*Int. J. Communications, Network and System Sciences*, **2013, 6, 1-6**
http://dx.doi.org/10.4236/ijcns.2013.61001 Published Online January 2013 (http://www.scirp.org/journal/ijcns)

# Public-Key Cryptosystems with Secret Encryptor and Digital Signature

**Boris S. Verkhovsky**
Computer Science Department, New Jersey Institute of Technology, Newark, USA
Email: verb73@gmail.com

Received October 20, 2012; revised November 28, 2012; accepted December 25, 2012

## ABSTRACT

This paper describes and compares a variety of algorithms for secure transmission of information via open communication channels based on the discrete logarithm problem that do not require search for a generator (primitive element). Modifications that simplify the cryptosystem are proposed, and, as a result, accelerate its performance. It is shown that hiding information via exponentiation is more efficient than other seemingly simpler protocols. Some of these protocols also provide digital signature/sender identification. Numeric illustrations are provided.

**Keywords:** Digital Signature; Discrete Logarithm; El Gamal Algorithm; Generator; Modular Exponentiation; Public Key Cryptosystem; Secure Communication; Sender Identification

## 1. Introduction and Basic Definition

This paper describes and *compares* special cases of several protocols for secure transmission of information: secret key establishment [1], poly-alphabetic analogue of affine protocol [2], El Gamal cryptographic protocol [3] and an analogue of the RSA algorithm [4], which crypto-immunity is not based on the difficulty of factorization {see EvESE algorithm in Sections 7 and 8}. The latter protocol {as well as all other protocols} is exclusively based on the difficulty of solving a discrete logarithm problem (DLP) [5,6] where the base $g$ is a generator in modular arithmetic with a modulus $p$ that is a safe prime.

**Definition1.1:** A prime integer $p$ is called a *safe* prime if

$$q := (p-1)/2; \tag{1.1}$$

is also a prime.

Therefore, for every $p \geq 7$ $q$ is *odd*. Moreover, if $p$ is a safe prime and $p \geq 23$, then $p \bmod 20 = 3$ or 7 or 19.

As it is demonstrated in [7], if $p$ is a safe prime, then the algorithm finding a generator $g$ is a computationally fast procedure. Some simplifications and innovations in this paper are based on the observation that $g = p-4$ is the generator for *every* safe prime $p \geq 7$.

The major innovation {Encryption via Exponentiation with a secret encryptor} is provided in Section 7 and demonstrated in Section 8.

## 2. System Parameters

If $p$ is a safe prime greater than or equal 7, then

$$g = p - 2^2; \tag{2.1}$$

is a generator for every $p$. Indeed, (1.1) and the Fermat Little Theorem (FLT) [5] imply that

$$g^q = \left(p-2^2\right)^q = (-1)^q\, 2^{p-1} = -1 \pmod p; \tag{2.2}$$

and

$$g^2 = \left(p-2^2\right)^2 = 2^4 \neq 1 (\bmod\, p); \tag{2.3}$$

otherwise $2^4 - 1 = 3 \times 5 = 0 \pmod p$; which is impossible if $p \geq 7$.

Parameters $p$ and $g$ are used by all participating users.

## 3. Establishment of Secret Encryptor

Step3.1: The sender (Alice) selects a large safe prime $p$ and transmits it to the receiver (Bob) via an open channel;

Step3.2: Respectively, Alice and Bob randomly select large integers $a$ and $b$ as their *private* keys;

Step3.3: Alice and Bob independently compute their *public* keys:

$$u := p - 2^{2a} \bmod p; \tag{3.1}$$

and

$$w := p - 2^{2b} \bmod p; \tag{3.2}$$

Step3.4: Using the public key of the receiver, Alice computes

$$e := w^a \bmod p; \tag{3.3}$$

 

Step3.5: Using the public key of the sender,
Bob computes

$$e \coloneqq u^b \bmod p\,; \tag{3.4}$$

where $e$ is their mutual *secret* encryptor.
*Remark*3.1: Parameters $a$, $b$ and $e$ must be distinct from $q$.

## 4. Several Ways to Hide Information

Suppose Alice wants to send a plaintext $m$ {which is represented in a numeric for $m$}, where the message $m$ satisfies

$$2 \le m \le p-2\,. \tag{4.1}$$

*Encryption* {Alice encrypts message $m$ and sends it to Bob, who decrypts it}:
*Remark*4.1: Alice has several alternatives: she can either create a ciphertext

$$c \coloneqq me \bmod p\,; \tag{4.2}$$

which is an equivalent of the ElGamal scheme [3]; or consider

$$C \coloneqq (m+e) \bmod p; \tag{4.3}$$

which is a poly-alphabetic shift cipher [8,9]; or she can split the secret key $e$ into two parts $e_{11}$ and $e_{12}$, and then consider

$$c_1 \coloneqq (e_{11}m + e_{12}) \bmod p. \tag{4.4}$$

The encryption (4.4) is an equivalent of the poly-alphabetic affine algorithm [2]. Yet another option is described in Section 6.

In general, Bob can apply any ordered binary operator $B$: $c \coloneqq (mBe) \bmod p$, provided that the inverse of $e$ exists, computable and it is unique for this operator.

Below we describe Alice's and Bob's actions if (4.2) or (4.3) are used. If (4.2) is applied, Alice transmits the ciphertext $c$ to Bob via an open channel.
*Decryption*: Using Alice's *public* key $u$ and his private key $b$, Bob computes a decryptor

$$d \coloneqq u^{p-1-b} \bmod p; \tag{4.5}$$

and finally, he recovers the plaintext:

$$f \coloneqq dc \bmod p\,;\{= m\}. \tag{4.6}$$

In (4.3) case the receiver computes his decryptor

$$D \coloneqq u^b \bmod p; \tag{4.7}$$

and

$$F \coloneqq (C-D) \bmod p;\{= m\}\,. \tag{4.8}$$

Furthermore, there are several ways {see (4.5) and (4.7)} to compute the decryptors $d$ and $D$ respectively. One requires exponentiation $p-1-a$ and another $a$. In the following Example4, $a$=35 and $p-1-a$=837 have binary "weights" 3 and 7 respectively. Therefore, $D$ requires fewer multiplications than $d$. Hence, it is faster to compute $D$ than $d$. In general, it is computationally advantageous if both Alice and Bob select their secret keys $a$ and $b$ with "smaller" binary "weights". However, these additional constraints provide clues for a potential intruder/ cryptanalyst.

Yet, there is an alternative algorithm for modular multiplicative inverse (AAMMI) proposed by the author of this paper in [10] and analyzed in [11]. Extensive computer experiments demonstrated that the AAMMI algorithm is computationally more efficient than (4.5); {see **Tables 1-3** below}.

In spite of computational simplicity, applications of the encryptor $e$ either in (4.2) or in (4.3) have negative sides: if at least one plaintext block, $m_i$, becomes known, an intruder can deduce every plaintext $m_k$. Indeed, since (4.2) implies

$$e \coloneqq c_i m_i^{-1} \pmod p\,; \tag{4.9}$$

then

$$m_k = c_k m_i c_i^{-1} \pmod p\,. \tag{4.10}$$

Analogously, the encryption (4.3) is vulnerable for cryptanalysis if at least one plaintext block, $m_i$, becomes known, the intruder can compute every plaintext $m_k$. Indeed, since (4.3) implies

$$e \coloneqq c_i - m_i \pmod p\,; \tag{4.11}$$

then

$$m_k = c_k - (c_i - m_i) \pmod p\,. \tag{4.12}$$

In order to overcome these deficiencies, the sender must compute dedicated encryptor $e(m)$ for *every* plaintext

**Table 1. Computation of MMI.**

| | | | |
|---|---|---|---|
| ***q*=53** | ***e*=49** | 4 | 1 |
| Stack | 1 | 12 | *** |
| ***z*=13** | 12 | ***1*** | ***0*** |

**Table 2. Computation of MMI of 195 modulo *p*–1=862.**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **862** | **195** | 82 | 31 | 20 | 11 | 9 | 2 | 1 |
| Stack | 4 | 2 | 2 | 1 | 1 | 1 | 4 | *** |
| ***z*=389** | 88 | 37 | 14 | 9 | 5 | 4 | ***1*** | ***0*** |

**Table 3. Algorithm for MMI $\{1 \le a \le t-1\}$.**

| | | | | | |
|---|---|---|---|---|---|
| $a_0 = t$ | $a_1 = a$ | $a_2$ | .. | $a_{n-1}$ | $a_n$ |
| *stack* | $q_1$ | $q_2$ | .. | $q_{n-1}$ | *** |
| $b_0 = \pm z$ | $b_1$ | $b_2$ | .. | $b_{n-1} = 1$ | $b_n = 0$ |

 

block $m$, and respectively, the receiver must compute dedicated decryptor $d(m)$ to recover this plaintext. Needless to say that this requires additional computational efforts, *i.e.*, it slows the transmission of information.

## 5. Dynamic Establishment of Secret Key between Sender and Receiver

Step5.1: The sender (Alice) selects a large safe prime $p$ and transmits it to the receiver (Bob) via an open channel;
Step5.2: Respectively, Alice and Bob randomly select large integers $a$ and $b$ as their private keys;
Step5.3: Alice and Bob independently compute their public keys:

$$u := p - 2^{2a} \bmod p; \quad (5.1)$$

and

$$w := p - 2^{2b} \bmod p; \quad (5.2)$$

Step5.4: The sender (Alice) selects an ephemeral secret key $x$, and using the public key of the receiver, she computes an ephemeral encryptor

$$e := w^x \bmod p; \quad (5.3)$$

and her *hint*

$$h := p - 2^{2x} \bmod p; \quad (5.4)$$

*Remark*5.1: Parameters $a$, $b$, $e$ and $x$ must be distinct from 1, $q$ and $p-1$.
Step5.5: The sender computes the ciphertext

$$c := me \bmod p; \quad (4.2);$$

and transmits $\{c,h\}$ to the receiver;
Step5.6: Using his public key, the receiver computes an ephemeral decryptor

$$d := h^{p-1-b} \bmod p; \quad (5.5)$$

and then computes

$$f := dc \bmod p \{= m\} \quad (4.6);$$

*Remark*5.2: As an alternative, the sender computes the ciphertext

$$C := (m+e) \bmod p \quad (4.3);$$

and sends $(C,h)$ to the receiver.

In this case, the receiver computes his ephemeral decryptor $D := h^b \bmod p$; (4.7) and then computes $F := (C-D) \bmod p. \{= m\}$; (4.8), *i.e.*, he recovers the original plaintext.

## 6. Numeric Illustrations-1

In the Examples 1 and 2 modifications that simplify the computation are introduced, and as a result, accelerate the secure transmission of information.

First of all, notice that there is no necessity to search for a generator in order to compute the hints and public keys.

***Example*1** {Alice sends message $m$ to Bob}:
Let $p$=47; $g$=4; $b$=25; $x$=33; and $m$=42.
*Encryption*:
Step6.1.1: Bob pre-computes his public key
$w := g^b = 2^{2b} \pmod p = 2^{50} = 16 \pmod{47}$;
Step6.2.1: Alice computes her *hint*

$$h := g^x = 2^{2x} = 2^{66} = 6 \pmod{47}$$

Step6.3.1: Using Bob's public key $w$, Alice computes her *encryptor e* and the ciphertext $C$:

$$e := w^x \bmod p = 16^{33} = 36 \pmod{47};$$

$$C := (m+e) \bmod p = 42 + 36 = 31 \pmod{47};$$

Step6.4.1: Alice sends the ciphertext and her hint to Bob:
$\{C,h\} = \{31,\ 6\}$;
*Decryption*:
Step6.5.1: Using his private key $b$, Bob computes his decryptor

$$D := h^b = 6^{25} \bmod 47 = 36 \quad \{= g^{bx} = e\};$$

Step6.6.1: Bob decrypts the ciphertext:

$$F := C - D = (31-36) \bmod 47 = 42 \quad \{= m\}.$$

*i.e.*, he recovers the plaintext $m$.

***Example*2:** Let now $p$=863; $g$=4; $a$=35; $y$=21; $m$=754; and suppose Bob wants to send a secret message $m$ to Alice.

***Encryption*:**
Step6.1.2: Alice pre-computes her public key

$$u := g^a = 2^{70} \pmod{863} = 660;$$

Step6.2.2: Bob computes his hint

$$h := g^y = 2^{42} \pmod{863} = 392;$$

Step6.3.2: Bob computes the ephemeral secret *encryptor e* and *ciphertext C*:

$$e := u^y = 660^{21} = \pmod{863} = 171;$$

$$C := m + e = (754+171) \bmod 863 = 62 \quad (5.1);$$

Step6.4.2: Bob sends the ciphertext $C$ and his hint $h$ to Alice.

***Decryption*:**
Step6.5.2: Using her private key $a$ and hint $h$, Alice computes her ephemeral *decryptor D*:
$D := h^a = 392^{35} \bmod 863 = 171 \quad \{= g^{ay}\}$;
Step6.6.2: Alice decrypts the ciphertext:

$$f := C - D = (62-171) \bmod 863 = 754 \quad \{= m\};$$

*i.e.*, she recovers the plaintext $m$.

 

*Remark*6.3: If the plaintext is intelligible, Alice accepts it as legitimate, *i.e*. this protocol also provides a *digital signature*.

## 7. Encryption via Exponentiation with Secret Encryptor (EvESE)

The following encryption/decryption scheme requires fewer exponentiations than the schemes described above.

***System design*:**

a). Users Alice and Bob select their private keys $a$ and $b$ respectively and then compute their public keys:

$$u_2 = g^a \bmod q; \tag{7.1}$$

and

$$w_2 = g^b \bmod q; \tag{7.2}$$

b). each user computes his/her *common secret encryptor*

$$e := u_2^b = w_2^a \pmod p; \tag{7.3}$$

c). if $e$ and $p-1$ are relatively prime,
*i.e*., if

$$\gcd(e, p-1) = 1; \tag{7.4}$$

then the users compute an integer $d$ that satisfies the equation

$$ed \bmod (p-1) = 1; \tag{7.5}$$

else, {if $\gcd(e, p-1) = v$ },
they consider

$$e := e/v; \tag{7.6}$$

and compute $d$ in (7.5). To find $d$ (decryptor), both users compute

$$d := e^{q-2} \bmod (p-1); \tag{7.7}$$

*Remark*7.1: As an alternative the users can apply the algorithm for MMI, [10,11]; {see Examples 3, 4 and **Table 3**};

***Encryption*:**

d). A sender of message $m$ computes the ciphertext

$$c_2 := m^e \bmod p\ ; \tag{7.8}$$

e). the ciphertext $c_2$ is sent to a receiver via an open communication channel{notice that no any hint is necessary!};

***Decryption*:**

f). The receiver computes

$$f_2 := c_2^d \bmod p;\ \{= m\}; \tag{7.9}$$

***Validation of EvESE Algorithm*:**

Since $\gcd(e, p-1) = 1$, *i.e*., $e$ is *odd* and distinct from $q$, then (7.5) implies the existence of an integer $k$ such that

$$ed = 1 + (p-1)k. \tag{7.10}$$

Therefore, (7.8) and the FLT imply that

$$\begin{aligned} f_2 &= c_2^d = m^{ed} = \\ &= m \times \left(m^{p-1}\right)^k = m \times 1^k \pmod p = m. \end{aligned} \tag{7.11}$$

*Remark*7.2: Although (7.8) and (7.9) resemble the RSA protocol [4], there are three distinct features:
1). the encryptor $e$ is a *secret* (not public!) key;
2). modulo reduction is executed with the public prime $p$ rather than with the product of two secret primes individually selected for each user;
3). this scheme does not require additional computations for sender identification; in other words, it automatically provides the digital signature.

## 8. Numeric Illustrations-2

***Example*3:** Let $p$=107; $g$=4; $a$=33; $b$=28; and $m$=42.

*System design*: {either $a$ or $b$ must be selected in such a way that $\gcd(e, q) = 1$ otherwise the MMI of $e$ modulo $q$ does not exist}; $u_2 := g^a \bmod q = 4^{33} \bmod 53 = 7$; and

$$w_2 := g^b \bmod q = 4^{28} \bmod 53 = 16;$$

$$e := u_2^b = 7^{28} = w_2^a = 16^{33} \bmod 53 = 49;$$

$$d := e^{q-2} \bmod q = 49^{51} \bmod 53 = 13\ . \tag{8.1}$$

*Remark*8.1: If $q$ is a very large integer, the computation of the multiplicative inverse in (8.1) requires many multiplications even if the "square-and-multiply" algorithm is used. Yet, computation of the MMI is much simpler using the algorithm described in [10] as it is shown in the **Table 1**:

Since the number of columns in the **Table 1** is *even*, then $d$:=$z$.

*Verification*: $de \bmod q = 13 \times 49 \bmod 53 = 1$**.**

Explanations are provided in Section 9.

***Encryption*:** A sender computes the ciphertext $c_2$, and transmits it to the receiver:

$$c_2 := m^e \bmod p = 42^{49} \bmod 107 = 48;$$

***Decryption*:** {using the decryptor $d$, the receiver recovers the plaintext}:

$f_2 := c_2^d \bmod p = 48^{13} \bmod 107 = 42$. Therefore, the message $m$ is correctly recovered by the receiver.

***Example*4:** Let now $p$=863; $g$=4; $a$=35; $b$=49; and $m$=756.

*System design*:

$$u_2 := g^a \bmod q = 4^{35} \bmod 863 = 660;$$

and

$$w_2 := g^b \bmod q = 4^{49} \bmod 863 = 213;$$

 

$$e := u_2^b = 660^{49} = w_2^a = 213^{35} \bmod 863 = 195.$$

Each user computes the MMI of 195 modulo 862; details are shown in the **Table 2**:

Since the number of columns in the **Table 2** is *odd*, then $z = p - 1 - d$ ; therefore

$$d = 2q - 389 = 473.$$

***Encryption*:**

$$c_2 := m^e \bmod p = 756^{195} \bmod 863 = 166;$$

***Decryption*:**

$$f_2 := c_2^d \bmod p = 166^{473} \bmod 863 = 756 = m.$$

## 9. Algorithm for MMI*a*z mod *t*=1

In this section, **Table 3** briefly describes the algorithm for the MMI. For more details, see [10] and [11].

In the following **Table 3** both integers *a* and *t* are the inputs and integer *z* is the output.

$$\text{Assign } a_0 := t;\ a_1 := a;$$

**for** $k = 1, 2, \cdots, n-1$ iterate and store in a stack the quotients

$$q_k = \lfloor a_{k-1} / a_k \rfloor;$$

$$a_{k+1} := a_{k-1} - q_k a_k ;$$

**repeat until** $a_n = 0$; or $a_n = 1$;
**if** $a_n = 0$, **then** the MMI does not exist;
**if** $a_n = 1$, **then** assign $b_n := 0$ ; and $b_{n-1} := 1$; **for** $k$ **from** $n-1$ **down to** 1 iterate $b_{k-1} = q_k b_k + b_{k+1}$ ;
**if** *n* is *odd*, **then** $z = b_0$, **else** $z = t - b_0$.

## 10. Complexity Analysis of EvESE Cryptosystem

On the system design level, each user performs two exponentiations to compute their public key (7.1) and (7.2), and the secret encryptor (7.3).

For the encryption, it is necessary to perform only one exponentiation (7.5). Analogously, for decryption, every receiver performs only one exponentiation (7.7). Although for the purpose of maintaining a high security level we need periodically select new private keys and re-compute the encryptor and decryptor, we do *not need to send the hints* with every block of transmitted message like it is done in the ElGamal algorithm. Since this algorithm is based on the difficulty of the DLP, it has certain advantages over the RSA algorithm based on factorization. One of them is that the encryptor and decryptor provide a digital signature (sender identification) since they are computed for communication between the specific pair of users (Alice and Bob). On the other hand, in the described form the algorithm cannot be applied for broadcasting of secret messages to several users. However, using the "carrousel" DHKE, we can generalize the proposed algorithm for communication among several users.

## 11. Algorithm EvESE Modification

If *e* is a power of 2, or $e = s \times 2^t$; where $s \geq 3$; and *s* is a small integer, then the algorithm does not work, since the decryptor either does not exist or the encryptor is too small. One option is to select new private keys *a* and *b* in hope that new *e* will be more suitable. However, this is a non-deterministic procedure. Besides it requires many multiplications of multidigit-long integers.

There are other simple options if $p>11$, *e* is *even* and $4 \leq e \leq p-3$:

a). if $e \neq q+1$, then assign $e := e-1$;
b). if $e \neq q-1$, then assign $e := e+1$.
Finally, if $e=2$, then $e := q+2$.

## 12. Conclusions

Notice that the ElGamal algorithm is just one of several constructive demonstrations how to dynamically apply a secret key for secure communication. Indeed, the *inverse* value of the decryptor is the same as encryptor. Therefore, both parties are dynamically establishing the common secret key (encryptor *e*) and then use it to hide the message *m* by multiplying it on the encryptor. Another possibility: instead of multiplying, they *add* the encryptor *e* to *m* or can consider exponentiation $m^e \bmod p$. Therefore, in the case of addition in (4.3), they eliminate two multiplications for every plaintext since $D=e$.

However, both protocols require twice more exponentiations than the EvESE algorithm described in the Section 7 and demonstrated in Section 8. As the analysis shows, the most efficient is to use the exponentiation (7.5) for the encryption.

I wish to express my appreciation to Dr. Roberto Rubino for his comments that improved the style of this paper.

## REFERENCES

[1] W. Diffie and M. E. Hellman, "New Directions in Cryptography", *IEEE Transactions on Information Theory*, Vol. 22, No. 6, 1976, pp. 644-654. doi:10.1109/TIT.1976.1055638

[2] A. J. Menezes, P. C. van Oorschot and S. A. Vanstone, "Handbook of Applied Cryptography", CRC Press, Boca Raton, 1997.

[3] T. ElGamal, "A Public Key Crypto-System and a Signature Scheme Based on Discrete Logarithms", *IEEE Transactions on Information Theory*, Vol. 31, No. 4, 1985, pp. 469-472. doi:10.1109/TIT.1985.1057074

[4] R. L. Rivest, A. Shamir and L. M. Adleman, "A Method of Obtaining Digital Signature and Public-Key Crypto-

 

systems", *Communication of ACM*, Vol. 21, No. 2, 1978, pp. 120-126. doi:10.1145/359340.359342

[5] C. F. Gauss, "Disquisitiones Arithme Ticae", 2nd Edition, Springer, New York, 1986.

[6] P. Garrett, "Making, Braking Codes: An Introduction to Cryptology", Prentice Hall, Upper Saddle River, 2001.

[7] B. Verkhovsky, "Deterministic Algorithm Computing All Generators: Application in Cryptographic Systems Design", *International Journal of Communications*, *Network and System Sciences*, Vol. 5, No. 11, 2012, pp. 715-719. doi:10.4236/ijcns.2012.511074

[8] J. Katz and Y. Lindell, "Introduction to Modern Cryptography", Chapman and Hall/CRC Press, New York, 2008.

[9] B. A. Forouzan, "Cryptography and Network Security", McGraw Hill, Boston, 2008.

[10] B. Verkhovsky, "Multiplicative Inverse Algorithm and Its Space Complexity", Annals of European Academy of Sciences, EAS, Liege, 2004, pp. 110-124.

[11] B. Verkhovsky, "Space Complexity of Algorithm for Modular Multiplicative Inverse", *International Journal of Communications*, *Network and System Sciences*, Vol. 4, No. 6, 2011, pp. 357-363. doi:10.4236/ijcns.2011.46041

 