on recent Java cards).

Concerning TLS, some Web browsers actually support TLS 1.2 with cypher suites using ECC, servers as well should be compliant soon. From the server perspective, ECC is more suitable because RSA is more time consuming for calculations with private key, then a distributed deny of service (DDOS) is more efficient on a server implementing RSA than on a server implementing only ECC.

6 References

Alia G. and Martinelli E., 2002 G. Alia and E. Martinelli, (2002), "Fast modular exponentiation of large numbers with large exponents", *Journal of Systems Architecture: the EUROMICRO*, volume 47, issue 14-15, page 1079-1088, http://dx.doi.org/10.1016/S1383-7621(02)00058-9

Amiel F. *et al.*, 2009 Frederic Amiel, Benoit Feix, Michael Tunstall, Claire Whelan, and William P. Marnane, (2009), "Distinguishing Multiplications from Squaring Operations", *Lecture Notes In Computer Science*, Springer Berlin, volume 5381, page 346-360, http://dx.doi.org/10.1007/978-3-642-04159-4_22

Aussel J.-D., 2007 Jean-Daniel Aussel, (2007), "Smart Cards and Digital Identity", *Telektronikk 3/4 2007*, ISSN 0085-7130, http://www.telektronikk.com/volumes/pdf/3_4.2007/Page_066-078.pdf

Briggs J.S. and Beresford R.A., 2001 J.S. Briggs and R.A. Beresford, (2001), "Smart cards in health", *Report for the Department of Health*, University of Portsmouth

Cordry J., 2009 Julien Cordry, (2009), "La mesure de performance dans les cartes à puce", French PhD, Conservatoire National des Arts et Métiers (CNAM)

CAPI Website, Microsoft cryptographic API, http://msdn.microsoft.com/en-us/library/aa380255(v=vs.85).aspx

David J.P. *et al.*, 2007 J.P. David, K. Kalach and N. Tittley, (2007), "Hardware Complexity of Modular Multiplication and Exponentiation", *IEEE Transactions on Computers*, 56(10), page 1308-1319, http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4302704

ECRYPT II, 2011 (2011), "ECRYPT II Yearly Report on Algorithms and Keysizes", European Network of Excellence for Cryptology II, editor: Nigel Smart, available online: http://www.ecrypt.eu.org/

EMV, 2011 Website, EMV specifications, (2011), http://www.emvco.com/

G&D SC 6.0, 2011 Website, specifications of G&D SmartCafe Expert 6.0 (Java Card 3.0.1), http://www.gi-de.com/gd_media/media/en/documents/brochures/mobile_security_2/nb/SmartCafe-Expert.pdf

Hoffstein J., Pipher J. and Silverman J.H., 2008 Jeffrey Hoffstein, Jill Pipher and J.H. Silverman, (2008), "An Introduction to Mathematical Cryptography", Springer, ISBN 9780387779935

Huang Z. and Li S., 2011 Zhen Huang, Shuguo Li, (2011), “Design and Implementation of a Low Power RSA Processor for Smartcard”, *IJMECS*, vol.3, no.3, page 8-14

Java Card Website, Java Card specifications, (2011), http://www.oracle.com/technetwork/java/javacard/overview/index.html

Kaku M., 2011 Michio Kaku, (2011), “Physics of the Future: How Science Will Shape Human Destiny And Our Daily Lives by the Year 2100”, Doubleday, ISBN 9780385530804

Koç C.K., 1994 Cetin Kaya Koç, (1994), “High-Speed RSA Implementation”, Technical Report TR-201, version 2.0, RSA Laboratories

Koç C.K. *et al.*, 1996 Cetin Kaya Koç, T. Acar and B.S. Kaliski Jr., (1996), “Analyzing and comparing Montgomery multiplication algorithms”, *IEEE Micro*, 16(3), page 26-33

List of CA, 2011 Website, list of certificate authorities on the Internet used by Windows root certificate program, http://social.technet.microsoft.com/wiki/contents/articles/2592.aspx

Lu H.K., 2007 H. Karen Lu, (2007), “Network smart card review and analysis”, *Computer Networks*, volume 51 (Elsevier North-Holland), page 2234–224

Markantonakis K. *et al.*, 2009 Konstantinos Markantonakis, Michael Tunstall, Gerhard Hancke, Ioannis Askoxylakis and Keith Mayes, (2009), “Attacking smart card systems: Theory and practice”, *Information Security Technical Report*, volume 14, page 46-56, http://www.sciencedirect.com/science/article/pii/S136341270900017X

Menezes A.J., Van Oorschot P.C. and Vanstone S.A., 1996 Alfred J. Menezes, Paul C. Van Oorschot and Scott A. Vanstone, (1997), “Handbook of Applied Cryptography”, CRC Press, ISBN 0849385237

NXP P5CD144, 2008 Website, specifications of NXP P5CD144 with coprocesseur FameXE, (2008), http://www.nxp.com/documents/data_sheet/P5CX012_02X_40_73_80_144_FAM_SDS.pdf

OCF, 2003 OpenCard Framework, (2003), http://opencard.sourceforge.net/

PKCS#11 Website, PKCS#11 specifications, http://www.rsa.com/rsalabs/node.asp?id=2133

Rehoui K., 2005 K. Rehioui, (2005), “Java Card Performance Test Framework”, Master thesis, Institute Eurecom and Université de Nice Sophia Antipolis, France

Sato H. *et al.*, 2005 H. Sato, D. Schepers and T. Takagi, (2005), “Exact Analysis of Montgomery Multiplication”, *Progress in Cryptology - INDOCRYPT 2004, Lecture Notes in Computer Science*, publisher: Springer Berlin, ISBN 9783540241300, volume 3348, page 1387-1394, http://dx.doi.org/10.1007/978-3-540-30556-9_23

Sepahvandi S. *et al.*, 2009 S. Sepahvandi, M. Hosseinzadeh, K. Navi and A. Jalali, (2009), “An Improved Exponentiation Algorithm for RSA Cryptosystem”, *International Conference on Research Challenges in Computer Science, ICRCCS '09*, page 128-132, http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5401230&isnumber=5401177

Smart N., 2002 Nigel Smart, (2002),“Cryptography: An Introduction, 3rd Edition”, publisher: Mc Graw Hill Higher Education, ISBN 9780077099879, available online: http://www.cs.bris.ac.uk/~nigel/Crypto_Book/

Sun H.-M. *et al.*, 2009 Hung-Min Sun, Mu-En Wu, M. Jason Hinek, Cheng-Ta Yang and Vincent S. Tseng, (2009), “Trading decryption for speeding encryption in Rebalanced-RSA”,

Journal of Systems and Software, volume 82, Issue 9, page 1503-1512, http://www.sciencedirect.com/science/article/pii/S0164121209000910

Tews H. and Jacobs B., 2009 Hendrik Tews and Bart Jacobs, (2009), "Performance Issues of Selective Disclosure and Blinded Issuing Protocols on Java Card", *Proceedings of the 3rd IFIP WG 11.2: International Workshop on Information Security Theory and Practice*, Smart Devices, Pervasive Systems, and Ubiquitous Networks, Springer Berlin, Heidelberg, http://dx.doi.org/10.1007/978-3-642-03944-7_8

Urien P., 2009 Pascal Urien, "TLS-tandem: a smart card for web applications", (2009), *Proceedings of the 6th IEEE Conference on Consumer Communications and Networking Conference*, page 3-4

Wu C.-L. *et al.*, 2006 Chia-Long Wu, Der-Chyuan Lou and Te-Jen Chang, (2006), "Computational complexity analyses of modular arithmetic for RSA cryptosystem", *23rd Workshop on Combinatorial Mathematics and Computation Theory*, page 215-224