*Int. J. Communications, Network and System Sciences*, **2012, 5, 715-719**
http://dx.doi.org/10.4236/ijcns.2012.511074 Published Online November 2012 (http://www.SciRP.org/journal/ijcns)

# Deterministic Algorithm Computing All Generators: Application in Cryptographic Systems Design

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

Received September 27, 2012; revised November 4, 2012; accepted November 8, 2012

## ABSTRACT

Primitive elements play important roles in the Diffie-Hellman protocol for establishment of secret communication keys, in the design of the ElGamal cryptographic system and as generators of pseudo-random numbers. In general, a deterministic algorithm that searches for primitive elements is currently unknown. In information-hiding schemes, where a primitive element is the key factor, there is the freedom in selection of a modulus. This paper provides a fast deterministic algorithm, which computes every primitive element in modular arithmetic with special moduli. The algorithm requires at most $O(\log^2 p)$ digital operations for computation of a generator. In addition, the accelerated-descend algorithm that computes small generators is described in this paper. Several numeric examples and tables illustrate the algorithms and their properties.

**Keywords:** Diffie-Hellman Key Exchange; ElGamal Cryptosystem; Generator; Generator of Pseudo-Random Numbers; Information Hiding; Primitive Element; Safe Prime

## 1. Introduction and Basic Definitions

To ensure a high level of crypto-immunity of some cryptographic systems, it is necessary to select a system parameter $g$ (called a primitive element) that satisfies certain conditions.

The primitive elements are used in the Diffie-Hellman secret key establishment (DHKE) protocol [1] and in the ElGamal algorithm [2] for secure exchange of information via open channels. They are also used in the design of generators of pseudo-random numbers [3].

In modular arithmetic, a primitive element $g$ modulo $p$ is an integer having the property that every integer $h$ coprime with $p$ can be expressed as a power of $g$ modulo $p$.

Therefore, powers of $g$ *generate* all non-zero elements of the multiplicative group of integers modulo $p$.

**Definition 1.1:** If an integer $g$ has a property that for every integer $1 \le h \le p-1$ there exists a corresponding integer $x$ such that

$$g^x \bmod p = h, \tag{1.1}$$

then $g$ is called a primitive element (generator, in short) and $x$ is called the discrete logarithm of $h$ to the base $g$ modulo $p$.

For every prime $p$ there exist several generators. For instance, if $p$ = 31, then $g$ = 3, 11, 12, 13, 17, 21, 22 and 24 are generators.

Leonhard Euler discovered the primitive elements, and Carl F. Gauss described their properties in [4]. A mathematically-oriented reader can find further results in [5,6].

The elliptic curve cryptography (ECC), initially described in [7,8], is an analogue of the ElGamal protocol. As a result, the ECC also requires selection of a point $G$ on the elliptic curve, which is an analogue of the generators in cyclic groups based on real integers. However, an efficient algorithm that computes $G$ is an open problem.

## 2. Verification Procedure

In order to verify whether $g$ is a generator for prime $p$, consider all factors of $p-1$.

**Proposition 2.1:** Suppose

$$p-1 = f_1^{e_1} f_2^{e_2} \cdots f_m^{e_m}; \tag{2.1}$$

where every integer $f_k \ge 2$ and every integer $e_k \ge 1$; if

$$g^{(p-1)/f_k} \bmod p \ne 1; \tag{2.2}$$

holds for every $k = 1, 2, \cdots, m$, then $g$ is a generator [9].

*Example* 2.1: Suppose $p$ = 71; let us find a generator. Since $p-1 = 70 = 2 \times 5 \times 7$, then $g$ is a generator if and only if $g^{35} \bmod 71 \ne 1$;

$$g^{14} \bmod 71 \ne 1; \quad \text{and} \quad g^{10} \bmod 71 \ne 1. \tag{2.3}$$

 

**Table 1** shows values of $g^n \bmod 71$.

Since $g = 7$ satisfies every condition in (2.3), therefore, after *fifteen exponentiations* we find that it is the generator for $p = 71$.

Although the conditions (2.2) are straight-forward to verify, if $m$ is large, then (2.2) requires factorization of $p-1$ [10] and $m$ exponentiations for each potential candidate. Also, if at least one of these conditions does not hold, it is necessary to consider the next candidate. In general, non-deterministic algorithms are typical for various problems in modular arithmetic.

## 3. Two Deterministic Algorithms

In information-hiding schemes, where a primitive element is necessary, there are alternatives for selecting a prime modulus.

**Definition 3.1:** If both $p$ and $(p-1)/2$ are primes, then $p$ is called a *safe* prime [9].

*Example* 3.1: Integers 5, 7, 11, 23, 47, 59, 83, 107, 167, 179, 227, 263, 347, 863, 9839, 6935459, 8331923 and 9522167 are examples of safe primes. Although the search for safe primes is not the goal of this paper, the following properties of safe primes eliminate numerous false candidates: $p \bmod 20 = 3$ or 7 or 19, otherwise $(p-1)/2$ is not a prime.

A non-deterministic algorithm for the selection of safe primes is provided in [9] {see 4.86 Algorithm}.

**Proposition 3.1:** If $p \geq 7$ is a safe prime, then

$$g_1 := p - \left\lfloor \sqrt{p} \right\rfloor^2; \tag{3.1}$$

is a generator.

*Remark* 3.1: Although (3.1) does not always compute the smallest generator, its value is small in comparison with $p$: $g_1 = O\left(2\sqrt{p}\right)$.

Indeed, $2 \leq g_1 \leq \left\lfloor 2\sqrt{p+2} \right\rfloor - 3.$ (3.2)

Moreover, if $p = \{7, 23, 47, 167\}$, then

$$g_1 = \left\lfloor 2\sqrt{p+2} \right\rfloor - 3 = \{3, 7, 11, 23\}.$$

If $p = \{11, 83, 227\}$, then for every $p$

$$g_1 = 2.$$

**Proposition 3.2:** If $p \geq 7$ is a safe prime, then

$$g_2 = (3p-1)/4 \text{; is a generator.} \tag{3.3}$$

*Proofs* of both propositions are provided in the next section.

**Table 2** provides fifteen examples of safe primes and three corresponding generators for each of them. For every safe prime, the procedures (3.1), (3.3) as well as (6.4), described in the sixth section, are deterministic and require at most one integer multiplication. As a result, in the ElGamal algorithm, the generator can be periodically renewed for enhancement of communication security. Notice that in **Table 2** for $p$ = 11, 23, 47, 59, 167, 179 and 347 $g_1 \leq g(m)$; and for the rest of these primes it is otherwise, *i.e.*, $g_1 > g(m)$.

## 4. Algorithm Computing All Generators

Both Propositions 3.1 and 3.2 are special cases of more general proposition.

**Proposition 4.1:** Let $p \geq 7$ be a safe prime; then for every integer $z$ that satisfies the inequalities

$$2 \leq z \leq p-2; \tag{4.1}$$

$$g = \left(p - z^2\right) \bmod p; \tag{4.2}$$

is a generator.

*Proof*: Definition 3.1 implies that for a safe prime

$$p - 1 = 2q; \tag{4.3}$$

where $q$ is an *odd* prime. Therefore, $g$ is a generator because by Fermat's Little Theorem [3,4]

$$g^q = \left(p - z^2\right)^q = (-1)^q z^{p-1} = -1 \pmod p \tag{4.4}$$

and $$g^2 = \left(p - z^2\right)^2 = z^4 \neq 1 \pmod p. \tag{4.5}$$

**Table 1. Choice of generator for $p$ = 71.**

| $g \backslash n$ | $n$ = 10 | $n$ = 14 | $n$ = 35 |
|---|---|---|---|
| $g$ = 2 | 30 | 54 | 1 |
| $g$ = 3 | 48 | 54 | 1 |
| $g$ = 5 | 1 | 57 | 1 |
| $g$ = 6 | 20 | 5 | 1 |
| $g$ = 7 | 45 | 54 | 70 |

**Table 2. Safe primes and corresponding generators.**

| $p$ | 11 | 23 | 47 | 59 | 83 | 107 | 167 | 179 | 227 | 263 | 347 | 863 | 983 | 2063 | 9839 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $g_1$ | 2 | 7 | 11 | 10 | 2 | 7 | 23 | 10 | 2 | 7 | 23 | 22 | 22 | 38 | 38 |
| $g(0)$ | 8 | 17 | 35 | 44 | 62 | 80 | 125 | 134 | 170 | 197 | 260 | 647 | 737 | 1547 | 7379 |
| $g(m)$ | 2 | 5 | 5 | 2 | 6 | 8 | 15 | 2 | 14 | 15 | 20 | 47 | 35 | 65 | 69 |

 

Suppose that (4.5) does not hold, *i.e*., there exists at least one $z = Z$, for which $Z^4 = 1 \pmod p$.

Therefore, it implies that

$$Z^4 - 1 \equiv (Z+1)(Z-1)\left(Z^2+1\right) \pmod p = 0. \quad (4.6)$$

However, none of three factors in (4.6) are congruent with *zero* modulo *p*: the first two are excluded by the constraints in (4.1); and the case

$$Z^2 = -1 \pmod p; \quad (4.7)$$

is also infeasible, since by Euler's criterion of quadratic residuosity [4,9]

$$(-1)^{(p-1)/2} = (-1)^q \bmod p = -1; \quad (4.8)$$

which implies that –1 does not have a square root modulo *p*. In other words, *z* in (4.7) has no real integer solution. Q.E.D.

*Proof of Proposition* 3.2: It is easy to verify that (3.3) is a special case of (4.2) for *z* = *q*. Indeed, consider

$$\begin{aligned} \left(p - q^2\right) &\equiv \left[p - \left(p^2 - 2p + 1\right)/4\right] \\ &\equiv \left[(3p-1) + \left(3p - p^2\right)\right]/4 \pmod p \end{aligned} \quad (4.9)$$

Since for every safe prime *p* mod 4 = 3 holds, then

$$\left(3p - p^2\right) \bmod 4 = 0. \quad (4.10)$$

In addition,

$$\left(3p - p^2\right) \bmod p = 0. \quad (4.11)$$

Because the last term in (4.9) is an integer, therefore, (4.9)-(4.11) imply (3.3). Q.E.D.

**Table 3** lists *all* generators for *p* = 47 as functions of parameter *z* {see (4.2)}.

Since every safe prime *p* has

$$\varphi(\varphi(p)) = q - 1; \quad (4.12)$$

distinct generators, {here $\varphi(x)$ is Euler'stotient function [4]}, the function *g*(*z*) generates each of them if *z* is changing on the interval [2, *q*]; {see (4.1) and **Table 3**}. In addition, (4.2) implies that

$$g(p - z) \equiv g(z) \pmod p. \quad (4.13)$$

For an illustration of (4.13), compare $g(z)$ in **Table 3** for *z* = 23 and 24; or for *z* = 22 and 25.

**Proposition 4.2:** If $p \geq 7$ is a safe prime, then for every $k = 0,1,\cdots,q-1$

$$g_3 = \left[(3p-1)/4\right]\left[(p+1)/4\right]^k \bmod p; \quad (4.14)$$

is a generator.

## 5. Algorithm for a "Small" Generator

This algorith *m* computes a small generator for every safe prime $p \geq 7$.

**Step 5.1:** Compute

$$g(0) := (3p-1)/4; \quad (5.1)$$

**Step 5.2:** Compute

$$m := \left\lfloor \sqrt{g(0) - 2} - 1/2 \right\rfloor; \quad (5.2)$$

**Step 5.3:** Compute the generator

$$g(m) := \left[g(0) - m(m+1)\right] \pmod p \quad (5.3)$$

## 6. Validation of Algorithm (5.1)-(5.3)

Let us consider a sequence of *acceleratingly* decreasing generators.

Let $g(0) := g_2 = (3p-1)/4$ (3.3);

Consider $z = (p-3)/2;$ (6.1)

and let

$$g(1) := \left(p - z^2\right) = \left(10p - 9 - p^2\right)/4 \pmod p, \quad (6.2)$$

where every term in (6.2) is an integer. Hence (4.10), (4.11) and (6.2) imply that

$$g(1) = (10p - 9 - 3p)/4 = (7p-9)/4 \bmod p.$$

**Proposition 6.1:** For every safe prime *p* there exists a subset *S* of generators $g(0), g(1), \cdots, g(m)$ such that for $k = 1,2,\cdots,m$ holds

$$g(k-1) - g(k) = 2k. \quad (6.3)$$

**Table 3. *p* = 47 and corresponding generators *g*(*z*).**

| *z* | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *g*(*z*) | 43 | 38 | 31 | 22 | 11 | 45 | 30 | 13 | 41 | 20 | 44 | 19 |
| *z* | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |
| *g*(*z*) | 39 | 10 | 26 | 40 | 5 | 15 | 23 | 29 | 33 | 35 | 35 | 33 |

 

For instance, if $p$ = 83, then $g(0)$ = 62; $g(1)$ = 60; $g(2)$ = 56; $g(3)$ = 50; $g(4)$ = 42; $g(5)$ = 32; $g(6)$ = 20; $g(7)$ = 6.

Therefore, (6.3) implies that for $k = 1, 2, \cdots, m$

$$g(k) = \left[ g(0) - k(k+1) \right] (\mathrm{mod}\ p). \quad (6.4)$$

As a result, we derive a monotone decreasing sequence of generators

$$g(0) > g(1) > \cdots > g(m-1) > g(m); \quad (6.5)$$

where an optimal $m$ minimizes $g(k)$, {see (6.4)}, under the constraints

$$g(k) \geq 2 \quad \text{and} \quad k(k+1) \leq g(0) - 2. \quad (6.6)$$

*Remark* 6.1: In the worst case

$$g(m) = \mathrm{O}\left( \sqrt{3p}/2 \right). \quad (6.7)$$

*Example* 6.1: Let $p$ = 47; then $g(0)$ = 35 and from (6.6) $m$ = 5. Hence $g(m)$ = 5, which is the smallest generator. Yet, $g_1 = 11$ (3.1).

*Example* 6.2: Let $p$ = 83; then $g(0)$ = 62 and $m$ = 7. Therefore, $g(7)$ = 6, which is the third smallest one after the generators 2 and 5.

Yet, $g_1 = 2$ (3.1), is the smallest generator.

*Example* 6.3: Let now $p$ = 9522167; then $g(0)$ = 7141625; and from (5.2) $m$ = 2671. Therefore, $g(2671)$ = 4713. Yet, $g_1$ = 4942.

The procedure described above finds small generators. In some cases, it even provides the smallest generators; {as in *Example* 6.1}. However, it does not find the smallest generator for every safe prime $p$. In that case select

$$g := \min\left[ g_1, g(m) \right]. \quad (6.8)$$

## 7. Results of Computer Experiments

Several hundred computer experiments with the safe prime $p$ randomly-selected on interval ($10^7$, $10^{10}$) confirm that for every $p$ there exists a monotone-decreasing subset $S$ of generators $g(0)$, $g(1)$, $g(2)$, $\cdots$, $g(m)$ that satisfy the inequalities (6.3).

For instance, if $p$ = 9622580663, then the number $m$ of generators in the subset $S$ is equal 84952. The experiments also indicate that $g_1 \leq g(m)$ with frequency $f_1 \approx 0.382$ and $g(m) < g_1$ with frequency $f_2 \approx 0.618$. It implies that, if only one algorithm is used to compute a "small" generator, then $g(m)$ should be computed, because $g(m) < g_1$ with probability close to 62%.

*Remark* 6.2: Notice that $f_2^2 = f_1$, *i.e.*, these frequencies satisfy the golden ratio equality.

## 8. Acknowledgements

I express my appreciation to W. Gruver and R. Rubino for their helpful suggestions for improvement of the manuscript. I am also grateful to E. Gerda and Y. Polyakov for their assistance in computer experiments.

## 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] 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

[3] D. Knuth, "The Art of Computer Programming, Vol. 2: Seminumerical Algorithms", 3rd Edition, Addison-Wesley, Reading, 1998, pp. 18-21.

[4] C. F. Gauss, "Disquisitiones Arithmeticae", 2nd Edition, Springer, New York, 1986.

[5] P. Ribenboim, "The New Book of Prime Number Records", Springer, New York, 1996. doi:10.1007/978-1-4612-0759-7

[6] E. Bach and J. Shallit, "Algorithmic Number Theory: Vol. 1: Efficient Algorithms", MIT Press, Cambridge, 1996.

[7] V. S. Miller, "Use of Elliptic Curves in Cryptography", *Advances in Cryptography-CRYPTO* (*LNCS* 218), 1986, pp. 417-426.

[8] N. Koblitz, "Elliptic Curve Crypto-Systems", *Mathematics of Computation*, Vol. 48, No. 20, 1987, pp. 203-209. doi:10.1090/S0025-5718-1987-0866109-5

[9] A. Menezes, P. van Oorschot and S. Vanstone, "Handbook of Applied Cryptography", CRC Press, Boca Raton, 1997, pp. 162-164.

[10] B. Verkhovsky, "Integer Factorization of Semi-Primes Based on Analysis of a Sequence of Modular Elliptic Equations", *Int. J. of Communications, Network and System Sciences*, Vol. 4, No. 10, 2011, pp. 609-615.

 

# Appendix

## Alternative Search for Small Generators

Consider a safe prime $p \ge 11$;

$$z := (p-5)/2;$$

and let

$$\begin{aligned} h(2) &:= \left(p - z^2\right) \\ &= \left(14p - 25 - p^2\right)/4 \pmod p. \end{aligned} \tag{A.1}$$

Since every term in (A.1) is an integer, therefore now,

$$p^2 = 11p \pmod 4; \tag{A.2}$$

and

$$p^2 = 11p \pmod p = 0. \tag{A.3}$$

Hence $h(2) = (3p-25)/4 \pmod p$.

In general, for $z := \lfloor p - (2m+1) \rfloor / 2$; (A.4)

$$h(m) = \left\lfloor (4m-5)p - (2m+1)^2 \right\rfloor / 4 \pmod p. \tag{A.5}$$

Thus,

$$h(m) = \lfloor h(2) - m(m+1) + 6 \rfloor \bmod p. \tag{A.6}$$

Since $h(2) + 6 = g(0)$, therefore (A.6) and (6.4) provide the same results.

 