Secure Computations on Handheld Devices with the Help of an Untrusted Server

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

Abstract

Recently, handheld devices have become one of the most popular computing tools. Although handheld devices are able to perform anything that a PC can do, their lack of computing power makes it next to impossible to perform some heavy calculations. Hence it appears very useful to have a combination of a handheld with a PC, where the PC can perform heavy calculations to assist the handheld. However, we must be assured that the PC will not have learnt anything from the interaction. In this paper, we show two schemes which involve some server-aided computation where the server has not learnt anything from the interaction with the handheld device. The first scheme is to generate a strong *prime number in a handheld, which can be used as a candidate for the RSA algorithm. The second scheme is to allow the server to behave as an authentication oracle on behalf of the handheld. The handheld will prepare a message that needs to be authenticated by sending it to the server in a* blinded *form, so that the server will not learn about the message. On the other hand, the handheld will not learn about the server's secret.*

1 Introduction

Handheld computers have become an important part of our life. Currently, handheld computer can perform the same operations as a normal personal computer (PC), but it is more desirable as its small size means it fits in a pocket. The tendency is for users to use handheld computers instead of a laptop. Thus the real need is to make handheld computers more useful for electronic commerce applications.

Connecting a handheld to a PC to help it to perform some computations seems to be a reasonable attempt to allow the handheld to have some additional computing power. The PC helps the handheld to perform some computations, which finally will be learnt by the handheld. In this scenario, there is a new security issue, due to the PC involved in the calculation. In the ideal case, the PC is trusted and hence there will be no security problems. However, due to its nature, a handheld is normally carried around and it seems to be unreasonable to assume that it can always find its *trusted PC partner*. Therefore, performing a computation with an untrusted PC has become a new and challenging problem, which recently has been addressed by several researchers (for example [2]).

Currently, the algorithm that is widely used to support electronic commerce is RSA [12]. Basically, the RSA algorithm can be used as an encryption algorithm or a digital signature. A practical example of using RSA algorithm is the SSL (Secure Sockets Layer) algorithm (which is basically an authentication mechanism followed by an encryption algorithm), which is used to protect an on-line transaction on the Internet. RSA has become very popular in the practical world, with the introduction of *Public Key Infrastucture* (PKI).

The main security feature of the RSA algorithm is the size of the key. It is well-known that an RSA algorithm with a short key is insecure [16]. According to [7], the acceptable RSA key size is 1881 bits, to make it secure till the year 2020. Delfs and Knebl [5, 15] and others have argued that the prime numbers used in the RSA algorithm must be *strong* prime numbers.

Our Contributions

In this paper, we consider the problem of server-aided computation for a handheld device, where the server is untrusted. There are two main cases that we would like to address. In the first case, the untrusted server performs some computations for the handheld, but all the inputs are obtained from the handheld. In other words, the server does not provide any additional input to the interaction. It is desirable that the server should not learn anything from the interaction at the end of the protocol. In the second case, the untrusted server will give an additional input to the computation. However, this input should not be learnt by the handheld. On the other hand, the input that is given by the handheld should not be learnt by the server.

We would like to address these two types of server-aided computations by presenting two distinct schemes.

The aim of the first scheme is to *generate* strong prime numbers to be used in the RSA algorithm. As noted in [2], generating a 1024 bit RSA key on a handheld device can take as long as 15 minutes, and it will drain the battery of the handheld. The obvious solution is to allow the handheld to communicate with a desktop or server and have the server generate the key. Then the key can be downloaded to the handheld. In [2], Boneh, Modadugu and Kim have considered performing this computation with the help of an untrusted server. However, their solution only allows generation of an *unbalanced* RSA key, which makes it impractical (for example, an unbalanced RSA scheme cannot be used to generate a digital signature). In this paper, we propose a secure scheme that allows us to construct a strong prime number in the handheld.

In the second scheme, we allow the server to become an *authentication oracle* which can provide a message authentication code for a given message by the handheld. The purpose of the protocol is to allow the handheld to send a *blinded* message m to the server, and the server will compute $f(m) = am + b$ in $GF(p)$, for the secret values a and b which are only known by the server. At the end of the protocol, the handheld will learn about $f(m)$ for his chosen m, but will gain no knowledge about a and b involved in the computation. On the other hand, the server will not learn about m which is sent by the handheld. We note that this problem is closely related to the question of "computing with encrypted data" [11]: Alice holds some input x, Bob holds a function f, and Alice should learn $f(x)$ in a one-round protocol, where Alice sends to Bob an "encryption" of x, Bob computes f on the "encrypted" data x and sends the result to Alice who "decrypts" this to $f(x)$.

Our schemes are applicable to any type of low-power handheld devices, for example: cellular phones, MP3 players, PDAs, etc. In our test-bed implementation, we use a Pocket PC PDA, with 206 MHz, as an example of the low-power handheld devices together with a Pentium IV PC with 1.60 GHz. In our implementation, we use a serial connection between the PDA and the PC to enable the computation. We note that the connection can be extended, for example using a *socket* communication. We record the computation time together with the message-passing time involved in the calculation. Changing the application layer (for example, by using a WinSock implementation) will only change the message-passing time without effecting the computation time.

The rest of this paper is organized as follows. In the next section, we will review several previous related works in this area. In section 3, we will review some of the *tools* which will be used in our proposed schemes. In section 4, we will describe the first scheme which enables the handheld to construct a strong prime number and outline its security considerations. In section 5, we will present the second scheme which allows the handheld and the untrusted server to collaboratively produce a message authentication code for a given message m. Section 6 concludes the paper.

Notation: The length of a number p is the length of its binary representation and is denoted by $|p|_2$. The notation $a \in_R G$ means that a is selected randomly from the group G.

2 Related Works

In this section, we review several previous works which are related to our discussion.

2.1 Boneh, Modadugu and Kim's *Unbalanced* RSA Keys Generation

The unbalanced RSA system was originally proposed by Shamir in [13]. In this system, the modulus n is chosen of the form $n = pR$, where p is a 512 bit prime and R is a 4096 bit random number. This system relies on the fact that in a 4096 bit random number, there must be a prime factor of at least 512 bits long (the probability that it does not have such a factor is less than $1/2^{24}$). Therefore, the resulting modulus n is as hard to factor as a standard modulus $n = pq$. We omit the description about encryption and decryption in this system and refer the reader to [13] for a more detailed account.

In [2], Boneh et al. proposed a scheme to generate an unbalanced RSA keys with the help of an untrusted server. They provided two schemes in their paper, with one or two untrusted servers involved. However, we note that the use of two untrusted servers in their scheme is impractical due to the assumption that the untrusted servers will not collude and share the information they have learnt. We also note that the unbalanced RSA scheme cannot be used to provide digital signatures [2, 13].

3 Tools

3.1 Gordon's Strong Prime Number Generation

In [6], Gordon proposed a simple method to find a *strong*, random, large prime of a given number of bits. By definition, a strong prime p is a prime number satisfying

- $p = 1 \pmod r$
- $p = s - 1 \pmod s$
- $r = 1 \pmod t$

where r, s and t are all large, random primes of a given number of bits. We will incorporate his method in our first scheme.

3.2 Oblivious Transfer

Oblivious Transfer (OT) is the ubiquitous tool in secure computation. The concept of OT was introduced by Rabin [8]. In this scheme, there are two polynomial time parties, Alice and Bob. Alice sends a bit to Bob in such a way that with 1/2 probability Bob will receive nothing and 1/2 probability Bob will receive the same bit. Alice does not have any knowledge which event has happened. Since then, Rabin's idea has attracted a lot of attention, where the most notable advance is one-out-of-two OT (also known as *all-or-nothing disclosure of secrets* [3]), denoted as OT_1^2. This definition can be extended to OT_1^n, by executing n instances of OT_1^2 [9].

The OT_1^2 is illustrated as follows. In the following interaction, Bob is the sender and Alice is the receiver. Alice's private input is a bit c and Bob's private inputs are $a_0, a_1 \in G$. Common inputs are p and g, where g is the generator in Z_q.

1. Bob chooses $\delta \in_R G$ and sends δ to Alice.
2. Alice chooses $\alpha \in_R Z_q$, computes $\beta_c = g^\alpha$, $\beta_{(c\oplus 1)} = \frac{\delta}{\beta_c}$, and sends β_0, β_1 to Bob.
3. Bob verifies whether $\beta_0\beta_1 \overset{?}{=} \delta$, and aborts if not. Otherwise, he chooses $r_0, r_1 \in_R Z_q$, computes $(e_0, f_0) = (g^{r_0}, a_0\beta_0^{r_0})$ and $(e_1, f_1) = (g^{r_1}, a_1\beta_1^{r_1})$ and sends (e_0, f_0, e_1, f_1) to Alice.
4. Alice obtains a_c by computing $\frac{f_c}{e_c^\alpha}$.

4 Strong Prime Numbers Generation in a Secure Server Aided Computation

In this section, we describe our method to generate a strong prime number in a handheld with the help of an untrusted server. The implementation is done with a Pocket PC PDA connected to a PC via a serial connection, as shown in Figure 1. Our method uses the idea of [2] and [6]. We describe the method in two stages: the primality test algorithm and strong prime generation.

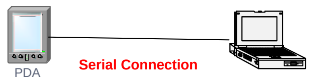

Figure 1. *Test-bed Architecture for Strong Prime Generation*

4.1 Primality Test Algorithm

We incorporate the method in [2] to determine whether a number is a prime or not. The idea is to compute $g^{(p-1)/2} \pmod p$ to test whether p is prime by realizing the fact that if $g^{(p-1)/2} = \pm 1 \pmod p$, where g is a random value in $\{1, \cdots, p-1\}$, then p is a prime with overwhelming probability [10]. The algorithm is illustrated in Algorithm 4.1, where the handheld $\mathcal{H}$ performs a test to determine whether p is a 512 bit prime number.

Algorithm 4.1: TEST_PRIME(p)

```
1.$\mathcal{H}$generates a 512 bit candidate $p$ that is not
divisible by small primes. We require that
$p \equiv 3$ mod $4$.
2.$\mathcal{H}$ generates a 4096 bit random number $R$.
3.$\mathcal{H}$ picks a random 160 bit $r$.
4.$\mathcal{H}$ picks a random 512 bit $a$.
5.$\mathcal{H}$ computes $z \leftarrow r + a(p-1)/2$
6.$\mathcal{H}$ computes $\mathcal{N} \leftarrow pR$
7.$\mathcal{H}$ sends $(\mathcal{N}, g, z)$ to the server $\mathcal{S}$.
8.$\mathcal{S}$ computes $\mathcal{X} \leftarrow g^z \pmod{\mathcal{N}}$.
9.$\mathcal{S}$ sends $\mathcal{X}$ back to $\mathcal{H}$.
10.$\mathcal{H}$ computes $\mathcal{Y} \leftarrow g^r \pmod p$.
11.$\mathcal{H}$ verifies whether $\mathcal{X} \overset{?}{=} \pm\mathcal{Y} \pmod p$. If not, then
 repeat Step 1. Otherwise, continue with step 12.
12.$\mathcal{H}$ runs a probabilistic primality test to verify that
$p$ is prime. If so, the algorithm has finished with a
prime $p$.
```

To verify the soundness of the algorithm, p is prime if and only if $\mathcal{X} = \pm\mathcal{Y}$, i.e. $g^{r+a(p-1)/2} = \pm g^r \pmod p$. The test will fail with overwhelming probability if p is not prime [10]. Having the primality test algorithm as above, we proceed with the strong prime number generation algorithm in the next section.

We note that the algorithm used in the above interaction is a probabilistic algorithm. Employing a deterministic algorithm, such as the one that is recently proposed by Agrawal, Kayal and Saxena [1], will eliminate step 12.

4.2 Strong Prime Number Generation

We follow the idea of [6] to produce a strong prime number. Firstly, we need to find two odd prime numbers, r and s, and apply Theorem 4.1 of [6].

Theorem 4.1 *If r and s are odd primes, then prime p satisfies*

$$p \equiv 1 \pmod r \equiv s-1 \pmod s$$

if p is of the form $p = u(r,s) + krs$ where

$$u(r,s) \equiv (s^{r-1} - r^{s-1}) \pmod{rs}$$

and k is an integer.

Proof: To prove this, we need to review Fermat's Theorem which states that if p is a prime that does not divide x, then $x^{p-1} = 1 \pmod p$. Since r and s are distinct primes, they cannot divide one another and we obtain

$$\begin{aligned} s^{r-1} &\equiv 1 \pmod r \\ r^{s-1} &\equiv 1 \pmod s \end{aligned}$$

Moreover,

$$\begin{aligned} s^{r-1} &\equiv 0 \pmod s \\ r^{s-1} &\equiv 0 \pmod r \\ krs &\equiv 0 \pmod r \\ &\equiv 0 \pmod s \end{aligned}$$

Now, by definition

$$\begin{aligned} p &\equiv u(r,s) + krs \\ &\equiv (s^{r-1} - r^{s-1}) \pmod{rs} + krs \end{aligned}$$

For any integers p, q and r, if $p = q$ then $p = q \pmod{r}$. Applying $p \pmod{r}$ to the above equation, we obtain

$$\begin{aligned} p &\equiv ((s^{r-1} - r^{s-1}) \pmod{rs} + krs) \pmod{r} \\ &\equiv s^{r-1} - r^{s-1} + krs \pmod{r} \\ &\equiv 1 - 0 + 0 \pmod{r} \\ &\equiv 1 \pmod{r} \end{aligned}$$

Using the same argument, applying $p \pmod{s}$ to the same equation, we obtain

$$\begin{aligned} p &\equiv ((s^{r-1} - r^{s-1}) \pmod{rs} + krs) \pmod{s} \\ &\equiv s^{r-1} - r^{s-1} + krs \pmod{s} \\ &\equiv 0 - 1 + 0 \pmod{s} \\ &\equiv -1 \pmod{s} \\ &\equiv s - 1 \pmod{s} \end{aligned}$$

◇

Because r and s are odd, then rs is odd, and krs is alternatively odd or even, with increasing k. Since all our interest is in odd prime numbers, it is convenient to consider p to be of the form

$$p_0 + 2krs$$

where if $u(r,s)$ is odd then $p_0 = u(r,s)$, otherwise $p_0 = u(r,s) + rs$. Since $2krs$ is even for all k, $p_0 + 2krs$ is always odd. Based on this discussion, we develop our algorithm to generate a strong prime number in handheld with the help of an untrusted PC as shown in Algorithm 4.2.

Algorithm 4.2: STRONG_PRIME(p)

```
1. Call Test_Prime(s)
2. Call Test_Prime(t)
3. $k \leftarrow 2$
4. $Finish \leftarrow False$
5. repeat
   { $r \leftarrow kt + 1$
   { if Test_Prime(r)
   {   then $Finish \leftarrow True$
   { $k \leftarrow k + 2$
   until $Finish$
6. $u \leftarrow (s^{r-1} - r^{s-1}) \pmod{rs}$
7. if ($u \pmod{2}$)
     then $p_0 \leftarrow u$
     else $p_0 \leftarrow u + rs$
8. $k \leftarrow 0$
9. $Finish \leftarrow False$
10. repeat
   { $p \leftarrow p_0 + 2krs$
   { if Test_Prime(p)
   {   then $Finish \leftarrow True$
   { $k \leftarrow k + 1$
   until $Finish$
11. return ($p$)
```

In the above algorithm, we incorporate the function `Test_Prime` which was developed earlier. This function requires an interaction with the untrusted server, but it will not leave any knowledge for the server after the interaction.

4.3 Security Consideration

The security of our scheme relies on the function `Test_Prime`, which requires an interaction with the untrusted server $\mathcal{S}$. From the interaction, $\mathcal{S}$ learns the value $z = a(p-1) + r$, but z is a random-looking 1024 bit value which contains some information about p. The best known algorithm for revealing p from z requires $2^{r/2}$ modular exponentiations [2] and due to the choice of r, the algorithm has security of approximately 2^{80}. Overall, we can claim that our algorithm is as secure as the algorithm proposed in [2]. If the scheme in [2] can be compromised, then our scheme can be broken. The full details of the proof will appear in the full version of this paper.

4.4 Improvement

We note that to generate a normal RSA modulus, the handheld is required to perform two consecutive processes as described above. We can speed up the computation by employing two different servers, which are connected through TCP/IP connection. The architecture is shown in Figure 2. Using this architecture, the speed is improved by a factor of 1.5 compared to the previous configuration.

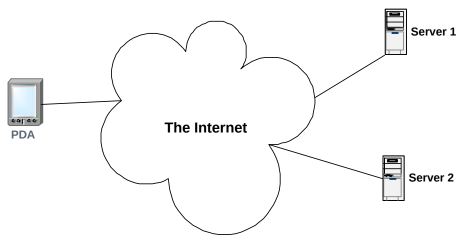

Figure 2. *Strong Prime Generation with the Help of Two Servers*

4.5 Implementation

We have implemented the algorithms proposed in this paper in a Pocket PC PDA with 206 MHz, together with its serial connection to a PC. We incorporate Shoup's NTL (Number Theory Library) [14] in our implementation. The result of the implementation is as follows.

Using the first configuration, we obtain the following result.

Process	*Time Elapsed (msec)*
Message Passing between PC and PDA	31
Computation time:	
PC	1,859
PDA	1,304

By replacing the configuration with the second architecture, we can speed up the process by 1.5 times.

5 Authentication Oracle Computation: An Application of Server-Aided Computation

In this section, we outline the second application of a server-aided computation. We note that this approach is similar to [4]. In this application, the server keeps two secret numbers, namely a and b, and whenever it accepts a message m, it will produce the authenticated tag which is $f(m) \equiv am + b \pmod p$ in $GF(p)$. The purpose of the protocol is as follows. The handheld $\mathcal{H}$ will send a message m to the server $\mathcal{S}$, and $\mathcal{S}$ will compute $f(m)$ and sends it back to $\mathcal{H}$. However, we require the following.

- At the end of the protocol, $\mathcal{S}$ should not learn about m.
- At the end of the protocol, $\mathcal{H}$ should not learn about a and b.

Intuitively, the basic idea is as follows. We assume that both parties agree that the computations are done over $GF(p)$, for some prime p. Let $|p|_2$ denote the size of p in binary. For simplicity, we assume that $|a|_2 \approx |m|_2 \approx |p|_2$. This can be achieved by using some trailing 0s padding if in fact $|a|_2 < |p|_2$, for instance. Represent a as its binary notation, according to the following: $a = \sum_{i\in|p|_2} a_i 2^{i-1}$, where $a_i \in \{0,1\}$, Similarly, represent m as its binary notation, namely $m = \sum_{i\in|p|_2} m_i 2^{i-1}$. The idea is to allow $\mathcal{S}$ to receive m_j, $j \in |p|_2$, if $a_i = 1, i \in |p|_2$, in a secret way. This is obtained by having $\mathcal{H}$ prepare the pair $(r_i, r_i + m_i)$ for a random number r_i and having $\mathcal{S}$ obtain what he wants via OT_1^2. When $\mathcal{S}$ knows that a_j is 1, $j \in |p|_2$, it will retrieve $r_i + m_i$. Otherwise, it will retrieve r_i. Then, it will multiply it with a_j. In conclusion, $\mathcal{S}$ will receive $am+r$. The process is summarized as $\tilde{m} = \sum_{i=1}^{|p|_2}\sum_{j=1}^{|p|_2}(a_{ij}m_{ij} + r_{ij}) = am + r$. Finally, $\mathcal{S}$ will add the result with b to make it $am + b + r$ and send it to $\mathcal{H}$. This will allow $\mathcal{H}$ to subtract $\sum_{i=1}^{|p|_2} r_i$ to obtain $am + b$. The complete protocol is illustrated in Algorithm 5.1. In this protocol, the computation is done in $GF(p)$.

Algorithm 5.1: PROTOCOL($\mathcal{H}$ *and* $\mathcal{S}$)

```
1. $\mathcal{H}$ prepares the pairs $(r_i, r_i + m_i)$.
2. For each pair $(r_i, r_i + m_i)$, $\mathcal{S}$performs an
   independent $OT_1^2$ with $\mathcal{H}$.
3. $\mathcal{S}$ sends $am + b + r$ to $\mathcal{H}$.
4. $\mathcal{H}$ subtracts it with $\sum_{i\in|p|_2} r_i$ to reveal $am + b$.
```

Lemma 5.1 *Protocol 5.1 is correct when both parties follow the protocol correctly.*

Proof: The sum that $\mathcal{H}$ obtains is $am+b+\sum_{i\in|p|_2} r_i$ from which $\sum_{i\in|p|_2} r_i$ can be easily subtracted to obtain $am + b$. In fact, in step 3, $\mathcal{H}$ receives $b + \sum_{i\in|p|_2}(a_i m_i + r_i)$, which is equal to $b + am + \sum_{i\in|p|_2} r_i$. $\diamond$

Lemma 5.2 *Protocol 5.1 is secure against the untrusted server.*

Proof: The security of the protocol relies on a secure OT_1^2. The server $\mathcal{S}$ will not learn the value m that $\mathcal{H}$ sent because it has been randomized by r. On the other hand, $\mathcal{H}$ will not learn a and b in the protocol. $\diamond$

Lemma 5.3 *This protocol is secure for one-round interaction.*

Proof: We note that at the end of the protocol $\mathcal{H}$ knows $y = am + b \pmod p$, for his choice of m. Knowing a pair of (m, y) in the above equation will not allow $\mathcal{H}$ to deduce the value of a and b, because there are exactly p different (a', b'), where $a', b' \in_R GF(p)$, which satisfy the equation with equal probability. However, if $\mathcal{H}$ runs the protocol for a second time, with the same set of a and b from s, then he will obtain the following equations:

$$y_1 \equiv am_1 + b \pmod p$$

$$y_2 \equiv am_2 + b \pmod p$$

for $m_1 \neq m_2$ and $y_1 \neq y_2$. Clearly, the rank of the above equation is 2, which will allow $\mathcal{H}$ to reveal a unique solution. Therefore, the protocol cannot be used for more than one interaction. An extension to $a_1 m^n + a_2 m^{n-1} + \cdots + a_n$ will allow it to be used for multiple rounds. $\diamond$

5.1 Implementation

We also implemented the second scheme in a Pocket PC PDA. The result is as follows.

Process	*Time Elapsed (msec)*
Message Passing between PC and PDA:	
$|p|_2 \approx 512$ *bits*	516
$|p|_2 \approx 1024$ *bits*	1,063
Computation time:	
PC	12,515
PDA	121,782

6 Conclusions and Further Works

We have shown two distinct applications that involve an untrusted server which enable a low-power device, such as a handheld computer, to perform some heavy computations. The applications that we have chosen show two different important features of a server-aided computation, which in one case, the server does not provide any extra information, and in other case, the server uses additional information. In both applications, the server will not learn anything from the interaction, but the client will learn the result of the computation without revealing the server's secret. We have done some experiments to show that the applications are feasible in a Pocket PC PDA, which is an example of a low-power device.

References

[1] M. Agrawal, N. Kayal, and N. Saxena. Primes in P. *preprint*, 2002.

[2] 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, Lecture Notes in Computer Science 1977*, pages 271 – 282, 2000.

[3] G. Brassard, C. Crepeau, and J. M. Robert. Information theoretic reductions among disclosure problems. *Proceedings of 27th FOCS*, 1986.

[4] Y.-C. Chang and C.-J. Lu. Oblivious Polynomial Evaluation and Oblivious Neural Learning. *Advances in Cryptology, Asiacrypt '01*, Lecture Notes in Computer Science Vol. 2248, pages 369 - 384, 2001.

[5] H. Delf and H. Knebl. Introduction to Cryptography. *Springer Verlag*, 2002.

[6] J. Gordon. Strong primes are easy to find. *Advances in Cryptology, Crypto '84*, pages 216 – 223, 1984.

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

[8] M. O. Rabin. How to exchange secrets by oblivious transfer. *Technical Report, TR-81, Computer Science Laboratory, Harvard*, 1981.

[9] M. Naor and B. Pinkas. Oblivious transfer and polynomial evaluation. *Proceedings of 31st Annual of ACM Symposium Theory of Computing*, pages 245 – 254, 1999.

[10] R. Rivest. Finding four million large random primes. *Advances in Cryptology - Crypto '90, Lecture Notes in Computer Science Vol. 537*, pages 625 – 626, 1990.

[11] R. L. Rivest, L. Adleman, and M. L. Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation, Academic Press*, 1978.

[12] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21, no. 2:120–126, 1978.

[13] A. Shamir. RSA for Paranoids. *CryptoBytes Vol. 1 No. 3*, 1995.

[14] V. Shoup. Number theory library. *Available online: http://www.shoup.net.*

[15] D. R. Stinson. *Cryptography: Theory and Practice, 2nd edition.* CRC Press, Boca Raton, New York, 2002.

[16] M. J. Wiener. Cryptanalysis of short RSA secret exponents. *IEEE Transactions on Information Theory* IT-36, 3:553 – 558, 1990.