Implementation of Elliptical Curve Cryptography Based Diffie-Hellman Key Exchange Mechanism in Contiki Operating System for Internet of Things

Prateek Thapar[1] and Usha Batra[2]

[1]Research scholar, SOES, GD Goenka University, Gurugram, Haryana, India prateekthapar@yahoo.com
[2] Dean, SOES, GD Goenka University, Gurugram, Haryana, India Dr.ushabatra@gmail.com

***Correspondence:** Prateek Thapar Email: prateekthapar@yahoo.com

ABSTRACT- Wireless Sensor Networks have gradually upgraded to Internet of Things (IoT) of embedded devices wherein the constrained devices have been connected directly onto the Internet. This transformation has not only facilitated the expansion in connectivity and accessibility of the sensor network but has also enabled one sensor network to interact with other through Internet. Security of IoT devices has been researched extensively. The challenge to transform the complex cryptographic algorithms into lighter and faster has kept researchers on their toes. Contiki-OS is one of the purest implementations of 6LoWPAN and IEEE 802.15.4. That makes Contiki-OS lightest and therefore preferred OS for implementation on ultra-low power sensor nodes. Elliptical Cryptography has proved to be the choice of most of the security researchers for constrained devices. However, there is very limited literature available on implementation of Elliptical Cryptography on Contiki-OS. The open-source libraries available for security implementation have not found to be supporting Cooja simulator of Contiki-OS. In this research work we demonstrate improved results in processing the Elliptical Cryptography Based implementation of Diffie-Hellman Key exchange mechanism in Contiki-OS using Cooja simulator. SECP160K1 curve has been implemented and the results in terms of ECDH computational time have been compared with previously published research works. This research demonstrates improved results in Cooja simulator than previous known results on hardware providing a leap ahead in efficiency of implementation.

Keywords: Internet of Things, IEE802.15.4, Security, Contiki-OS, Elliptic Curve Cryptography, Diffie Hellman Key Exchange.

ARTICLE INFORMATION
Author(s): Prateek Thapar and Usha Batra;
Special Issue Editor: Dr. S. Gopalakrishnan;
Received: 01/05/2022; **Accepted**: 09/06/2022; **Published**: 30/06/2022;
e-ISSN: 2347-470X;
Paper Id: 0422SI-IJEER-2022-20;
Citation: 10.37391/IJEER.100245
Webpage-link:
https://ijeer.forexjournal.co.in/archive/volume-10/ijeer-100245.html

This article belongs to the Special Issue on **Intervention of Electrical, Electronics & Communication Engineering in Sustainable Development**

Publisher's Note: FOREX Publication stays neutral with regard to jurisdictional claims in Published maps and institutional affiliations.

1. INTRODUCTION

Smart devices like Kitchen Electronics, Security cameras and fences, Health Monitoring Systems, Connected Vehicles and Industrial Automation are latest trends in the field of Internet of Things currently and planned in near future. However, the IoT devices used in such applications are predominantly powered and therefore can afford to use IEEE 802.11 protocol for connecting to the Internet. Whereas, there are ultra-low power IoT devices which have bare minimum power to sustain the constrained requirement of resources so as to accomplish a considerable life. Such IoT devices / motes / constrained nodes have minimal resources to be consumed by security protocol.

There are numerous lightweight libraries implementing symmetric cryptography and hash functions specifically designed for embedded devices. However the keys for asymmetric cryptographic functions are large integers and demand large memory and processing cycles for deployment. This makes the implementation of protocols difficult in an efficient way.

A well-known example for a 16-bit platform targeted towards IoT applications is the MSP430 family of ultra-low power microcontrollers from Texas Instruments [1]. The C standard mathematical library of MSP430 includes mathematical functions defined in <math.h>. This library is widely used for microcontrollers as it is part of C compiler. It provides only basic math functions like trigonometric, exponentiation and logarithmic functions [2].

Our literature survey indicates that there are very few attempts to build numerical computation libraries that can perform heavy computations on microcontrollers. Texas Instruments provide IQMath and QMath libraries for its MSP430 and MSP432 devices. These libraries contain a collection of mathematical routines for C programmers. However, this collection is restricted only to basic mathematical functions such as trigonometry and algorithmic functions [3].

Another peculiarity among the libraries is that they have limited support to Contiki-OS. Tiny ECC [4] utilizes C code of large integer operations in RSAREF 2.0 [5] by porting it to NesC code on TinyOS for implementation. TinyDTLS [6] is implementation of Elliptic Curve Cryptography (ECC) on an 8-bit microcontroller but the implementation on GitHub is only for 32-bit architecture and with pre-shared key mode. Other libraries that are available online are NanoECC [7], BearSSL [8], FlexibleECC [9], MicroECC[10], WolfSSL [11] and PolarSSL [12].

This research work is focused on implementation of our mathematical engine on ultra-low power microcontroller MSP430 using WisMote of Cooja simulator in Contiki-OS. Diffie Hellman Key Exchange is discussed in *Section 2*. *Section 3* comprises of elliptic curves and mathematical operations on elliptic curves in finite field. The experimental setup and results are thereafter explained in *Section 4* and the research work is concluded in *Section 5*.

2. DIFFIE HELLMAN KEY EXCHANGE

A message in the pure form, without any alterations or additions, conveying the intended meaning is known as message in plain text. This plain text message, when in a coded form, having worked upon with intentions to hide the message from third party, is called cipher text. The process of conversion of plain text to cipher text is call encryption and the reversal from cipher text to plain text is called decryption. The study of algorithms or schemes used for encryption and decryption is called cryptography. The encryption algorithms have been developed through research and are publicly known. However, the key used for encryption is the guarded secret that prevents unauthorized decryption of the cipher text.

The knowledge of encryption algorithm used and the key can expose the cipher text to third party. Since the computational power of computers is so high, the brute force programs use almost all the encryption algorithms to convert the cipher text to plain text if the key is known. This process would not even take minutes. Therefore, key management is one of the most focused research topic under cryptography.

Whitfield Diffie and Martin E. Hellman under a paper “New Directions in Cryptography” [13] introduced the concept of key exchange that is popularly known as Diffie-Hellman key exchange mechanism.

Typically to the cryptographic texts, considering Alice (A) as source or first party and Bob (B) as destination or second party, where A needs to communicate with B in form of a message (m). The message m in plain text is converted to cipher text using a cryptographic algorithm and a secret key (k). The challenge now is to communicate the cipher text along with the secret key. In case, both are communicated on same line or method of communication, there is high probability of the third party or eavesdropper Eve (E) to get both cipher text and secret key. Such communications required a separate secure communication line for transmission of secret key for keeping the message secure.

Diffie-Hellman introduced the concept of Public Key Cryptosystem and public key distribution system to handle the requirement of a separate secure line for communication of secret key. The public key cryptosystem works with two keys with each party known as his private key and public key. The private key as the name suggests is not shared with anyone while the public key is freely distributed to public. The set of keys have following properties:

(a) It is feasible to compute inverse pairs of keys known as private and public key.
(b) Text encrypted by using public key can be decrypted by only using private key and vice a versa.
(c) It is not feasible to derive private key from known public key and the algorithm.

Diffie-Hellman proposed that the private key is the secret key that remains with the owner of the key while the public key is freely distributed among others. The key pair as per the properties is such that anything encrypted by private key can only be decrypted by use of its paired public key and vice a versa. Therefore, this scheme provides not only the confidentiality of the message but also serves the purpose of authentication due to its property. The key pair is also used as digital signature for the purpose of authentication in client server architecture.

3. ELLIPTICAL CURVES

An elliptic curve over a field (K) is a nonsingular cubic curve in two variables, f (X,Y)=0, with a K-rational point (which may be a point at infinity). The field K is usually taken to be the complex numbers (C), reals (R), rationals (Q), algebraic extensions of (Q), p-adic numbers (Q_p), or a finite field [14]. The curve is defined by a math function and is symmetric along X-Axis. The curve is described by following equation:-

$$y^2=x^3+ax+b$$
$$\text{Where } 4a^3+27b^2\neq 0 \qquad (1)$$

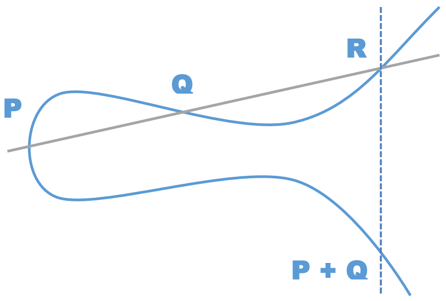

Figure 1: Adding Point (P and Q) on an Elliptic Curve

This equation is called *Weierstrass normal form* for elliptic curves and the condition is required to exclude singular curves. Additionally, negative of a point is defined as the reflection of that point on the curve in x-axis. Since, the equation of elliptic curve has only two solutions for any given x value, the resultant of addition of a point and its reflections is Zero. This Zero (0) denotes the Point at Infinity, also known as Identity Element. Hence the solution to the equations now may be written as:-

$$E = \{(x, y): y^2= x^3+ ax + b\} \cup \{O\} \qquad (2)$$

The points on elliptic curve form an Abelian group as following groups of mathematics are applicable to them:

CLOSURE: If a, b are members of group then $a+b$ is also member of group.

ASSOCIATIVITY: (a + b) + c = a + (b + c)
IDENTITY ELEMENT: a + 0 = a = 0 + a
INVERSE: a + b = => b= -a
COMMUTATIVITY: a+b = b+a

In event of three aligned points on an elliptic curve, their sum is Zero. Hence:

P+Q+R=0 => P+Q= -R (reflection of R) *(3)*

This equation enables us to derive the sum of two points on the curve. Elliptical Curve cryptography requires to perform below mentioned two basic arithmetic operations on the points on curve (P and Q to get R):-

1. Algebraic Addition: This addition is carried out using following formulae:-

P = (x_p, y_p) Q = (x_q, y_q) R = (x_r, y_r)

Slope $m = \frac{yp-yq}{xp-xq}$ if P and Q are distinct

$m = \frac{3x^2+a}{2y}$ if P = Q

$$x_r = m^2 - x_p - x_q \quad (4)$$
$$y_r = m(x_r - x_p) - y_p \quad (5)$$
'or' $y_r = m(x_r - x_q) - y_q$

2. Scalar Multiplication: This function is multiplication of a natural number with a point on the curve. Scalar multiplication of nP(n times point P) can be performed by n times algebraic addition of point P. But this would result in order O(2^K) which is computationally heavy. Hence, in order to reduce the computational load we use Double-and-Add method [15].

3.1 Double and Add Algorithm for Point Multiplication

In order to use Elliptic Curve for implementation of Diffie-Hellman Key Exchange, we require the private key to be large integer of the order of 256 bits. This integer is multiplied to the primitive element that raises its complexity exponentially. An efficient method of performing this multiplication is Double and Add algorithm as explained here under:

Step 1: Convert integer into Binary format.

$$mP = P + P + \cdots + P \in E(F_p)$$

Step 2: Start from right most bit to the left most bit in steps of one bit.

$m = m_0 + m_1.2 + m_2.2^2 + \cdots + m_r.2^r \; where \; m_0, \ldots, m_r \in \{0,1\}$.

Step 3: Double the point in each step.

Step 3(a): In addition to step 3, Add the point when the encountered bit value is 1.

$$mP = m_0P + m_1.2P + m_2.2^2P + \cdots + m_r.2^rP$$

Result: At the end of the loop, resultant multiplicative is achieved.

Through use of Double and Add method to compute multiples of a point, we perform $\log_2(m)$ doublings and $\frac{1}{2}\log_2(m)$ additions.

3.2 Hasse's Theorem

Given an Elliptic Curve E modulo p, the number of points on the curve is denoted by #E and is bounded by:-

$$p + 1 - 2\sqrt{p} \leq \#E \leq p + 1 + 2\sqrt{p} \quad (6)$$

Since we are dealing with large prime number p, the value $1 - 2\sqrt{p}$ or $1 + 2\sqrt{p}$ is too small to be considered. Hence, it can be easily considered that the no. of points on curve are approximately equal to the range of prime p. Consequently, in order to select a curve of this 2^{256} elements, a prime number having 256 bits is required.

3.3 Elliptic Curve Based Discrete Logarithmic Problem (ECDLP)

Authors in [13] were the first to propose use of Discrete Logarithmic Problem as a One Way Function in pursuit of creation of a secure connection over a public channel. One way function may be considered such a function f: X->Y in which it is feasible to compute f(x) given x, however, it is infeasible to calculate x with f(x) =y. Hence its use in public key cryptosystem where the public key can be shared along with the cipher text on a public channel without compromising the security as the private key is secure with the owner.

In terms of Elliptic Curve Cryptography, the Discrete Logarithmic Problem is to find the integer 'd' where

$1 \leq d \leq \#E$ (E being the Elliptic Curve)

Integer 'd' is the one used to compute a point 'T' on the curve by performing the group operation on primitive point 'P' such that

$$dP = T$$

ECDLP when applied in Public Key Cryptosystem represents 'd' as private key (integer) and 'T' as the point on Elliptic Curve (E) whose equation and primitive point 'P' is shared on the public channel.

3.4 Elliptic Curve Over Finite Field

Finite Field is a set of finite elements like set of integers modulo p, where p is a prime number denoted as $\mathbb{F}p$. Implementation of public key cryptosystem uses a pair of algorithms constituting Discrete Logarithmic problem over a finite field of elements. A finite field with modulo p, where p is a prime number, consists of all integers from 0 to p-1.

Addition and multiplication are the two binary operations that are applicable on finite field. These operations are associative and commutative. These operations follow 'Clock Arithmetic' as the remainder (modulo) starts from 0 and increments till p-1 and repeats thereafter.

The formulae for the algebraic addition or doubling of points under finite fields may be represented as hereunder:

Slope:

if P ≠ Q $\quad m = \frac{yp-yq}{xp-xq} \mod p$ *(7)*

if P = Q $\quad m = \frac{3x^2+a}{2y} \mod p$ *(8)*

Resultant Point Coordinates:

$$x_r = m^2 - x_p - x_q \bmod p \qquad (9)$$

$$y_r = m(x_r - x_p) - y_p \bmod p \qquad (10)$$

$$\text{or } y_r = m(x_r - x_q) - y_q \bmod p \qquad (11)$$

However, the division in a finite field is not a simple operation. In order to perform a ÷ b in $\mathbb{F}p$, we need to first find the multiplicative inverse of b over $\mathbb{F}p$ and then multiply it with a as in a ÷ b = a.b $^{-1}$

The multiplicative inverse of a number in finite field can be found using the Extended Euclidean Algorithm.

3.5 Extended Euclidean Algorithm

Extended Euclidean Algorithm [16] aims to find two integers ‘u’ and ‘v’ which fulfill

$$\gcd(a, b) = ua + vb$$

Recursive iterations of this down till you get *a = 1*
and then the reverse substitution gets the value $\frac{a}{b}\ (mod\ p)$
Since we have calculated till *a = 1*
Hence equations forms as $\frac{1}{b}\ (mod\ p)$ => b^{-1}(mod p) *(12)*

3.5.1 Function Extended Euclidean

Input:
a Positive integer argument.
b Positive integer argument.
Output:
k The greatest common divisor of a and b

```
Algorithm:
(u,v) Integers such that ua+vb=k.
assert $a \geq 0 \wedge b \geq 0$
(c, d) ← (a, b)
($u_c,v_c,u_d,v_d$) ← (1,0,0,1)
while c ≠ 0
        do
        Invariant:
        $u_c a+v_c b=c \wedge u_d a+v_d b=d$
        q ← ⌊ d/c⌋ (c, d) ← (d − qc, c)
        ($u_c,v_c,u_d,v_d$)←($u_d - qu_c, v_d - qv_c, u_c, v_c$)
        od
return d, ($u_d$, $v_d$)
```

3.6 Mathematical Implementation

Contiki-OS is a pure ‘C’ language implementation so as to keep the minimum footprint in the flash memory of constrained nodes. The constrained nodes have very limited memory and processing power. Therefore, in order to keep the program implementation light and use minimum power while operation, C programming language is used. This limits the use of heavy mathematical operations on the constrained node. However, the mathematical operations involved in cryptographic implementations are quite bulky in terms of utilization of large prime numbers and also programming complex mathematical equations. Python programming language is high level and fast programming language, which is generally used for solving the cryptographic problems, but it requires much higher computational power and corresponding memory, which is not available in constrained nodes. Hence, usage of python programming language is not recommended.

Therefore, the using pure C operations are mandatory in programming the functionalities in constrained nodes. C programming language supports only two byte integers with a limitation till 65,535 in case of unsigned integer. However, the cryptographic operations require prime numbers ranging from 128 bits to 2048 bits or higher. This necessitates the requirement of usage of an efficient cryptographic implementation, which would provide higher complexity in solving the cryptographic equations with shorter prime numbers as keys.

Hence the merit of using Elliptic Curve cryptography in constrained nodes. Still, the issue of programming the mathematical operations remains a challenge. This challenge is mitigated through use of structures of integers posing as building blocks for large numbers used in these cryptographic operations. The operations are performed using Hex values in form of strings as input and output. These hex values are there after converted into structures having array of integers representing the values.

The inbuilt mathematical functions of addition, subtraction, multiplication, division, modulo etc. cannot be used due to the large numbers being represented as array of integers. Due to this limitation all the basic functions, over and above the other necessary equations, have to be programmed in this implementation. Hence, a major portion of the implementation of ECDH is actually implementation of mathematical functions using large numbers as array of integers. This increases the difficulty of programming by multifold, still keeping the memory footprint low and lower requirement of computational power and hence low usage of battery and thereby increasing the life of a constrained sensor node.

4. EXPERIMENTAL SETUP AND RESULTS

The mathematical engine has been programmed in C language so as to work as a shared library in the Contiki operating system for experimental purpose. The mathematical engine along with the code for ECDH can be used in other operating systems for IoT as most of the operating systems are implemented using C language.

MSP430 is a 16-bit RISC processor containing 16 fully addressable single-cycle CPU registers. However, four of sixteen registers are special purpose registers *i.e.* program counter, stack pointer, status register and constant generator. Therefore, only twelve register can be used for implementation. Additionally, MSP430 does not have multiplication instruction that poses a huge disadvantage in attaining processing time in this processor especially in a simulator.

We have chosen SECP160K1 curve [17] for our experimentation with standard parameters as shown in *Table 1.*

Table 1: Parameters of SECP160K1

Parameter	Value
a	0
b	7
p	0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF FEFFFFAC73
g	0x3B4C382CE37AA192A4019E763036F4F 5DD4D7EBB,0x938CF935318FDCED6BC2 8286531733C3F03C4FEE
n	0x100000000000000000001B8FA16DFAB9 ACA16B6B3
h	1

We have carried out the experiment in Contiki-OS version 2.7. Contiki-OS has a simulator Cooja which emulates the motes and we have tested the ECDH on WisMote. WisMote is a MSP430 series 5 microcontroller with 16-bit RISC architecture having RAM of 16Kb and ROM of 256Kb.

The mathematical engine was run as a shared library and the code for ECDH was embedded in the broadcast example. *c* program available in prime examples of Contiki-OS version 2.7. The time taken for running the ECDH function was compared with other similar implementation of ECDH in previous researches and comparison is shown below as *Table 2.*

Table 2: Comparison of Execution Time in Clock Cycles

Author [Reference]	Clock Cycles
Liu and Ning [18]	25,290,000
Wenger and Werner [19]	17,559,862
Hinterwa ̈lder et al [20]	12,625,570
Szczechowiak et al [21]	11,520,000
Wenger [22]	11,442,840
This work	10,810,680

We determined the time in clock cycles as is available on Cooja simulator. Two nodes of WisMote were taken in this network with 100% visibility to each other so that no packets are dropped in the communication. As shown in the *Table 2*, the results in this research work are faster than the previous published researches available with the authors.

However, it is pertinent to note that the experimentation has been carried out in simulator which is run at 1000% speed for 10 times at a random time gap. Additionally, a disclaimer, Contiki-OS version 2.7 is not fully tested at maximum clock rate. It can run stably under Low Power Mode 1 (LPM 1). In case version of Contiki-OS, which is stable for full computational speed, is used then much better results can be seen.

5. CONCLUSION

This paper brings out implementation of ECDH using SECP160K1 curve on ultra-low power microcontroller mote i.e WisMote on Cooja simulator of Contiki-OS version 2.7. The research work has demonstrated high speed computation of ECDH by improving the total time consumed. This research work is focused on achieving higher and efficient security for constrained nodes of IoT device network.

REFERENCES

[1] D.Dang, M. Plant and M. Poole. Wireless connectivity for the Internet of Things (IoT) with MSP430 microcontrollers (MCUs). Texas Instruments white paper at https://www.ti.com/lit/wp/slay028/slay028.pdf?ts=1638301587285&ref_url=https%253A%252F%252Fwww.google.com%252F

[2] Sanchez, J.; Canton M.P. Microcontrollers: High Performance Systems and Programming; CRC Press: Boca Raton, FL, USA, 2018.

[3] Texas Instruments Incorporated. MSP430 IQMathLib user guide, Texas Instruments Incorporated: Dallas, Tx, USA, 2015.

[4] A. Liu and P. Ning, "TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks," 2008 International Conference on Information Processing in Sensor Networks (ipsn 2008), 2008, pp. 245-256, doi: 10.1109/IPSN.2008.47.

[5] RSA Laboratories. RSAREF: A cryptographic toolkit (version 2.0) March 1994.

[6] Ismail, M. Aiman, and Thomas C. Schmidt. "A DTLS Abstraction Layer for the Recursive Networking Architecture in RIOT." arXiv preprint arXiv: 1906.12143 (2019).

[7] Szczechowiak, Piotr, et al. "NanoECC: Testing the limits of elliptic curve cryptography in sensor networks." European conference on Wireless Sensor Networks. Springer, Berlin, Heidelberg, 2008.

[8] Silde, Tjerand. "Comparative study of ECC libraries for embedded devices." Norwegian University of Science and Technology, Tech. Rep (2019).

[9] Wenger, E., Unterluggauer, T., Werner, M.: FLECC GitHub repository. https://github.com/IAIK/flecc_in_c. Last accessed November 30, 2021.

[10] MacKay, K.: Micro-ECC GitHub repository. https://github.com/kmackay/micro-ecc. Last accessed November 30, 2021.

[11] WolfSSL: GitHub repository. https://github.com/wolfSSL/wolfssl. Last accessed November 30, 2021.

[12] ARMmbed: mbed TLS GitHub repository. https://github.com/ARMmbed/mbedtls. Last accessed November 30, 2021.

[13] Whitfield Diffie and Martin Hellman. New directions in cryptography. IEEE transactions on Information Theory, IT-22(6):644-654, November 1976.

[14] Weisstein, Eric W. "Elliptic Curve." From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/EllipticCurve.html

[15] Silverman, J. H. (2006). An introduction to the theory of elliptic curves. link: http://www.math.brown.edu/jhs/Presentations/WyomingEllipticCurve.Pdf

[16] Iliev, Anton, and Nikolay Kyurkchiev. "The faster extended Euclidean algorithm." Collection of scientific works from conference. 2018.

[17] SEC2: Recommended Elliptic Curve Domain Parameters. Certicom Research. https://www.secg.org/SEC2-Ver-1.0.pdf. Last visited on November 30, 2021.

[18] A. Liu and P. Ning. TinyECC: A configurable library for elliptic curve cryptography in wireless sensor networks. In Proceedings of the 7th International Conference on Information Processing in Sensor Networks (IPSN 2008), pp. 245–256. IEEE Computer Society, 2008.

[19] E. Wenger and M. Werner. Evaluating 16-bit processors for elliptic curve cryptography. In Smart Card Research and Advanced Applications — CARDIS 2011, vol. 7079 of Lecture Notes in Computer Science, pp. 166–181. Springer Verlag, 2011.

[20] G. Hinterwa ̈lder, C. Paar, and W. P. Burleson. Privacy preserving payments on computational RFID devices with application in intelligent transportation systems. In Radio Frequency Identification Security and Privacy Issues — RFIDSec 2012, vol. 7739 of Lecture Notes in Computer Science, pp. 109–122. Springer Verlag, 2012.

[21] P. Szczechowiak, L. B. Oliveira, M. Scott, M. Collier, and R. Dahab. NanoECC: Testing the limits of elliptic curve cryptography in sensor networks. In Wireless Sensor Networks — EWSN 2008, vol. 4913 of Lecture Notes in Computer Science, pp. 305–320. Springer Verlag, 2008.

[22] E. Wenger. Hardware architectures for MSP430-based wireless sensor nodes performing elliptic curve cryptography. In Applied Cryptography and Network Security — ACNS 2013, vol. 7954 of Lecture Notes in Computer Science, pp. 290–306. Springer Verlag, 2013.