ELLIPTIC CURVE CRYPTOGRAPHY IN SMALL DEVICES

Andrzej Chmielowiec
Enigma Information Security Systems Sp. z o.o.
8 Cietrzewia Str., 02-492 Warsaw, Poland
E-mail: achmielowiec@enigma.com.pl

ABSTRACT

Today many aspects of our jobs and lives are controlled by electronic devices. This is mostly visible in the field of access management. There exist many solutions that increase security and eliminate unauthorised access to critical resources. All of them play a crucial role in such applications as:

- *electronic payment and banking,*
- *electronic document circulation and access,*
- *intelligent buildings.*

In many such solutions digital signatures are a good mechanism to control user rights and access modes. Unfortunately such digital algorithms as RSA and DSA are inefficient in small devices (computation time is relatively long). So if we want to use classical signature algorithms, we have to use devices with special cryptographic accelerators. This automatically increases the cost and can be accepted only in the case of large scale production. For medium and small scale production we want to propose the ECDSA (Elliptic Curve Digital Signature Algorithm) algorithm implemented on a general purpose 8-bit microcontroller.

The main goal of this article is to show that ECDSA is a very good alternative to classical algorithms and can be effectively implemented in devices with limited resources. First we give a brief outline of digital signature algorithms and key lengths which are recommended by NIST. Then we show theoretical aspects of ECDSA and methods of its implementation. At the end we present a small USB token which can be used to generate ECDSA signatures. It is an example of a low cost device based on:

- *8-bit microcontroller ATmega8 (8KB program memory, 1KB SRAM memory),*
- *USB-UART bridge CP2103.*

The described solution illustrates how microcontroller peripherals can be used to implement communication interfaces and a truly random bit generator. We also consider other possible applications of the proposed architecture and give estimated cost calculations for our examples.

1 INTRODUCTION

Last few years gave us great progress in the Elliptic Curve Cryptography (ECC) standarization process. We can see effect of this in such documents as: RFC 4492 [4], ANSI X9.62 [2], ANSII X9.63 [1], IEEE 1363 [7], FIPS 186-3 [8], SEC 1 [10] and SEC 2 [11].

One can wonder if we really need a different cryptosystem than good, old RSA. The point is that 1024 bits modulus is not enough today. Extending key length up to 2048 and 3072 bits is a problem in hardware implementations with limited resources (smart cards, secure tokens). Elliptic Curves Cryptography offers the same security level with shorter key lengths.

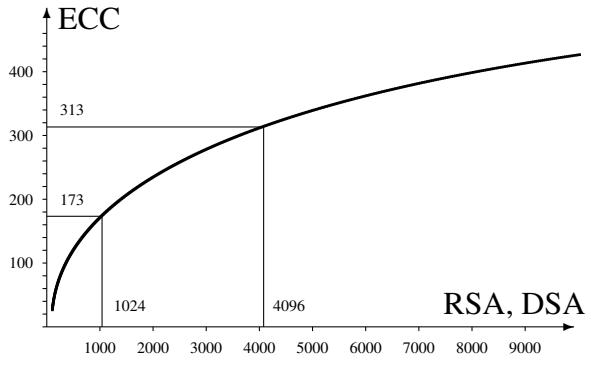

Figure 1: Ilustration of equivalent key lengths for ECC and RSA, DSA.

Figure 1 shows correlation between key lengths for ECC and RSA. We can see that RSA

with 1024 bits modulus is equivalent to ECC with only 173 bits key and RSA with 4096 bits modulus is equivalent to ECC with 313 bits key. This disproportion is based on the hardness of factorization and elliptic curve discrete logarithm problem. Complexity of the first is subexponential and fastest algorithm (Number Field Sieve) solves factorization problem in asymptotic time

$$O(\exp(\log N^{\frac{1}{3}} \log\log N^{\frac{2}{3}})),$$

where N is RSA modulus. Discrete logarithm on elliptic curve seems to be much harder than integer factorization. The best algorithm (Polard's ρ) is exponential and works in assymptotic time

$$O(\sqrt{N}),$$

where N is an integer interpreted as ECC private key. Presented comparison explains why key sizes of RSA algorithm grow much faster than for ECC. We should also note that smaller keys need less computational power and are well suited for small devices.

1.1 Key lengths and NIST recommenations

The following table [9, p. 63] shows how long keys should be used to obtain the desired security level.

Security level	Symmetric algorithm	Public key size	
		RSA/DSA	ECC
80	Skipjack	1024	160
112	3DES	2048	224
128	AES-128	3072	256
192	AES-192	7680	384
256	AES-256	15360	512

This comparison was done by NIST and is based on actual state of the art in cryptanalysis. We can observe that 128 bit security can be achieved by using RSA with 3072 bits modulus. Signature generation using such key is rather slow even on large machines and can cause significant delays in server communication. This of course implies that today we don't have technology to implement so large keys in smart cards and secure tokens.

The next table is also NIST recommendation [9, p. 66] and presents what key length should be used to protect information during next two decades.

Security lifetimes	Security level	Minimum key size	
		RSA/DSA	ECC
up to 2010	80	1024	160
up to 2030	112	2048	224
beyond 2030	128	3072	256

2 ELLIPTIC CURVE CRYPTOGRAPHY

Elliptic curves are known in mathematis over hundred years but their applications to cryptography were found twenty years ago by Diffie, Hellman and ElGamal. These authors in [5] and [6] showed how general group structures can be used to safely exchange secrets and generate signatures. Today standards described as ECDH (Elliptic Curve Diffie-Hellman) and ECDSA (Elliptic Curve Digital Signature Algorithm) are in fact formal modifications of ideas proposed by Diffie, Hellman and ElGamal.

2.1 Basics of elliptic curves

Elliptic curve over field K is defined as a set of solutions of non-singular Weierstrass equation, given by

$$E : Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6.$$

Non-singularity means that for all points $P \in E$ we have

$$\frac{\partial E}{\partial X}(P) \neq 0 \text{ or } \frac{\partial E}{\partial Y}(P) \neq 0.$$

We also add special point $\mathcal{O}$ called point at infinity. As shown in [12], points of elliptic curve E with point at infinity form a group. If field K is finite then also group is finite, and can be used as a base of Diffie-Hellman protocol or ElGamal signature scheme.

Elliptic curve given by equation E is in the most general form. If we assume that characteristic of field K is greater than 3 then equation

$$E_1 : Y^2 = X^3 + a_4X + a_6$$

represents all curves up to isomorphism. We have simmilar situation if field characteristic is 2. In this case all curves can be represented by one of the following equations

$$E_2 : Y^2 + XY = X^3 + a_2X^2 + a_6,$$

$$E_3 : Y^2 + a_3Y = X^3 + a_4X + a_6.$$

Curves expressed in the form of E_1 and E_2 are used in cryptography, while curves of the form E_3 have very bad properties and can't be used in practice.

2.2 Elliptic curve group law

If we consider elliptic curves over real numbers then the group law has very nice graphic illustration. Figures 2 and 3 show how elliptic curve point addition and point doubling can be done.

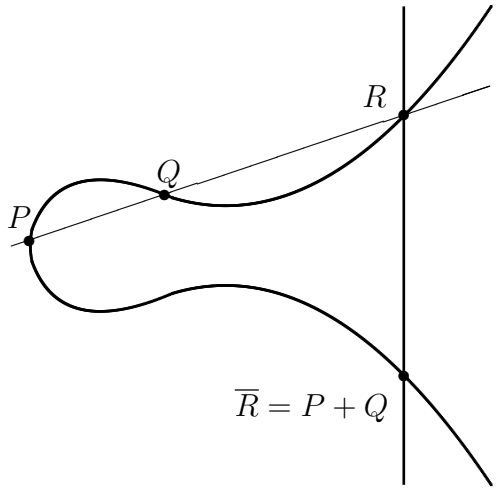

Figure 2: Elliptic curve point addition over $\mathbb{R}$.

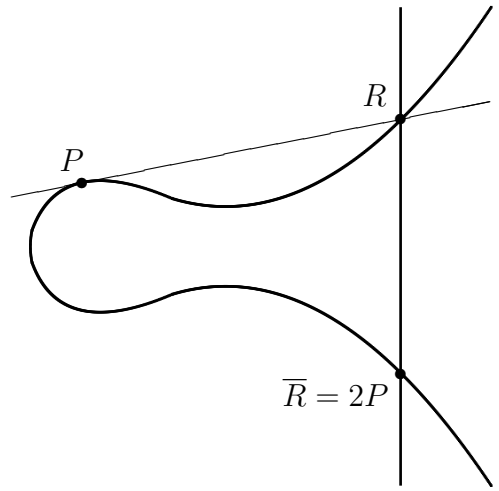

Figure 3: Elliptic curve point doubling over $\mathbb{R}$.

In general case, there exist algebraic formulas which are used to find result of group law. Assume now that elliptic curve is given by equation E as described at the beginning of the section. Let $P_1 = (x_1, y_1) \in E$ and $P_2 = (x_2, y_2) \in E$ be such that $P_1 \neq -P_2$ then

$$-P_1 = (x_1, -y_1 - a_1 x_1 - a_3),$$

and

$$P_3 = (x_3, y_3) = P_1 + P_2 \neq \mathcal{O}.$$

Where coordinates x_3 and y_3 are computed using following formulas

$$\begin{aligned} x_3 &= \lambda^2 + a_1\lambda - a_2 - x_1 - x_2, \\ y_3 &= -(\lambda + a_1)x_3 - \mu - a_3. \end{aligned}$$

If $x_1 \neq x_2$ then we take

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1},\ \mu = \frac{y_1 x_2 - y_2 x_1}{x_2 - x_1},$$

and if $x_1 = x_2$ then we have to take

$$\lambda = \frac{3x_1^2 + 2a_2 x_1 + a_4 - a_1 y_1}{2y_1 + a_1 x_1 + a_3},$$

$$\mu = \frac{-x_1^3 + a_4 x_1 + 2a_6 - a_3 y_1}{2y_1 + a_1 x_1 + a_3}.$$

Presented formulas are not used in practical implementations because of the division. This operation is relatively expensive and has very bad influance on the algoritm complexity. It is the main reason why people use so called *projective* or *weight projective* (*Jacobian*) coordinates. The main idea of this representation is to store denominators as third variable and go back to affine coordinates when all computations are done. Details about projective representation can be found in [7].

Point addition and doubling are used to find arbitrary point multiple. To obtain point $P = [m]G$ we can use double-add method.

1. $P \leftarrow \mathcal{O}$
2. while $m \neq 0$ do
3. if $m \equiv 1 \mod 2$ then
4. $P \leftarrow P + G$
5. $G \leftarrow [2]G$
6. $m \leftarrow \lfloor m/2 \rfloor$
7. return P

Above algorithm allows to compute multiple m of given point G in $\log m$ doublings and $\frac{1}{2}\log m$ additions in average case. There exist faster algorithm which uses the fact that finding opposite elliptic curve point is very easy and cheap.

This method converts multiple m to its *non-adjacent form* (NAF). In classical binary representation natural numbers are treated as sums of powers of 2 with weight equal to 0 or 1. In the case of NAF number is represented as a sum of powers of 2 with weights belonging to set $\{-1, 0, 1\}$. The following algorithm [3, p. 67] converts number in classical representation to number in NAF.

1. Let $m = \sum_{k=0}^{l-1} m_k 2^k$, $m_k \in \{0, 1\}$
2. $c_0 \leftarrow 0$
3. for $k = 0$ to l do
4. $\quad c_k \leftarrow \lfloor (m_k + m_{k+1} + c_k)/2 \rfloor$
5. $\quad s_k \leftarrow m_k + c_k - 2c_{k+1}$
5. return NAF$(m) = (s_l s_{l-1} \dots s_0)$

This leads to a little bit more complicated algorithm.

1. $P \leftarrow \mathcal{O}$
2. $s \leftarrow$ NAF(m)
3. for $k = 0$ to l do
4. $\quad$ if $s_k = 1$ then
5. $\quad\quad P \leftarrow P + G$
6. $\quad$ else if $s_k = -1$ then
7. $\quad\quad P \leftarrow P + (-G)$
8. $\quad G \leftarrow [2]G$
9. return P

Presented method is faster than classical double-add and computes multiple m of given point G in $\log m$ doublings and $\frac{1}{3} \log m$ additions in average case.

2.3 Elliptic curve digital signature algorithm (ECDSA)

Now we present short description of ECDSA algorithm and underline points hardest to realize in practical implementation. In our consideration we assume that we have some elliptic curve E, its point G of order n and private key d_U.

1. Select random number $k \in [1 \dots n-1]$ and compute $R = (x_R, y_R) = [k]G$.
2. Convert field element x_R to integer $\overline{x}_R$.
3. Set $r = \overline{x}_R \mod n$. If $r = 0$ go to step 1.
4. Derive integer e from message hash H as described in [10].
5. Use private key d_U to compute

$$s = k^{-1}(e + rd_U) \mod n.$$

   If $s = 0$ then return to step 1.

6. Output signature $S = (r, s)$.

Hardest part of the above algorithm is step 1 were we have to

- compute generator multiple $[k]G$,
- generate random number k.

The problem is not how to perform this steps at all, but how to do it securely. We must remember that if an attacker can derive ephemeral key k then he can use signature components r, s and e to compute secret key d_U

$$d_U = r^{-1}(sk - e) \mod n.$$

3 IMPLEMENTATION AND RESULTS

3.1 Hardware description

Our implementation is based on three basic circuits:

1. microcontroller ATMega8 (8kB of internal flash for program memory, 1kB of internal SRAM, 0.5kB of internal EEPROM),
2. microcontroller CP210X (USB-UART bridge),
3. external EEPROM memory (at least 32kB) to store multiples of generator and public objects (certificates, public keys etc.).

All these components are small and can be easily mounted on 20mm $\times$ 50mm PCB board.

3.2 Random bit generator

Random bit generator is one of the most important modules of our implementation. It is necessary to generate static user keys and ephemeral keys during ECDSA signature generation. To be absolutely sure that no one will be able to predict random sequence we should implement truly random bit generator. This is really hard problem in digital devices because of the deterministic character of all operations. Fortunately ATMega8 is equipped with 10 bit

Analog Digital Converter (ADC) which can be used to measure microcontroller voltage noise.

ADC was constructed to convert analog signal to its digital representation. This module compares given input voltage V_{IN} with reference voltage V_{REF}. Result of this operation is stored as a 10-bit number

$$\left\lfloor \frac{V_{IN}}{V_{REF}} \cdot 1024 \right\rfloor .$$

All we have to do is set ADC probing frequency (about 120kHz in our case) and read less significant bit of the conversion as our random seed. It should be strongly underlined, that applied frequency is very close to its maximum value. Using much lower or much higher frequencies is not recommended. In the first case conversion may be too accurate and we will get same results each time. In the second one module will not be able to compare V_{IN} with V_{REF} which also can give same results each time. Here is sample C code written for avr-gcc compiler which ilustrates how generator collects seed.

```
volatile octet_t adc_low;
volatile octet_t adc_high;

/* ADC interrupt handler. */
SIGNAL(SIG_ADC)
{
  adc_low = ADCL;
  adc_high = ADCH;
  ADCSRA &= ~((1 << ADEN) | (1 << ADIE));
}

/* Random bit generator. */
void_t
rbg_generate_bits(octet_t *dst, len_t len)
{
  /* ADC configuration see ATMEL ATMega8
   * documentation. */
  ADMUX = 0x40;
  ADCSRA = (1 << ADPS2) | (1 << ADPS1)
    | (1 << ADPS0);

  /* ... */

  /* Start single AD conversion. */
  ADCSRA |= (1 << ADEN) | (1 << ADIE);
  ADCSRA |= (1 << ADSC);

  /* Waiting for end of conversion. */
  while ( (ADCSRA & (1 << ADEN)) != 0 );

  /* Now adc_low contains low octet of
   * conversion. */

  /* ... */
}
```

Experimental results shows that in sequences generated using our implementation there is more zeroes than ones. Probability of 0 is significantly larger than probability of 1. We will propose two ways to reduce this efect. First method is based on elementary probability calculus.

Suppose that our generator gives bit $b = 0$ with some constant probability $P(b = 0) = \frac{1}{2} + \varepsilon$ where $|\varepsilon| < \frac{1}{2}$. If ε is too large, then such generator does not pass any statistical test and its application to cryptographic purposes is not a good idea. Fortunately we can easily combine its outputs and get generator with arbitrary small value of $|\varepsilon|$. The following proposition gives theoretical background to our construction.

Proposition 1 Suppose that we have random variable b such that

$$\begin{aligned} P(b = 0) &= \frac{1}{2} + \varepsilon, \\ P(b = 1) &= \frac{1}{2} - \varepsilon. \end{aligned}$$

If $b_1, \ldots, b_n$ is sequence of independent random variables with above probability distribution then for $B_n = b_1 \oplus \cdots \oplus b_n$ we have

$$\begin{aligned} P(B_n = 0) &= \frac{1}{2} + \varepsilon(2\varepsilon)^{n-1}, \\ P(B_n = 1) &= \frac{1}{2} - \varepsilon(2\varepsilon)^{n-1}. \end{aligned}$$

Proof: We will use mathematical induction to proof this properties. First induction step for $n = 1$ is obvious because of the assumprion that $P(b = 0) = \frac{1}{2} + \varepsilon$ and $P(b = 1) = \frac{1}{2} - \varepsilon$. To perform second induction step we will use the following fact

$$\begin{aligned} &P(B_n = 0) \\ &= P(b_1 \oplus \cdots \oplus b_n = 0) \\ &= P(b_1 \oplus \cdots \oplus b_{n-1} = 0)P(b_n = 0) + \\ &P(b_1 \oplus \cdots \oplus b_{n-1} = 1)P(b_n = 1) \\ &= P(B_{n-1} = 0)P(b_n = 0) + \\ &P(B_{n-1} = 1)P(b_n = 1). \end{aligned}$$

Now we can combine previous formula with induction assumption and get

$$P(B_n = 0)$$

$$= P(b_1 \oplus \cdots \oplus b_{n-1} = 0)P(b_n = 0) +$$
$$P(b_1 \oplus \cdots \oplus b_{n-1} = 1)P(b_n = 1)$$
$$= \left(\frac{1}{2} + \varepsilon(2\varepsilon)^{n-2}\right)\left(\frac{1}{2} + \varepsilon\right) +$$
$$\left(\frac{1}{2} - \varepsilon(2\varepsilon)^{n-2}\right)\left(\frac{1}{2} - \varepsilon\right)$$
$$= \frac{1}{2} + \varepsilon(2\varepsilon)^{n-1}.$$

■

Above proposition gives us recipe how to produce better random sequence. Of course we should remember that this solution slows down generator frequency. If there is a need to have generator wich gives 0 with probability belonging to the interval $\left[\frac{1}{2} - \delta, \frac{1}{2} + \delta\right]$ then we should XOR at least

$$n = \left\lceil 1 + \frac{\ln(\delta) - \ln(\varepsilon)}{\ln(2\varepsilon)} \right\rceil$$

bits of unbalanced sequence. This means that generator will be n times slower.

Second idea how to reduce redundancy of random bit generator is heuristic and based on information theory. Suppose that we have sequence of n bits which contains only $k < n$ bits of entropy. This means that only about k bits of n are really random and the entropy of this sequence doesn't have its maximum value. Information theory tells us that in such case we should be able to use compresion algorithm and reduce sequence length. Of course it can't be algorithm for standard data compresion as ZIP because of standarized format and file headers. But using methods aplied in hash functions seems to be good idea. Our generator implements SHA-1 compression function which reduces seed size from 512 to 160 bits.

3.3 Generation of keys and signatures

Having good random bit generator we can implement such cryptographic features as key generator and signature algorithm. Most important operation during execution of above functions is computation of elliptic curve generator multiple. Using standard double-add or double-add-sub methods is not recommended because of the Simple Power Analysis (SPA) attack. It is based on observation that device power consumption during elliptic curve point doubling is radically different than during point addition or subtraction. The following algorithm presents typical implementation which can be broken using this type of side channel attack.

1. $P \leftarrow \mathcal{O}$
2. while $m \neq 0$ do
3. if $m \equiv 1 \mod 2$ then
4. $P \leftarrow P + G$
5. $G \leftarrow [2]G$
6. $m \leftarrow \lfloor m/2 \rfloor$
7. return P

One can see that there are two expensive instructions

1. point doubling $G \leftarrow [2]G$ and
2. point addition $P \leftarrow P + G$.

Unfortunately point addition is executed only if adequate bit of m is set. If an attacker can measure current consumed by device during computations then he will be able to detect all bits of every computed multiple. This leads to the conclusion that he will know all keys generated by the device (both static and ephemeral).

To avoid problem with SPA we decided to implement algorithm of point multiplication based on precomputed generator multiples. During device initialization process there are determined values of $G_0 = G, G_1 = [2]G, G_2 = [4]G, \ldots, G_{\lfloor \log_2 n \rfloor} = [2^{\lfloor \log_2 n + 1 \rfloor}]G$ where n is generator order. As you can see this eliminates point doubling in algorithm loop and protects our implementation from SPA.

1. $k \leftarrow 0$
2. $P \leftarrow \mathcal{O}$
3. while $m \neq 0$ do
4. if $m \equiv 1 \mod 2$ then
5. $P \leftarrow P + G_k$
6. $m \leftarrow \lfloor m/2 \rfloor$
7. $k \leftarrow k + 1$
8. return P

In the same way we can modify double-add-sub method.

1. $k \leftarrow 0$
2. $P \leftarrow \mathcal{O}$

3. $s \leftarrow \text{NAF}(m)$
4. for $k = 0$ to l do
5. if $s_k = 1$ then
6. $P \leftarrow P + G_k$
7. else if $s_k = -1$ then
8. $P \leftarrow P + (-G_k)$
9. $k \leftarrow k + 1$
10. return P

We should remember that NAF representation of number m can have one more digit than classic binary form. This means that above algorithm needs points $G_0, \ldots, G_{\lfloor \log_2 n \rfloor}$ and $G_{\lfloor \log_2 n+1 \rfloor} = [2^{\lfloor \log_2 n+2 \rfloor}]G$.

3.4 Running times

Two main components of our implementation are

1. random bit generator which uses SHA-1 compression function to increase entropy of generated bitstring,

2. algorithm finding generator multiples which is based on method described at the end of previous subsection.

Time results at 8MHz clock frequency are presented in the following table.

Times at 8MHz		
ECC domain	Signing time [s]	Key generation time [s]
secp160r1	1.6	1.3
secp192r1	2.1	1.7
secp224r1	2.8	2.4
secp256r1	4.6	4.1

Next table shows binary code size of small operating system which can be used to sign given message digest.

System code size	
ECC domain	Code size [bytes]
secp160r1	7080
secp192r1	6978
secp224r1	7060
secp256r1	7194

4 CONCLUSIONS

Presented solution can be used as cryptographic USB token which implements ECDSA functionality. The following list shows its potential applications

- creating Virtual Private Networks (VPN) based on TLS protocol and X509 certificates,

- used as digital key to login to computer system,

- random bit generator which is used to seed pseudorandom software generator.

From practical point of view this device is a low cost project.

1. We don't need any special and expensive software. Standard GCC distribution supports most of the Atmel AVR microcontrollers and allows access to any hardware feature from C language level.

2. Generated application may be sent to the device using low cost In System Programmer (ISP).

Total cost of all electronic elements used to construct device is about 15 USD (retail price).

References

[1] ANSI. Public key cryptography for the financial services industry: Key agreement and key transport using elliptic curve cryptography. *ANSI X9.63*, 1999.

[2] ANSI. Public key cryptography for the financial services industry: The elliptic curve digital signature algorithm. *ANSI X9.62*, 2005.

[3] I. Blake, G. Seroussi, and N. Smart. *Elliptic Curves in Cryptography*. Cambridge University Press, 1999.

[4] N. Bolyard, V. Gupta, C. Hawk, and B. Moeller. Elliptic curve cryptography cipher suites for transport layer security. *RFC 4492*, 2006.

[5] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 1976.

[6] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 1985.

[7] IEEE. Standard specifications for public key cryptography. *IEEE 1363*, 2000.

[8] NIST. Digital signature standard. *FIPS 186-3*, 2006. Draft version.

[9] NIST. Recommendation for key management - part 1: General. *NIST Special Publication 800-57*, 2006. Draft version.

[10] SECG. Elliptic curve cryptography. *SEC 1*, 2000. <http://www.secg.org>.

[11] SECG. Recommended elliptic curve domain parameters. *SEC 2*, 2000. <http://www.secg.org>.

[12] J. H. Silverman. *The Arithmetic of Elliptic Curves*. Springer-Verlag, 1986.