ECSC-128: New Stream Cipher Based on Elliptic Curve Discrete Logarithm Problem*

Khaled Suwais[1], Azman Samsudin[1]

[1] School of Computer Sciences, Universiti Sains Malaysia (USM)
Minden, 11800, P. Penang / Malaysia
{khaled, azman}@cs.usm.my

Abstract. ECSC-128 is a new stream cipher based on the intractability of the Elliptic Curve discrete logarithm problem. The design of ECSC-128 is divided into three important stages: Initialization Stage, Keystream Generation Stage, and the Encryption Stage. The design goal of ECSC-128 is to come up with a secure stream cipher for data encryption. ECSC-128 was designed based on some hard mathematical problems instead of using simple logical operations. In terms of performance and security, ECSC-128 was slower, but it provided high level of security against all possible cryptanalysis attacks.

Keywords: Encryption, Cryptography, Stream Cipher, Elliptic Curve, Elliptic Curve Discrete Logarithm Problem, Non-Supersingular Curve.

1 Introduction

Data security is an issue that affects everyone in the world. The use of unsecured channels for communication opens the door for more eavesdroppers and attackers to attack our information. Therefore, current efforts tend to develop new methods and protocols in order to provide secure communication over public and unsecured communication channels. In this paper we are interested in one class of the Symmetric Key cryptosystems, which is known as Stream cipher. The basic design is derived from the One-Time-Pad cipher, which XOR a sequence stream of plaintext with a keystream to generate a sequence of ciphertext. Currently, most of the stream ciphers are based on some logical and mathematical operations (e.g. XOR, shift, rotation, etc), while the others are based on the use of Linear Feedback Shift Registers (LFSR). Both types of ciphers are proven to provide high performance stream ciphers according to the results presented in [1], [2]. However, generally these ciphers are not able to reach the required highest level of security [3], [4], [5]. Various attacks managed to manifest the weaknesses of existing ciphers. Therefore, the direction of using simple logic and mathematical operations has shown the imperfection of some known stream ciphers.

* The authors would like to express their thanks to Universiti Sains Malaysia for supporting this study.

The idea behind our proposed stream cipher is based on the use of Elliptic Curve Discrete Logarithm Problem (ECDLP). The ECDLP along with other two intractable problems are considered the three most important pivots in current secure public-key cryptosystems. The other two problems are the Integer Factorization Problem (IFP), and the Discrete Logarithm Problem (DLP). The first use of ECDLP was in 1985 by Koblitz [6] and Victor Miller [7] independently. They proposed a new cryptosystem known as Elliptic Curve Cryptosystem (ECC), whose security depends on the discrete logarithm problem over the points on an Elliptic Curve.

The general idea of ECDLP rests on the difficulty of finding integer k given points kG and G, on Elliptic Curve E defined over a finite field with q elements (Fq), where in common, q is a large prime number or a field of characteristic two. In fact, the reason of choosing ECDLP as a core for our proposed stream cipher rests on the belief that ECDLP is harder than IFP and DLP [8]. The best known algorithm to solve the ECDLP has exponential complexity compared to the algorithm used to solve *IFP* which has sub-exponential complexity [9], [10]. In addition, the National Security Agency (NSA) announced *Suit B* at the *RSA* conference in 2005, which exclusively uses ECC for Digital Signature and Key Exchange [11]. Therefore, ECDLP with no doubt will enable our proposed stream cipher to reach the high level of security.

2 Related Work

Currently, many state-of-the-art stream ciphers seem to have appeared. The obvious relation between current ciphers enabled us to classify them into two important distinguishable categories. The first category is called the *logical and bit manipulation-based stream ciphers*. This category includes those ciphers whose construction design rests on some logical operations (Shift, XOR, etc), and uses some other substitution techniques. Examples of ciphers that belong to this category are: RC4 [12] and SEAL [2]. The second category includes all ciphers whose design rests on the use of the LFSRs. This category is labeled as *LFSR-based stream ciphers*, and it includes SNOW [13], LILI-128 [4], and other stream ciphers.

2.1 Logical and Bit-Manipulation-Based Stream Cipher

The concept of constructing a stream cipher aims to deliver a secure algorithm against all kind of attacks. In 1987, Ron Rivest of RSA Security designed RC4 [12], which later became the most widely used stream cipher. RC4, as any other stream ciphers, generates a stream of bits (keystream) which will be XOR'ed with a stream of plaintext bits to produce the ciphertext bits. Generating the keystream in RC4 requires a permutation array S of all 256 possible byte using the RC4 Key Scheduling Algorithm (*KSA*). At the present time, RC4 is not recommended for use in new application. As appears in [14], several weaknesses of the *KSA* algorithm of RC4 can be summarized in two points. The first weakness is the existence of massive classes of weak keys. These classes enabled the attackers to determine a large number of bits of *KSA* output by using a small part of the secret key. Thus, the initial outputs of the weak keys are disproportionally affected by a small portion of key bits. The second weakness rests on a related key vulnerability. Therefore, RC4 falls off the standards for secure cipher.

Another example that belongs to this category is SEAL. It was introduced by Rogaway and Coppersmith as a fast software encryption algorithm [2]. The notion of SEAL is based on a mapping of a 32-bit string n to an L-bit string under the control of a 160-bit length secret key of. In general, the length of the output, L, is limited to 64 KB. A few years later, a cryptanalysis attack presented by Handschuh and Gilbert [15] showed that an attack is capable of distinguishing SEAL from a random function by using 2^{30} computations. In addition, the attack has sufficient power to derive some bits from SEAL secret tables.

2.2 LFSR-Based Stream Ciphers

A LFSR is a hardware device made up by registers, which is capable to hold one value at a time [16]. The values are elements from a chosen field Fq (for binary field $q = 2$, or $q = 2^w$ for extension field of the binary field). The purpose of using LFSR was to deliver a stream cipher with high performance property. In most cases, the immediate output of LFSR is not acceptable as a keystream generator since the value production process is done in linear fashion. The linearity of the operations makes the direct output of LFSR easy to predict. Thus it requires a combination of other techniques to reach a reasonable level of security.

SNOW and LILI-128 are two examples of stream ciphers based on the used of LFSRs. The design of SNOW rests on the combination between LFSR and a Finite State Machine (FSM) with a recurrence defined over the Galois Field $GF(2^{32})$. It accepts both 128-bit and 256-bit keys. The keystream is generated by combining the values from LFSR with values in FSM in time t:

$$Z_t = \mathrm{FSM}_{out\_t} \oplus \mathrm{LFSR}_{out\_t} \tag{2.1}$$

SNOW was broken by two attacks known as Guess-and-Determine (GD) attack [5]. These two attacks utilize the relationships between internal values and relationship employed to form the keystream symbols from the internal values. The GD attack determines many internal values based on its guessing of some internal values using the above relationship (Equation 2.1) with time complexity $O(2^{256})$.

LILI-128 is another example of LFSR based stream ciphers designed by Dawson et al [4]. The key length used in LILI-128 is 128 bits. It uses two binary LFSRs labeled by LFSR_c (39 bits), LFSR_d (89 bits), and two functions to generate a pseudorandom binary keystream sequence. The keystream generator includes two components: *Clock Controller* and *Data Generator*. The clock controller uses LFSR to generate an integer sequence which controls the clocking of the LFSR within the data generator component. Time-Memory-Tradeoff attack has been used to attack the design of weak keys are disproportionally affected by a small portion of key bits. The second weakness rests on a related key vulnerability. Therefore, RC4 falls off the standards for secure cipher.

Another example that belongs to this category is SEAL. It was introduced by Rogaway and Coppersmith as a fast software encryption algorithm [2]. The notion of SEAL is based on a mapping of a 32-bit string *n* to an *L*-bit string under the control of a 160-bit length secret key of. In general, the length of the output, *L*, is limited to 64 KB. A few years later, a cryptanalysis attack presented by Handschuh and Gilbert [15] showed that an attack is capable of distinguishing SEAL from a random function by using 2^{30} computations. In addition, the attack has sufficient power to derive some bits from SEAL secret tables.

2.2 LFSR-Based Stream Ciphers

A LFSR is a hardware device made up by registers, which is capable to hold one value at a time [16]. The values are elements from a chosen field Fq (for binary field $q = 2$, or $q = 2^w$ for extension field of the binary field). The purpose of using LFSR was to deliver a stream cipher with high performance property. In most cases, the immediate output of LFSR is not acceptable as a keystream generator since the value production process is done in linear fashion. The linearity of the operations makes the direct output of LFSR easy to predict. Thus it requires a combination of other techniques to reach a reasonable level of security.

SNOW and LILI-128 are two examples of stream ciphers based on the used of LFSRs. The design of SNOW rests on the combination between LFSR and a Finite State Machine (FSM) with a recurrence defined over the Galois Field $GF(2^{32})$. It accepts both 128-bit and 256-bit keys. The keystream is generated by combining the values from LFSR with values in FSM in time *t*:

$$Z_t = \mathrm{FSM}_{out\_t} \oplus \mathrm{LFSR}_{out\_t} \qquad (2.1)$$

SNOW was broken by two attacks known as Guess-and-Determine (GD) attack [5]. These two attacks utilize the relationships between internal values and relationship employed to form the keystream symbols from the internal values. The GD attack determines many internal values based on its guessing of some internal values using the above relationship (Equation 2.1) with time complexity $O(2^{256})$.

LILI-128 is another example of LFSR based stream ciphers designed by Dawson et al [4]. The key length used in LILI-128 is 128 bits. It uses two binary LFSRs labeled by LFSR_c (39 bits), LFSR_d (89 bits), and two functions to generate a pseudorandom binary keystream sequence. The keystream generator includes two components: *Clock Controller* and *Data Generator*. The clock controller uses LFSR to generate an integer sequence which controls the clocking of the LFSR within the data generator component. Time-Memory-Tradeoff attack has been used to attack the design of word (sub-keystream). These sub-keystream bits are XOR'ed with the plaintext bits (one word of plaintext at a time) to generate a stream of ciphertext. The design of this cipher is divided into three main stages: Initialization Stage (IS), Keystream Generation Stage (KGS), and Encryption Stage (ES).

- Initialization Stage (IS):

At this stage, the input key λ (128-bit) determines the initial value of point P_1 on the curve E, and the initial value of the key k (128-bit length). The value of k is generated by:

$$k = (\lambda \oplus V) << r \qquad \textbf{(3.4)}$$

where V is constant value, and r is variable value based on the second half of the bits from $\lambda \oplus$ V. The symbol (<<) refers to the left shift operation by r position. The generated key k will be later multiplied by point P_1 to produce new point P_2 on the curve E. The Elliptic Curve E is formed by computing function (3.5) defined over Fp (where p is a publicly known large irreducible polynomial).

$$E\colon y^2 = x^3 + a\,x^2 + \mathrm{b} \quad (\text{where } a, b \text{ elements in } Fp) \qquad \textbf{(3.5)}$$

Given the curve E and the value of λ, the point generation on E is straightforward. The process of generating P_1 includes embedding λ on E with the correct (x,y) coordinates. The embedding process involves solving Equation (3.5) over an irreducible (prime) polynomial field. The solution (as implemented in [18]) requires two matrices (*Tmatrix*, *Smatrix*) and one vector (*Trace_Vector*). The *Smatrix* used to compute the square root of a polynomial modulo irreducible polynomial. With the assumption that the solution to Equation (3.5) exists, the *Tmatrix* is formed from the set of basis vectors which are summed together. The *Trace_Vector* is used to determine if the solution exists.

- Keystream Generation Stage (KGS):

The generated point P_1 from the previous stage (IS) is used as base point to generate a keystream K_s of 320-bit length. The idea of generating K_s is based on multiplying the key k by the point P_1 in a process known as *Point Multiplication.* Point multiplication is the idea behind the ECDLP. The rule of multiplying P_1 by k over Fq is stated as follow [19]:

Given point $P_1 = (\mathrm{x}_1, \mathrm{y}_1)$ and integer k, then $P_2 = kP_1 = (\mathrm{x}_2, \mathrm{y}_2)$ is new point on the curve E generated by multiplying P_1 by itself k times.

The point multiplication process consists of a finite set of point additions and doubling operations ($kP_1 = P_1 + P_1 + P_1 + \ldots + P_1$). Adding P_1 to itself (doubling operation) computed in equation (3.6).

$$\Theta = x + \frac{x}{y}$$

$$X_2 = \Theta^2 + \Theta + b \qquad \textbf{(3.6)}$$

$$Y_2 = x_1 + (\Theta + 1)\, x_1$$

The addition will be performed k times, and the success of the multiplication operation means we have a new point P_2 with new coordinates (x_2, y_2) on E. The procedure of generating K_s based on the points generated from the previous stage is illustrated in **Fig 1**.

- Encryption Stage (ES):

The process of encrypting the input stream of plaintext involves some actions to control the number of times we need to generate new keystreams. The length of the keystream K_s generated from (KGS) is 320-bit. K_s is divided into ten words (32-bit for each sub-keystream). Each sub-keystream is XOR'ed with 32-bit of plaintext (*Pt*) to produce a stream of ciphertext (*Ct*). The encryption routine will check the status of the unused bits of K_s to determine if the encryption process needs more keystream bits. When the routine returns zero unused bits, new value for integer k will be generated by adding 1 to the previous values k. In turn, k will be converted into new point P_1 as discussed in the first stage (IS). Subsequently, the encryption routine will invoke the keystream generator (in KGS) to generate new 320-bit keystream based on the multiplication of the new values of P_1 and k. The generated keystream can be XOR'ed with new plaintext bits to generate new stream of ciphertext. The structure of Encryption Stage (ES) is portrayed in **Fig 2.**

Given P_1, P_2 on $E(Fp)$ and Input λ

- Compute $k = (\lambda \oplus V) << r$
- P_1 = embed (λ) on E
- $P_2 = k P_1$ on E
- Transform P_2 into K_s

Fig. 1. The Functionality of *IS* and *KGS* stages

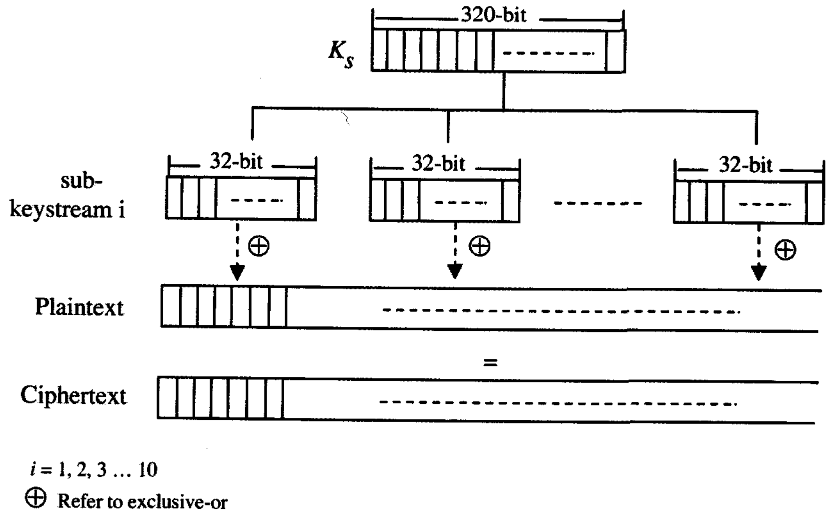

Fig. 2. The Design of the Encryption Stage (ES).

As described above, the Initialization Stage (IS) will control the keystream generator in (KGS) by feeding it with the point P_1 and the integer k. These two parameters will be multiplied in (KGS) to create a new point P_2, which forms the new keystream K_s after transformation. The encryption process is controlled by the first two stages (IS and KGS) in terms of generating new points and later generating new keystream to perform the encryption on the given stream of plaintext. The overall structure of our proposed stream cipher is divided into three related stages as appears in **Fig 3**.

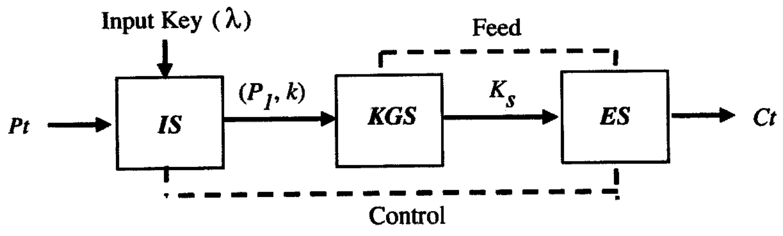

Fig. 3. Overall Design of ECSC-128.

3.2 The Implementation of ECSC-128

ECSC-128 is a word-oriented stream cipher. All operations are done on 32-bit word, which makes ECSC-128 applicable for 32-bit platforms. Unlike other stream ciphers, the key setup stage (IS) requires one XOR operation on one 128-bit variable which can be executed in less than 1 µs on Pentium 4 2.00 GHz.

ECSC-128 encrypts 27108 byte/second for a given input key (128-bit) on Pentium 4 CPU (2.00 GHz). Compared to other stream ciphers, the encryption rate of ECSC-128 is considered slow. The reason is the time that is required to calculate Θ when we perform point addition operation. The calculation of Θ requires an inversion operation over Fp, which is the most time consuming among EC operations. However, there is a high possibility of increasing the speed of ECSC-128 by improving Θ calculation which requires inversion operation. Point Multiplication on E is the core of ECSC-128, since point multiplication is an ECDL problem in EC. For that reason, ECSC-128 performance can be improved by making more efforts to enhance the operations involved in calculating the value of Θ.

4 Security Analysis

The cryptographic strength of ECSC-128 lies in the difficulty of solving the intractability of ECDLP. In this section we analyze ECSC-128 in terms of the security design properties as well as its strength against the cryptanalysis attacks.

4.1 Brute-Force Attack

ECSC-128 uses *non-Supersingular* curves instead of using supersingular curves. The reason is that supersingular curves are prohibited by all standards as mentioned previously. For a given curve E, the total number of points that satisfy equation E

(which is known as the order of the curve #*E*) must contain big prime number. Hasse's Theory [18] stated that the range of #*E* lies in the range:

$$2^n - 2\sqrt{2^n} + 1 \leq \#E \leq 2^n + 2\sqrt{2^n} + 1 \qquad \textbf{(4.1)}$$

where *n* is the number of bits in the field. For ECSC-128, *n*=160, and therefore the range is between:

$1.4615016373309029182036836237902 \times 10^{48}$

and

$1.4615016373309029182036860416418 \times 10^{48}$

The quantity of numbers in the range of the above equation is 2^{68} = $2951479051793528258 56\times 10^{20}$. Therefore, brute force search for the largest prime within this range is infeasible, since matching such range to a particular curve is non-trivial. From another perspective, the length of the input key for ECSC-128 is 128-bit, which is secure against brute force search as well. The length of the key used in ECSC-128 was chosen to achieve the expected security against brute force search attack and to satisfy the required balance of the difficulty between the forward and the inverse operations. The forward operation of ECSC-128 is the point multiplication ($P_2 = kP_1$) on *E*, which is fast enough compared to its inverse operation (retrieve *k* given P_1 and P_2). In general, the difficulty of the inverse operation against the key length must be exponential for secure cryptographic primitives as portrayed in **Fig 4** [20].

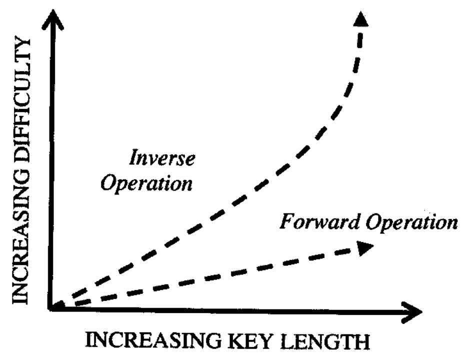

Fig. 4. Difficulty of forward, inverse operation against key length

According to the intractability of ECDLP, applying brute force is infeasible. Logically, finding *k* from $P_2 = kP_1$ means performing repeated addition operation starting from P_1, $2P_1$, until we find kP_1. Then, we double P_1 then we add P_1 to the doubled point $2P_1$ to find $3P_1$, then $3P_1$ to P_1 finding $4P_1$ and so on. This attack is avoided in ECSC-128 by using a large prime field. This means that; the number of possible values of k is large. Searching through the bearable values of k will take hundreds of years using current available processors.

4.2 Baby-Step-Giant-Step Attack

The baby-step-giant-step algorithm is used to solve the discrete logarithm problem in arbitrary groups [21]. Let n be the order of a point P (order(P): is the number of times we can add the point to itself until we find the point at infinity) on E. This attack requires a memory for $\sqrt{n}$ points and time complexity of $O\left(\sqrt{n}\right)$ [22], which is infeasible due to the use of a large prime number n in ECSC-128.

4.3 Pollard's rho Attack

Pollard's rho is a collision-based algorithm which was first proposed by J. Pollard in 1978 [23]. It relies on finding collisions between elements of a cyclic group. Thus, Pollard's algorithm can be applied to a group of points on E. The idea of Pollard's attack is to reveal the solution to ECDLP by computing the trail of points on E. This computation is known as a *walk*, and it is achieved by generating a sequence of points using an iterating function. The success ratio of this attack is related to calculating the square root of the size of the cyclic group. In fact, according to the time complexity of Pollard's algorithm $O\left(\sqrt{\#E}\right)$, ECSC-128 is secured against this attack. However, ECSC-128 uses a large enough value of $\#E$ (see section 3), which make Pollards rho attack infeasible.

5 Conclusion

In this paper we present a new stream cipher (ECSC-128) based on the intractability of the Elliptic Curve discrete logarithm problem (ECDLP). ECSC-128 is divided into three main stages: Initialization Stage (IS), Keystream Generation Stage (KGS), and the Encryption Stage (ES). Complete descriptions of ECSC-128 stages, implementation, and an overview of possible attacks have been discussed. ECSC-128 has changed the current direction of stream ciphers from using simple logic and mathematical operations into a new direction of using complex mathematical problems that are computationally infeasible.

References

1. M. Boesgaard, M. Vesterager, T. Pedersen, J. Christiansen, and O. Scavenius: Rabbit: A New High-Performance Stream Cipher, In Proc. Fast Software Encryption 2003, volume 2887 of LNCS. Springer, 2003.
2. P. Rogaway, D. Coppersmith: A Software-Optimized Encryption Algorithm, Journal of Cryptology, vol. 11, num, 4, pp. 273-287, 1998.
3. Scott R. Fluhrer, I. Mantin and A. Shamir: Weaknesses in the Key Scheduling Algorithm of RC4. Selected Areas in Cryptography. pp1 – 24, 2001.
4. E. Dawson, A. Clark, J. Golic, W. Millan, L. Penna, L. Simpson: The LILI-128 Keystream Generator, NESSIE submission, In Proc. of the first Open NESSIE Workshop 2000.
5. Christophe De Canniere: Guess and Determine Attack on SNOW. Nessie public report. NES/DOC/KUL/WP5/011/a. www.cryptonessie.org. 2001
6. N. Koblitz: Elliptic curve cryptosystems, in Mathematics of Computation, Vol. 48, pp. 203–209. 1987.
7. V. Miller, Use of elliptic curves in cryptography, CRYPTO 85, 1985.
8. Advanced Technology Center: Elliptic Curve Cryptography. TATA Public Report. http://www.atc.tcs.co.in/esecurity/research.html. 2006.
9. N. Jha and I. Sengupta: A generalized scheme for multiparty secure access using elliptic curve cryptography, In Proc. of the 6th Intl. Conference on Information Technology, pp. 119-124. Bhubaneswar, India. December 2003.
10. Zhu, H: Survey of Computational Assumptions Used in Cryptography Broken or Not by Shor's Algorithm. School of Computer Science. Montreal, McGill University. Master in Science. 2001
11. NSA: NSA Suite B Cryptography, http://www.nsa.gov/ia/industry/crypto_suite_b.cfm. 2005.
12. R. Rivest: The RC4 Encryption Algorithm. RSA Data Security Inc. Document No. 003-013005-100-000000. Mar. 1992.
13. P. Ekdahl, and T. Johansson: SNOW-A New Stream Cipher, In Proc. of First NESSIE Workshop, Heverlee, Belgium. 2000.
14. S. Fluhrer, I. Mantin, and A. Shamir: Weaknesses in the key scheduling algorith of RC4. In Proc. 8th workshop on selected Areas in cryptography, LNCS 2259. Springer-Verlag. 2001.
15. H. Handschuh and H. Gilbert: X^2 cryptanalysis of the SEAL encryption algorithm. In Proc. of the 4th Workshop on Fast Software Encryption, volume 1267 of Lecture Notes in Computer Science, pages 1-12. Springer-Verlag, 1997.
16. P. Ekdahl: On LFSR based Stream Ciphers Analysis and Design. Information Technology. Lund, Lund University. Ph.D. 2003.
17. S. Babbage: Cryptanalysis of LILI-128. NESSIE Public Report, www.cosic.esat.kuleuven.ac.be/nessie/reports. 2001.
18. M. Rosing: Implementing elliptic curve cryptography. Manning Publications Co., Greenwich, CT, USA. 1999.
19. A. J. Menezes: Elliptic Curve Public Key Cryptosystems. Boston: Kluwer Academic Publishers. 1993.
20. Certicom: An Elliptic Curve Cryptography (ECC) Primer. Catch the Curve Series. http://www.certicom.com/download/aid-317/WP-ECCprimer.pdf. 2005
21. D. Shanks: Class number, a theory of factorization and genera. In Proc. Symp. Pure Math. 20, pages 415--440. AMS, Providence, R.I. 1971.
22 T.E. Gueneysu, C. Paar, J. Pelzl: On the security of Elliptic Curve Cryptosystems against Attacks with Special-Purpose Hardware, 2nd Workshop on special-purpose Hardware for Attacking Cryptographic Systems - SHARCS 2006, April 3-4. 2006
23. J. M. Pollard: Monte Carlo methods for index computation mod p. mathematics of computation, 32(143):918-924. July 1978.