On Using Fast Exponentiation Algorithm in PDAs (or: How Secure is the Discrete Logarithm Problem Assumption in PDAs?) Extended Abstract

Willy Susilo, Jianyong Huang and Jennifer Seberry
Centre for Computer Security Research
School of Information Technology and Computer Science
University of Wollongong
Wollongong 2522, AUSTRALIA
Email: {wsusilo, jyh33, jennie}@uow.edu.au

Abstract

Personal Digital Assistants (PDAs) are the miniature of normal size PCs, with a very limited computational power. In this paper, we investigate the security of PDAs when they are used to perform some cryptographic applications. In our context, we investigate the computation $y = g^x \pmod p$*, for a prime* p*, which is believed to be secure in the sense of the Discrete Logarithm Problem (DLP) assumption. To be more precise, knowing only* p*,* g *and* y*, it is hard to derive* x*. We note that this computation is the most important operation in most cryptographic algorithms. However, due to the limited computational power of PDAs, such computation requires some amount of time (and battery life). We show that by observing one of these parameters, we can reduce the hard problem of DLP to be predictable, and hence it is not secure. We also show how to securely generate these kind of computations with PDAs by employing some different techniques, so that they will not reveal any additional information to a passive eavesdropper. In contrast to previous works, we do not assume that the attacker can take the full control of the PDA. This assumption is only applicable to a smart card whenever it is used in a malicious smart card reader.*

Keywords: Network Security, Mobile Computing, Personal Digital Assistant, Fast Exponentiation

1 Introduction

Personal Digital Assistants (PDAs) have become an important part of our life. Currently, PDAs can perform the same operations as normal personal computers (PC), but they are more desirable as their small size means they fit in a pocket [24]. The recent trend is for users to use PDAs instead of a laptop. Thus there is a real need to make PDAs more useful for electronic commerce applications.

We note that there were several previous works that attempted to connect PDAs to PCs to assist them to perform some "heavy" computations, for example [6]. However, due to the recent development of PDAs, for example the new Intel XScale processor with 400 MHz, it is possible for PDAs to perform such computations by themselves. The problem with connecting PDAs to normal PCs is the fact that the PCs are not *trusted* [6], and therefore, some secure *challenge-response* protocols are required to enable the PDA to perform such computations. This will of course generate some overheads, in terms of computations and communications generated by the protocols. Moreover, due to their mobility, it is desirable for PDAs to perform any computation by themselves without involving any additional PC.

We only consider the Pocket PC environment (also known as Windows CE) in this paper. However, an extension to another operating system such as Linux (used in Yopi [28] and Sharp's Zaurus [23]) is straightforward.

In this paper, we are interested in using PDAs with a basic cryptographic calculation, i.e. the use of the *square-and-multiply* algorithm to find y from $y = g^x \pmod p$. For a complete survey on Fast Exponentation algorithms, we refer the reader to [11]. We are interested to know whether there is any relation between the time (or the battery life) taken to perform the computation and the secret

value (namely x), which is $x = log_g(y)$. The result is surprisingly predictable and it shows that using the square-and-multiply algorithm on a PDA is not secure. This basic cryptographic computation is the most important operation required in most cryptographic algorithms (for example, Diffie-Hellman key exchange algorithm [9], ElGamal encryption algorithm [10], etc.).

The rest of this paper is organized as follows. In the next section, we briefly describe some related background information on PDAs and the Discrete Logarithm Problem (DLP) related to our purpose. In section 3, we show the results of our experiment and point out the insecurity of the square-and-multiply operation in a PDA. In section 4, we discuss several other techniques that could be used to achieve secure computation on a PDA. Section 5 concludes the paper. In the Appendix, we list the results of our experiment.

1.1 Notations

The ring of integers modulo an integer p is denoted by Z_p, and its multiplicative group, which contains only the integers relatively prime to p, by Z_p^*. $|x|_2$ denotes the size of x in its binary representation. $\lfloor x \rfloor$ denotes the largest integer that is $\leq x$.

1.2 Our Contributions

In this paper, we show that applying the square-and-multiply algorithm to compute $g^x \pmod p$ will reveal some information to a passive attacker who can measure the time taken to perform such computation. We also show that battery consumption could reveal similar information. We argue that this measurement can be done efficiently when the device uses wireless connections, such as Bluetooth or Wireless LAN, and the connection has not been properly configured. We also show a way to securely compute this value on a PDA by using techniques other than square-and-multiply.

2 Related Information and Background

In this section, we briefly give an overview of a PDA, and in particular a Pocket PC device. We also describe the primitive cryptographic calculation that we are interested in exploring, together with some previous works.

2.1 Pocket PC PDA

Pocket PC is a new operating system, which is not a port from Windows NT or Windows 9x [19]. The Windows CE (or Pocket PC) APIs are modeled after those of Windows NT, but internally Windows CE is a new code base. The Pocket PC operating system is the operating system that is used in a Pocket PC PDA. The current release is Pocket PC 2002, named after its predecessor, Pocket PC 2000 (also known as Pocket PC 3).

There are four main types of Pocket PC hardware, namely Palm-size Pocket PC (Ps/PC), Pocket-size Pocket PC (P/PC), Handheld PC (H/PC) and CEPC (a standard desktop PC running Windows CE). Starting from Pocket-size Pocket PC version 1.1 (equipped with Windows CE 2.11), a standard Windows CE device supports the ADO, MFC, ATL and eVB code [19]. The standard processor used in the current Pocket PC device is a Strong ARM which runs at 206 MHz. The latest processor is an Intel XScale processor which runs at 400 MHz, but unfortunately the current Pocket PC 2002 operating system has not been optimized to use this ability.

A Pocket PC device can be synchronized with a desktop PC through synchronization software called "ActiveSync". The synchronization can be done through either a serial port, a USB port, an infrared port, a Bluetooth "virtual" port or a network (Ethernet connection) and Remote Access Service (RAS) server. ActiveSync serves four major purposes: data synchronization, file management, file backup and software installation.

2.2 Discrete Logarithm Problem (DLP) Assumptions and the Square-and-Multiply Algorithm

One of the most important cryptographic assumptions used in the construction of several cryptosystems is the Discrete Logarithm Problem (DLP) Assumption, which is defined as follows [25].

Problem Instance. $I = (p, \alpha, \beta)$, where p is prime, $\alpha \in Z_p$ is a primitive element and $\beta \in Z_p^*$.

Objective. Find the unique integer a, $0 \leq a \leq p-2$, such that $\alpha^a \equiv \beta \pmod p$. In other words, $a \equiv log_\alpha(\beta)$.

It is well-known that this problem is considered to be intractable [25]. The DLP assumption has been used to create several cryptosystems, including the ElGamal [10] cryptosystem. This intractability assumption has also been used to create signature schemes, such as fail-stop signatures [27] and undeniable signatures [7]. It is also used to create a server-aided computation [3], such as the one proposed in [6]. Maurer and Wolf [17] proved that the Diffie-Hellman problem and the DLP are polynomial-time equivalent in a cyclic group G of order $|G| = \prod p_i^{e_i}$, where all the multiple prime factors of $|G|$ are polynomial in $log|G|$.

It is predicted in [16] that the size of the Discrete Logarithm field that is used in DLP must be at least 1881 bits, to make it secure until the year 2020. It is also noted that the size of subgroup DLP must be at least 151 bits to make it secure for the same duration.

The common way to compute $y = g^x \pmod p$ is by using a technique that is known as "square-and-multiply" [25]. This method reduces the number of modular multiplications required to compute $g^x \pmod p$ to at most 2ℓ, where $\ell = |x|_2$. The method is illustrated as follows.

Algorithm 2.1: SQUAREANDMULTIPLY(g, x, p)

comment: Compute $g^x \pmod p$

$z \leftarrow 1$
for $i \leftarrow \ell - 1$ **downto** 0
do
$z \leftarrow z^2 \pmod p$
if $x_i \stackrel{?}{=} 1$
then $z \leftarrow z * g \pmod p$

In the above algorithm, the exponent x is represented in binary notation as follows

$$x = \sum_{i=1}^{\ell} x_i 2^i$$

where $x_i = 0$ or 1, for $0 \leq i \leq \ell - 1$.

We note that in Algorithm 2.1, the number of squarings is $|x|_2$. The number of modular multiplications is equal to the number of 1s in the binary representation of x. This algorithm is well-known and is the most common algorithm used to compute $g^x \pmod p$. For the complete history of this algorithm, we refer the reader to [14].

It is noted in [8] that the time required for the multiplication part of the algorithm is constant, independent of the factors, except that if the intermediary result of the multiplication is greater than the modulus, then an additional *reduction* has to be performed at the end of the multiplication. That means that for some factors, the multiplication time will be longer than for others. Knuth [14] gives a right-to-left version of the algorithm, which has the advantage of not needing to know ℓ ahead of time. Several other methods could be used to compute the above operation. For a complete survey, we refer the reader to [11] .

2.3 Related Works: Side-Channel Attacks

An important related work is the concept introduced by Kocher which is known as the "Timing Attack" [15]. He showed that an attacker may be able to find fixed Diffie-Hellman exponents, factor RSA keys, and break other cryptosystems, by only observing the amount of time required to perform private key operations [15]. In general, this is known as the "side-channel attack". Although the result presented in his paper is quite theoretical, Kocher gave the basic idea that it is possible to exploit some secret elements of the system by measuring time taken. The main problem with his approach is the fact that the attacker must have a very good knowledge of the implementation of the system he is attacking. The first practical attack was described at the rump session of Crypto '97 by Lenoir. Another practical implementation of a timing attack was described in [8], which was proposed with the implementation on a smart card. The error correction that could be used to improve the result of [8] was proposed in [12].

In [15], Kocher mentioned several countermeasures to prevent timing attacks. The straightforward solution is to make all operations take exactly the same amount of time. However, this method is not practical. A better solution is to use a blinding method [8]. Joye proposed several techniques to prevent timing attacks [13] in a smart card.

Kocher et al. introduced another kind of attack, known as power analysis attack [21, 22]. Power analysis attacks work by measuring the power consumption of a tamper-resistant device when it performs the cryptographic operations. Power analysis attacks fall into two basic categories. One is Simple Power Analysis (SPA), and the other is Differential Power Analysis (DPA).

We note that most of the work was introduced mainly to examine the security of symmetric key cryptographic algorithms. For example, the analysis of the Digital Encryption Standard (DES) and the Advanced Encryption Standard (AES) [20] were described in [22] and [5], respectively. The work on the power analysis attack on the public key algorithm was introduced in [26]. More specifically, Messerges et.al. [26] tried to attack the modular exponentiation in a smart card. This is closely related to our work, however [26] assumed that the attacker has the full control over the smart card when it is used. The attacker owns the malicious smart card reader to enable his malicious side channel attack when the smart card is used in the reader. We cannot make this assumption in this work since the PDA does not need to be connected to any other peripheral when it performs the cryptographic algorithm.

We would also like to highlight that the work on smart cards is completely different to the one that we are investigating. A PDA (or a handheld device, in a more general term) has more computational power compared to a smart card [4], and therefore, it is desirable to allow a PDA

to perform some cryptographic computations without the need of an untrusted/trusted PC.

3 Experiments with Square-and-Multiply Algorithm in PDAs

In this section, we describe our test bed experiment to compute $g^x \pmod p$ in a PDA. For the first experiment, we are interested in the time taken to complete such a computation on a PDA. We investigate the relation between the time taken to perform such a computation, since a PDA has a very limited computational power, and the secret exponent x. The result is surprising since we can estimate the secret exponent x that is used in the computation by only observing the time taken to perform the computation. We note that this result is not applicable when the computation is performed in a PC. We also extend our experiment to measure the battery life of a PDA, which also produces some similar results. The details of the experiment and the result are illustrated in Figure 2. The complete result of our experiment will appear in the final version of this paper.

3.1 Experiment Settings

We conducted an experiment with an iPaq Pocket PC H3630, which has a Strong Arm processor running at 206 MHz. The program, written in Embedded Visual C++, computes $y = g^x \pmod p$ and records the time taken to perform this computation. The computation is done using the square-and-multiply method given in Algorithm 2.1.

3.2 Experiment Results

As a result of the experiment, we captured the time elapsed for every possible x in Z_q. We include some results of the experiment in the Appendix for completeness. In the experiment, we found an interesting relation between the time elapsed and the number of squarings and multiplications that were performed. To make our explanation clearer, some sample data and results are listed in Figure 2 (Note: s is the total number of squarings, and m is the total number of multiplications).

From now on, let s denote the number of squarings (the third column in the table above), let m denote the number of multiplications (the fourth column), let t denote the time in seconds and let T denote the time in milliseconds ($t = \frac{T}{1000}$).

Based on our observations, we conclude that the following equation applies

$$(s + m - 1) * 1000 \approx T$$

To be more precise, we obtain

$$t = \lfloor s + m - \frac{1}{2} \rfloor \tag{1}$$

Hence, by observing the time required to perform the computation, we can estimate the number of squarings and multiplications that have been performed.

We note that $s = |x|_2$. We also observe that $m \leq s$. Applying this knowledge to equation (1), we can find the number of possible x that satisfy equation (1) (denoted by δ) by only observing t as follows.

$$\delta = \sum_{s=\lfloor \frac{t}{2}+1 \rfloor}^{\lfloor t+\frac{1}{2} \rfloor} \binom{s-1}{m-1} \tag{2}$$

Applying the knowledge from (1), we obtain

$$\delta = \sum_{s=\lfloor \frac{t}{2}+1 \rfloor}^{\lfloor t+\frac{1}{2} \rfloor} \binom{s-1}{\lfloor t+\frac{1}{2} \rfloor - s} \tag{3}$$

Theorem 3.1 *Given the time t (in seconds) taken to compute $g^x \pmod p$ in a PDA, there are δ possible solutions to x, where*

$$\delta = \sum_{s=\lfloor \frac{t}{2}+1 \rfloor}^{\lfloor t+\frac{1}{2} \rfloor} \binom{s-1}{\lfloor t+\frac{1}{2} \rfloor - s}$$

Proof: We know that $m \leq s$, and $s = |x|_2$. We require that $\lfloor s + m - \frac{1}{2} \rfloor = t$. From this equation, we obtain that $m = \lfloor t + \frac{1}{2} \rfloor - s + 1$. We know that for every single s, there will be $\binom{s-1}{m-1}$ possible numbers. Since the number of possible s's range from $\lfloor \frac{t}{2} + 1 \rfloor$ to $\lfloor t + \frac{1}{2} \rfloor$, then there are δ possible solutions, where

$$\delta = \sum_{s=\lfloor \frac{t}{2}+1 \rfloor}^{\lfloor t+\frac{1}{2} \rfloor} \binom{s-1}{m-1}$$

Incorporating the previous knowledge about m, then the equations are satisfied. □

Theorem 3.2 *Computing $y = g^x \pmod p$ in a PDA by using the square-and-multiply algorithm is not secure.*

Proof: If we can observe the time taken to perform this computation, say t, then by using theorem 3.1, we can find δ possible solutions to the equation. We note that by knowing p, g and y, together with the knowledge of δ possible solutions, one can find the unique x that has been used in the computation. This contradicts the DLP assumption and hence this computation is not secure. □

3.3 Extensions

We also conducted similar experiments by observing the battery consumption that was used to perform the computation $y = g^x \pmod p$ and the result is similar. We note that in practice, observing the battery life or the time can be done quite easily, especially if the PDA is connected to either an Intranet/Internet (via a Wireless LAN 802.11, for example) or a Personal Area Network (for example by using Bluetooth). This condition makes the computation vulnerable against a passive attacker who only listens to the communication.

4 An Alternative Way to Perform A Secure Computation on PDAs

We have shown that the use of an efficient square-and-multiply algorithm to compute $y = g^x \pmod p$ is insecure. In this section, we discuss several alternatives that could be used to perform such a computation on a PDA which would not reveal any additional information to the attacker. We start our discussion with a trivial scheme which includes a trusted PC. Then we extend our work to remove the need of a PC (either trusted or untrusted) and allow the PDA to perform such computation by itself.

4.1 Client-Server Model with a Trusted PC

The first solution that could be employed is by using a PC as a *partner* for a PDA. It means that the PC should conduct the computation and send the result to the PDA. This solution is somewhat trivial because the PC must be trusted with the knowledge of the secret key x. The simplest protocol is as follows.

PDA $\rightarrow$ PC: x, g, p
PC computes $y = g^x \pmod p$
PC $\rightarrow$ PDA: y

We note that the above protocol is subject to a man-in-the-middle attack. However, if the communication is performed by using a secure link, for example by using a USB/serial cable or a PDA cradle (as illustrated in Figure 1), then the attack is not applicable.

Figure 1. *Connection to a PC with a serial connection*

We note that this protocol is not applicable in practice. Normally, a PDA is carried everywhere the owner goes. With this condition, the protocol requires the PC partner of the PDA to be carried also, which is impractical.

We also note that if the PDA has set up a partnership with a PC (via the ActiveSync partnership, for example), then it is possible to share a long-term secret key on both sides. With this additional long-term secret key, the secret value x can be encrypted by the long-term secret key before it is sent to the PC, and the result should also be encrypted using the same key. This solution is better than the previous one, but the security of the scheme depends on the symmetric key algorithm that is used at both ends. Although a slight modification using a challenge-response protocol to obtain a fresh key is possible, it does not add much security.

As mentioned earlier, this method requires a trusted PC. With some modification, we can remove the need of a trusted PC and replace it with an untrusted PC. However, since the PDA can perform the operation by itself (c.f. a smart card), we try to find an alternative method which does not require a connection to a PC.

4.2 A Simple and Secure Scheme on a PDA: The First Scheme

In this section, we propose a technique that could be used to compute the above cryptographic operation. This technique is inspired by [1], together with the basic technique known as the *left-to-right 4-ary exponentiation* [18].

The idea of the method that we use in this section is as follows. To compute $g^x \pmod p$, we only need to know $g^i \pmod p$, where $i = 0, 1, \cdots, 15$. The method works as follows. Firstly, we convert x to its binary representation. Then we process each byte (4 bits) starting from its most significant byte. For each iteration, we use knowledge of $g^i \pmod p$ to find the number represented by each byte of x. Then we re-run the iteration until we have the final result.

Using the above idea, our first scheme is described as follows.

1. PDA computes $y_i = g^i \pmod p$, where $i = 1, 2, 3, \cdots, 15$.

x	x in Binary	s	m	Time (in msec)
7	111	3	3	5090
9	1001	4	2	4998
10	1010	4	2	5063
12	1100	4	2	5088
16	10000	5	1	5066

Figure 2. *Result of Our Experiment*

2. PDA computes $y = g^x \pmod p$ using the above knowledge as follows.
   (a) PDA converts x to its binary representation.
   (b) PDA uses the knowledge it gains from step 1 to do the following steps.
      i. Let i be the most significant byte of x (4 bits only). We note that $g^i \pmod p$ is obtained from the look-up table that is generated by PDA. Let $result = 1$.
      ii. If there are any bytes of x that have not been processed, then do the following.
         A. $result = result^{16} \pmod p$.
         B. $result = result \cdot g^i \pmod p$.
         C. $i \leftarrow$ the next byte (4 bits) of x.
         D. Goto (ii)
      iii. Return result.

We note that in the first step, the timing attack as we described earlier is still applicable. However, in the second step, the passive eavesdropper cannot launch the timing attack against the above computation.

Example 4.1 *Consider the following toy example. Suppose the PDA wants to compute* $g^{220} \pmod p$. *We note that* 220 *in binary is* 11011100. *Firstly, the PDA will obtain the knowledge of* $g^i \pmod p$ *where* $i = 1, \cdots, 15$. *Knowing that the first byte of 220 is* 1101 *(which is 13 in decimal), the PDA obtains* $g^{13} \pmod p$ *from its look-up table. Next, the PDA obtains the least significant bytes of* 220 *(i.e. 1100 or 12 in decimal), and looks up the value of* $g^{12} \pmod p$. *Finally, PDA "combines" the results by multiplying* $g^{12} \pmod p$ *and* $g^{208} \pmod p$ *to obtain* $g^{220} \pmod p$. *This operation can be done securely without any possible timing attack as we described earlier.*

Corollary 4.1 *A passive eavesdropper cannot learn the secret,* x, *used by the PDA by observing the time/battery that has been consumed.*

4.3 The Second Scheme

The technique that we use in this section is inspired by [2]. We note that this technique is similar to the idea of addition-subtraction chains described in [11]. We note that in practice, we will not use a subtraction chain (which means division) if the operation cannot be performed efficiently. However, in an operation on elliptic curves, the division is easy and therefore including the subtraction chain will not significantly increase the computation cost.

1. PDA selects n random numbers, $\mathcal{R} = \{r_1, r_2, \cdots, r_n \in \mathcal{Z}\}$, where
$$\sum_{i \in n} r_i = x \pmod q$$
2. PDA computes
$$y_i = g^{r_i} \pmod p$$
3. PDA computes
$$y = \prod_{i \in n} y_i \pmod p$$
to reveal y.

Lemma 4.1 *The above interaction is sound.*

Proof (sketch): Because $\sum_{i \in n} r_i = x \pmod q$, and $y_i = g^{r_i} \pmod p$, then

$$\begin{aligned} y &= \prod_{i \in n} y_i \pmod p \\ &= \prod_{i \in n} g^{x_i} \pmod p \\ &= g^x \pmod p \end{aligned}$$

□

Example 4.2 *Consider a toy example, where the PDA would like to compute* $g^{30} \pmod p$. *The protocol is illustrated as follows.*

1. *PDA selects 5 numbers:* $\{2, 3, 5, 7, 11\}$.
2. *PDA computes* $y_i = g^{x_i} \pmod p$, *where* $x = \{2, 3, 5, 7, 11\}$ *and gain the knowledge* $\{y_i\}$.
3. *PDA has the knowledge of* $g^2 \pmod p$, $g^3 \pmod p$, $g^5 \pmod p$, $g^7 \pmod p$ *and* $g^{11} \pmod p$. *Then, PDA computes*

$$\begin{aligned} \tilde{y_1} &= g^2 \cdot g^3 \cdot g^7 \pmod p \\ \tilde{y_2} &= g^5 \cdot g^5 \cdot g^{-1} \pmod p \\ \tilde{y} &= \tilde{y_1} \cdot \tilde{y_2} \cdot \tilde{y_2} \pmod p \end{aligned}$$

4. *PDA has successfully computed* $g^{30} \pmod p$.

In the above example, a passive attacker could find out that there were three multiplications performed in the last step for each y_i*. However, it is not possible for him to discover the exponent that was used in the above computation.*

Theorem 4.1 *Using the above protocol, the timing attack proposed in the earlier section will not be effective.*

Proof (sketch): The timing attack proposed earlier is only effective when square-and-multiply is used. In the above protocol, the PDA does not use this method to compute the final result, and therefore the attack will not work. □

5 Conclusions

We have described the result of an experiment which shows the insecurity of a PDA to conduct some computation like $g^x \pmod p$. We showed that by only observing the time required to perform such computation (or the battery life), we could derive the secret value that was used in the computation. We do not assume that the attacker has the full control over the PDA (c.f. [26]). We suggested several other methods to perform such a computation securely on a PDA. Our methods do not require any help from a normal PC.

References

[1] Private discussion with Ueli Maurer.

[2] Private discussion with Walter Aeillo.

[3] N. Asokan, G. Tsudik, and M. Waidner. Server-supported signatures. *Journal of Computer Security vol. 5 no. 1*, pages 91 – 108, 1997.

[4] D. Balfanz and E. W. Felten. Handheld computers can be better smart cards. *Proceedings of the 8th Usenix Security Symposium*, 1999.

[5] E. Biham and A. Shamir. Power Analysis of the Key Scheduling of the AES Candidates. *Second Advanced Encryption Standard (AES) Candidate Conference*, 1999.

[6] D. Boneh, N. Modadugu, and M. Kim. Generating RSA keys on a handheld using an untrusted server. *The First International Conference on Cryptology in India, Indocrypt 2000, LNCS 1977*, pages 271 – 282, 2000.

[7] D. Chaum, E. van Heijst, and B. Pfitzmann. Cryptographically strong undeniable signatures, unconditionally secure for the signer. *Interner Bericht, Fakultät für Informatik*, 1/91, 1990.

[8] J. F. Dhem, F. Koeune, P. A. Leroux, P. Mestré, J. J. Quisquater, and J. L. Willems. A Practical Implementation of the Timing Attack. *Proc. CARDIS '98, Smart Card Research and Advanced Applications*, 1998.

[9] W. Diffie and M. Hellman. New directions in cryptography. *IEEE IT*, 22:644–654, 1976.

[10] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31:469 – 472, 1995.

[11] D. M. Gordon. A survey of fast exponentiation methods. *J. Algorithms*, 27(1):129–146, 1998.

[12] G. Hachez, F. Koeune, and J. J. Quisquater. Timing Attack: What can be achieved by a powerful adversary? *http://www.dice.ucl.ac.be/crypto*.

[13] M. Joye. Recovering the lost efficiency of exponentiation algorithms on smart cards. *Electronics Letters*, page 38(19), September 2002.

[14] D. E. Knuth. *Seminumerical Algorithms, volume 2 of The Art of Computer Programming*. Addison-Wesley, Reading, Massachusetts, second edition, 1981.

[15] P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. *Advances in Cryptology, Crypto '96, LNCS 1109*, pages 104 – 113, 1996.

[16] A. Lenstra and E. Verheul. Selecting cryptographic key sizes. http://www.cryptosavvy.com/. *Extended abstract appeared in Commercial Applications*, Price Waterhouse Coopers, CCE Quarterly Journals, 3:3 – 9, 1999.

[17] U. Maurer and S. Wolf. The Relationship Between Breaking the Diffie-Hellman Protocol and Computing Discrete Logarithms. *SIAM Journal on Computing, vol. 28, no. 5*, pages 1689–1721, 1999.

[18] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, New York, 1997.

[19] C. Muench. *The Windows CE Technology Tutorial*. Addison Wesley, 2000.

[20] National Institute of Standards and Technology (NIST). Advances Encryption Standard development effort. http://csrc.nist.gov/encryption/aes/.

[21] J. J. P. Kocher and B. Jun. Differential power analysis. *Crypto'99*, pages 388–397.

[22] J. J. P. Kocher and B. Jun. Introduction to Differential Power Analysis and Related Attacks. *http://www.cryptography.com/dpa/technical*, 1998.

[23] Sharp. Zaurus: Personal mobile tool. *http://www.myzaurus.com/*.

[24] F. Stajano. *Security for Ubiquitous Computing*. John Wiley & Sons, 2002.

[25] D. R. Stinson. *Cryptography: Theory and Practice, second edition*. CRC Press, Boca Raton, New York, 2002.

[26] E. A. D. T. S. Messerges and R. H. Sloan. Power Analysis Attacks of Modular Exponentiation in Smartcards. *Cryptographic Hardware and Embedded Systems, First International Workshop*, pages 144–157, 1999.

[27] E. van Heijst and T. Pedersen. How to make efficient fail-stop signatures. *Advances in Cryptology - Eurocrypt '92*, pages 337–346, 1992.

[28] M. Williams. Korean start-up works hard to Pocket Linux. *InfoWorld News Online. http:// www.infoworld.com/articles/pi/ xml/00/04/17/ 000417pilinuxpda.xml*.